emSSL

Secure Sockets Library

User Guide & Reference Manual

Document: UM15001
Software Version: 3.20.0
Revision: 0
Date: November 19, 2026



A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2015-2026 SEGGER Microcontroller GmbH, Monheim am Rhein / Germany

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

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

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

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: November 19, 2026

Software	Date	By	Description
3.20.0	261119	TISC	Fix spelling and grammar mistakes. Chapter *Introduction to emSSL* • Added *System architecture*. Chapter *Exploring emSSL* • Updated section *Using a PC to try emSSL* to reflect changes to the sample application. • Updated section *Scanning websites with emSSL* to reflect changes to the sample application. Chapter *Using emSSL* • Updated section *ROT13 server* to adapt the example to the new sample transport APIs. • Updated section *Installing certificates and keys* (replaced Bin2C utility with PrintCert utility). Chapter *Configuring emSSL* • Updated the list of available compile-time configuration switches.
3.18.0	260820	TISC	Chapter *Using emSSL* • Added section *Defining a transport API*. Chapter *API reference* • Added `SSL_IS_BLOCKING_FUNC`. • Added `SSL_SESSION_GetRecordSizeLimit()`. • Added `SSL_SESSION_SetRecordSizeLimit()`. Chapter *Configuring emSSL* • Added Wall clock time function for sample transport APIs.
3.16.0	260720	TISC	Chapter *Using emSSL* • Fix various typos and sample commands. • Update recommended key lengths. • Add section *Audit events*. Chapter *Configuring emSSL* • Add section Audit event delivery.
3.14.0	260616	TISC	Update to latest emCrypt software version (2.60.0). Chapter "*Configuring emCrypt*" • Introduced the new concept of *Optimization goals*. • Added configuration instructions for random number generators. • Described compile-time and runtime configuration in more detail. • Improved configuration instructions in general.
3.12.1	260415	PC	Update to latest software version.
3.12.0	260112	RH	Update to latest software version.
3.11.1	251222	PC	Chapter "OS Integration" • Removed `SSL_OS_Sleep()`. • Added `SSL_OS_GetTime32()`.
3.11.0	250929	PC	Chapter "API reference" • Added `SSL_SESSION_Process()`. • Updated sections for nonblocking mode.
3.10.0	250721	PC	Chapter "API reference" • Added `SSL_SESSION_SetPSK()`.
3.8.0	250425	PC	Update to latest software version.
3.6.0	240712	PC	Update to latest software version.
3.4.0	240516	PC	Update to latest software version.
3.2.0	231110	PC	Update to latest software version.
3.0.0	230331	PC	Beta release for TLS 1.3.
2.64.0	220501	PC	Chapter "API Reference" • Updated description of `SSL_SESSION_Connect()`.
2.62.0	220201	PC	Update to latest software version.
2.60.0	210423	PC	Chapter "API Reference"

 © 2015-2026 SEGGER Microcontroller GmbH

Software	Date	By	Description
			• Added SSL_ROOT_CERTIFICATE_AddDER(). • Added SSL_ROOT_CERTIFICATE_LoadDER(). Chapter "OS Integration" • Added SSL_OS_Sleep(). Chapter "API reference" • Added section "Data types".
2.58.3	200714	PC	Update to latest software version.
2.58.2	200629	PC	Update to latest software version.
2.58a	190405	PC	Update to latest software version.
2.58	190311	PC	Update to latest software version.
2.56	181010	PC	Chapter "Configuring cryptography" • Added iMX RT10xx DCP coprocessor.
2.54a	180704	PC	Update to latest software version.
2.54	180621	PC	Chapter "Configuring cryptography" • Added STM32 AES coprocessor. • Added STM32 HASH coprocessor.
2.52	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 message signers".
2.50	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	170728	PC	Chapter "API Reference" • Added "Preprocessor symbols". • Added SSL_SUITE_GetCipherName(). • Added SSL_SUITE_GetIanaName(). • 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	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.

 © 2015-2026 SEGGER Microcontroller GmbH

Software	Date	By	Description
			Chapter "OS Integration" • Added reference pages for SSL_OS_* functions.
2.30	160802	PC	Chapter "Configuring emSSL". • 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	160530	PC	Chapter "Configuring emSSL". • Added CRYPTO configuration.
2.12	160316	PC	Chapter "Configuring emSSL". • Added Camellia cipher suites.
2.11	151201	PC	Chapter "Configuring emSSL". • Added CCM and CCM-8 cipher suites.
2.10	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	150521	PC	Initial release.

 © 2015-2026 SEGGER Microcontroller GmbH

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

 © 2015-2026 SEGGER Microcontroller GmbH

Table of contents

1 Introduction to emSSL 14
1.1 What is emSSL? 15
1.2 Design goals 16
1.3 Features 17
1.4 Package content 18
1.4.1 Include directories 18
1.5 System architecture 19
1.5.1 Overview 19
1.5.2 Information flows 20
1.5.3 Secrets 20
1.5.4 Security parameters 20
2 Exploring emSSL 22
2.1 Using a PC to try emSSL 23
2.2 Scanning websites with emSSL 26
2.3 Moving to embedded hardware 27
3 Using emSSL 29
3.1 Sample applications 30
3.1.1 A note on the samples 30
3.1.2 Where to find the sample code 30
3.2 What to expect 31
3.3 ROT13 server 33
3.3.1 Application entry 33
3.3.2 Accepting SSL connections 33
3.3.3 Serving the connection 35
3.3.4 Reading a line of text 36
3.3.5 Transforming characters using ROT13 37
3.3.6 SSL_ROT13Server.c complete listing 38
3.4 ROT13 client 41
3.4.1 Application entry 41
3.4.2 Making SSL connections 41
3.4.3 Sending requests 42
3.4.4 SSL_ROT13Client.c complete listing 44
3.5 Certificates 47
3.5.1 Types of certificate 47
3.5.2 Self-signed certificates 47
3.6 Creating certificates using OpenSSL 48
3.6.1 Creating RSA certificates 48

 © 2015-2026 SEGGER Microcontroller GmbH

3.6.2 Creating DSA certificates 49
3.6.3 Creating ECDSA certificates 50
3.7 Installing certificates and keys 52
3.7.1 Installing a single RSA certificate and key 52
3.7.2 Installing multiple certificate types 53
3.7.3 Installing root certificates 54
3.7.4 Certificate conversion utility reference 55
3.8 Defining a transport API 56
3.8.1 Sample Transport APIs 56
3.8.2 Data types 57
3.8.3 SSL_TRANSPORT_API 58
3.8.4 SSL_SEND_FUNC 59
3.8.5 SSL_RECV_FUNC 60
3.8.6 SSL_GET_TIME_FUNC 61
3.8.7 SSL_IS_UDP_FUNC 62
3.8.8 SSL_IS_BLOCKING_FUNC 63
3.8.9 Implementation hints and limitations 64
3.9 Audit events 66
3.9.1 Data types 68

4 API reference 71

4.1 Preprocessor symbols 72
4.1.1 Version number 72
4.1.2 Cipher suite IDs 73
4.1.3 Session flags 79
4.1.4 Logging flags 81
4.1.5 Warning flags 83
4.2 Data types 84
4.2.1 SSL_SEND_FUNC 85
4.2.2 SSL_RECV_FUNC 86
4.2.3 SSL_GET_TIME_FUNC 87
4.2.4 SSL_IS_UDP_FUNC 88
4.2.5 SSL_IS_BLOCKING_FUNC 89
4.2.6 SSL_TRANSPORT_API 90
4.3 Information functions 91
4.3.1 SSL_GetVersionText() 92
4.3.2 SSL_GetCopyrightText() 93
4.4 Control functions 94
4.4.1 SSL_Exit() 95
4.4.2 SSL_Init() 96
4.5 Configuration functions 97
4.5.1 SSL_CIPHER_Add() 98
4.5.2 SSL_CLIENT_ConfigMutualAuth() 99
4.5.3 SSL_CURVE_Add() 100
4.5.4 SSL_CURVE_Remove() 101
4.5.5 SSL_CURVE_GetName() 102
4.5.6 SSL_MAC_Add() 103
4.5.7 SSL_MEM_Add() 104
4.5.8 SSL_MEM_ConfigSystem() 105
4.5.9 SSL_MEM_GetContext() 106
4.5.10 SSL_PROTOCOL_Add() 107
4.5.11 SSL_PROTOCOL_GetText() 108
4.5.12 SSL_ROOT_CERTIFICATE_Add() 109
4.5.13 SSL_ROOT_CERTIFICATE_AddDER() 110
4.5.14 SSL_ROOT_CERTIFICATE_LoadDER() 111
4.5.15 SSL_SERVER_ConfigMutualAuth() 112
4.5.16 SSL_SetDefaultCertificateAPI() 113
4.5.17 SSL_SIGNATURE_ALGORITHM_Add() 114
4.5.18 SSL_SIGNATURE_SIGN_Add() 115

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.19 SSL_SIGNATURE_VERIFY_Add() 116
4.5.20 SSL_SUITE_Add() 117
4.6 Session control functions 120
4.6.1 SSL_SESSION_Accept() 121
4.6.2 SSL_SESSION_ClrFlags() 122
4.6.3 SSL_SESSION_Connect() 123
4.6.4 SSL_SESSION_Disconnect() 124
4.6.5 SSL_SESSION_GetRecordSizeLimit() 125
4.6.6 SSL_SESSION_GetSessionID() 126
4.6.7 SSL_SESSION_GetSocket() 127
4.6.8 SSL_SESSION_GetSuite() 128
4.6.9 SSL_SESSION_Prepare() 129
4.6.10 SSL_SESSION_Process() 130
4.6.11 SSL_SESSION_Receive() 131
4.6.12 SSL_SESSION_QueryFlags() 132
4.6.13 SSL_SESSION_Send() 133
4.6.14 SSL_SESSION_SendStr() 134
4.6.15 SSL_SESSION_SetAllowedSuites() 135
4.6.16 SSL_SESSION_SetCertificateAPI() 136
4.6.17 SSL_SESSION_SetFlags() 137
4.6.18 SSL_SESSION_SetProtocolRange() 139
4.6.19 SSL_SESSION_SetPSK() 140
4.6.20 SSL_SESSION_SetRecordSizeLimit() 141
4.7 Cipher suite functions 142
4.7.1 SSL_SUITE_CopyName() 143
4.7.2 SSL_SUITE_FindByID() 144
4.7.3 SSL_SUITE_FindByIndex() 145
4.7.4 SSL_SUITE_GetCipherName() 146
4.7.5 SSL_SUITE_GetIanaName() 147
4.7.6 SSL_SUITE_GetID() 148
4.7.7 SSL_SUITE_GetKeyExchangeName() 149
4.7.8 SSL_SUITE_GetMACAlgorithmName() 150
4.7.9 SSL_SUITE_GetPKAlgorithmName() 151
4.7.10 SSL_SUITE_QueryNull() 152
4.7.11 SSL_SUITE_QueryPKAlgorithm() 153
4.7.12 SSL_SUITE_QueryRequiresECC() 154
4.7.13 SSL_SUITE_QueryRequiresPSK() 155
4.7.14 SSL_SUITE_QueryValidity() 156
4.8 Diagnostic functions 157
4.8.1 SSL_AddLogFilter() 158
4.8.2 SSL_AddWarnFilter() 159
4.8.3 SSL_ERROR_GetText() 160
4.8.4 SSL_RemoveLogFilter() 161
4.8.5 SSL_RemoveWarnFilter() 162
4.8.6 SSL_SetLogFilter() 163
4.8.7 SSL_SetWarnFilter() 164
4.9 Internal functions, variables, and data structures 165
4.10 API evolution 166

5 Configuring emSSL 168

5.1 Compile-time configuration 169
5.1.1 SSL_DEBUG 169
5.1.2 SSL_LOG_BUFFER_SIZE 169
5.1.3 SSL_MAX_SESSION_TICKET_LEN 169
5.1.4 SSL_TRANSPORT_API_GET_TIME_FUNC 169
5.1.5 SSL_MAX_APP_DATA_FRAGMENT_LEN 170
5.1.6 SSL_DTLS_CONTENT_SIZE 170
5.1.7 SSL_SESSION_CACHE_SIZE 170
5.1.8 SSL_AUDIT_ENABLED 170

 © 2015-2026 SEGGER Microcontroller GmbH

5.2 Runtime configuration 171
5.2.1 Cipher suites explained 171
5.2.2 Adding cipher suites 171
5.2.3 Adding ciphers 174
5.2.4 Adding MACs 175
5.2.5 Adding public key signature verifiers 175
5.2.6 Adding public key message signers 175
5.2.7 Adding signature algorithms 176
5.2.8 Adding elliptic curves 177
5.2.9 Adding TLS protocols 177
5.3 SSL-OS integration 179
5.3.1 SSL-OS API 179
5.3.2 SSL-OS binding for embOS 184
5.3.3 SSL-OS binding for bare metal 187

6 Configuring emCrypt 190

6.1 Overview 191
6.2 Runtime configuration 192
6.2.1 CRYPTO_X_Config() 192
6.2.2 CRYPTO_X_Panic() 194
6.2.3 Example runtime configurations 194
6.3 Compile-time configuration 197
6.3.1 Optimization goals 197
6.3.2 Lookup table storage location 198
6.3.3 Compile-time configuration reference 198
6.4 CRYPTO-OS integration 206
6.4.1 Sample bindings 206
6.4.2 CRYPTO-OS API 206
6.5 emCrypt reference 211
6.5.1 Runtime API 211
6.5.2 Compile-time API 220
6.5.3 Hardware acceleration 230

7 Resource usage 266

7.1 Memory footprint 267
7.1.1 Target system configuration 267
7.1.2 ROM use 267
7.1.3 RAM use 268
7.2 Minimal system configuration 270

8 Best practice 271

8.1 SSL/TLS protocol versions 272
8.2 DTLS protocol versions 273
8.3 Fallback to lower versions 274
8.4 Compression 275
8.5 Server name indication 276
8.6 General guidelines 277
8.7 Recommended cipher suites 278
8.8 Implementation details 279
8.9 Truncated HMAC 280

9 Reference information 281

9.1 Cipher suites 282
9.2 TLS specifications 283
9.3 Extensions 284
9.4 Cryptography 285
9.5 Hardware acceleration 286
9.5.1 RSA and elliptic curve accelerators 286

 © 2015-2026 SEGGER Microcontroller GmbH

9.5.2 Hash and MAC accelerators 286
9.5.3 Bulk encipherment accelerators 286
9.5.4 Vendor-optimized and certified libraries 286
9.5.5 What to do if you require alternative cryptography 286

10 Patents and export controls 287

10.1 Using RSA cipher suites 288
10.2 Using DSA cipher suites 289
10.3 Using ECC cipher suites 290
10.4 Export controls 291

11 Glossary 292

12 Indexes 295

12.1 Function index 296
12.2 Subject index 298

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 1

Introduction to emSSL

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

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

1.4 Package content

emSSL is provided as 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):

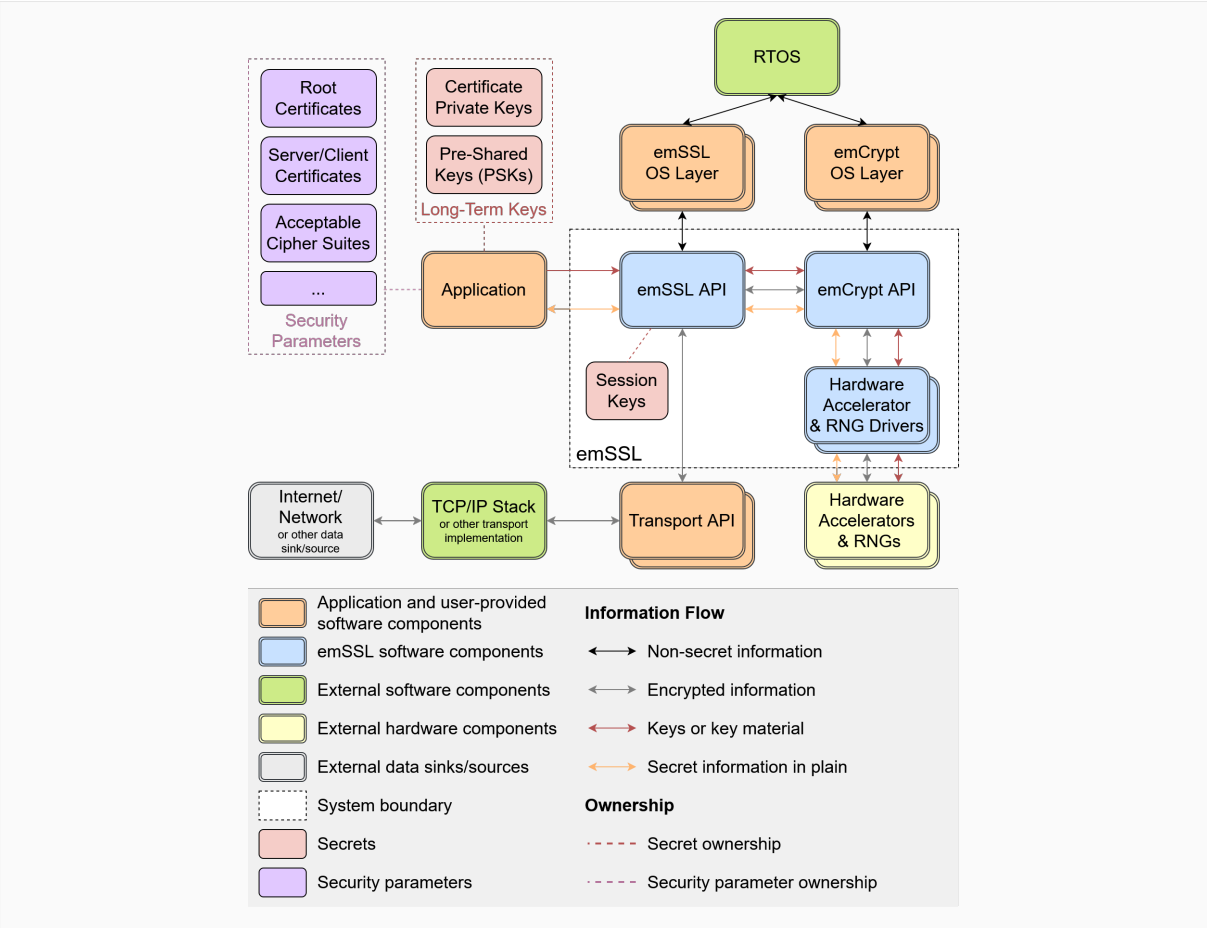
- `Config`
- `CRYPTO`
- `SEGGER`
- `SSL`

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

> **Warning**
>
> 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.

 © 2015-2026 SEGGER Microcontroller GmbH

1.5 System architecture



System architecture

1.5.1 Overview

The figure above shows the system architecture of emSSL.

For the purpose of this discussion, emSSL comprises three components. It consists of its own emSSL API component and two components borrowed from SEGGER emCrypt: the emCrypt API and the drivers for hardware accelerators and random number generators (RNGs). The following paragraphs describe each component of emSSL as well as relevant external components that emSSL interacts with.

Stacked boxes (a second outline offset behind the first) denote components for which several interchangeable implementations exist.

emSSL API

The core of emSSL is its API component, which implements the TLS protocol logic and provides a high-level, easy-to-use interface for the application. It is tightly coupled with the emCrypt API.

emCrypt API

The emCrypt API provides the basic cryptographic operations required to run the TLS protocol, such as symmetric ciphers, asymmetric cryptographic algorithms, key exchange algorithms, hashes, and MACs.

Hardware Accelerator & RNG Drivers

The drivers connect the emCrypt API with external hardware RNGs and with external hardware accelerators for symmetric ciphers and hashes.

 © 2015-2026 SEGGER Microcontroller GmbH

RTOS, emSSL/emCrypt OS Layer

Both emSSL and emCrypt require some basic services from the underlying real-time operating system (RTOS), such as locking mechanisms and a way to retrieve a system tick count. Because emSSL itself is designed to be RTOS-agnostic, the user must implement a couple of callback functions that provide those services (see *SSL-OS integration* on page 179 and *CRYPTO-OS integration* on page 206 for more details and samples). These callback functions constitute the emSSL and emCrypt OS layers, which are located outside the emSSL system and interface with the emSSL and emCrypt API components, respectively.

Transport API

emSSL is also agnostic to the underlying transport that carries the encrypted data streams. This transport is typically a TCP/IP stack that communicates over the network or internet. Similar to the OS layers, a handful of callback functions need to be implemented by the user (see *Defining a transport API* on page 56 for more details and samples). These callback functions constitute the Transport API, which is located outside the emSSL system and interfaces with the emSSL API.

Application

The application contains the business logic of the embedded device. It is developed by the user and makes use of the emSSL API for secure communication.

1.5.2 Information flows

The application initiates the information flow. The first step to establish a confidential and authenticated communication channel with a remote host is to set up emSSL by providing the required key material, certificates, and other security parameters to the emSSL API.

Then, the application instructs the emSSL API to perform a TLS handshake with the peer. The emSSL API generates, sends, receives, and parses the required handshake messages. In this process, emSSL accesses the transport API to dispatch and retrieve messages. It also accesses the emCrypt API to perform the necessary cryptographic operations.

If available, the emCrypt API can access hardware accelerators through its drivers. When a hardware accelerator is used, the key material and data required to perform a cryptographic operation are transferred to that accelerator. Once the cryptographic operation is complete, its result is transferred back from the accelerator to the emCrypt API.

Similarly, the emCrypt API can query supported hardware random number generators through the respective drivers. Random numbers are either used by the emCrypt API internally or delivered to the emSSL API for use during the TLS handshake.

Once the handshake is completed, the application is able to use the established channel to send and receive data. In particular, the application can pass plaintext data to the emSSL API, which encrypts and authenticates the data using the emCrypt API and passes it to the transport API for transmission. Furthermore, the application can ask the emSSL API to receive data from the peer. The emSSL API then retrieves the encrypted data from the transport API, decrypts and verifies the data using the emCrypt API, and provides the plaintext data to the application for further processing.

1.5.3 Secrets

Long-term secrets, such as private keys for certificates or pre-shared keys, are stored, managed, and delivered to the emSSL API by the application.

Any short-term, session-related secrets, such as negotiated session keys, are managed by emSSL internally. They only leave the emSSL system when they need to be passed to a hardware accelerator.

1.5.4 Security parameters

The TLS protocol requires certain security parameters to be defined. They include:

 © 2015-2026 SEGGER Microcontroller GmbH

- A list of trusted root certificates, see *Installing root certificates* on page 54.
- If required, a server or client certificate (or certificate chain), see *Installing certificates and keys* on page 52.
- A list of acceptable cipher suites, ordered by preference, and their respective primitives, see *Runtime configuration* on page 192.
- The minimum and maximum TLS protocol versions to be supported, see *Adding TLS protocols* on page 177.

Those security parameters are set by the application through the emSSL API.

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

2.1 Using a PC to try emSSL

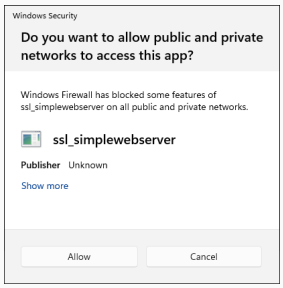
emSSL is shipped with a precompiled example that demonstrates a simple SSL web server. You can run the example on Windows or Linux and connect to the local web server on port 8443.

```
C:> SSL_SimpleWebServer.exe

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
emSSL Simple Secure Web Server V3.18.0 compiled Aug 27 2026 14:24:21

Waiting for connection on port 8443...
_
```

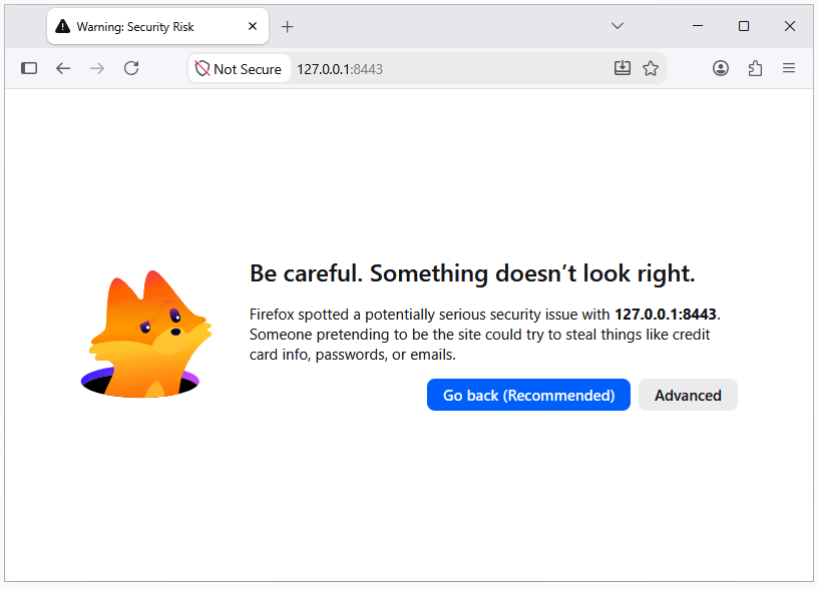
When you run this on Windows, Windows Firewall may present a dialog asking whether to grant network access to the application:



Windows Firewall dialog

Grant access by clicking *Allow*, otherwise you will not be able to serve web pages to clients.

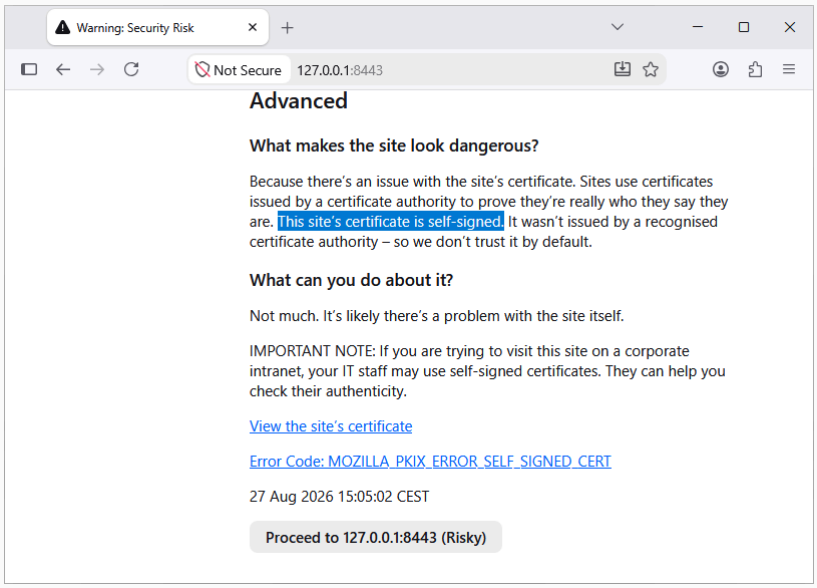
The web server application is waiting for a client to connect to it such that it can serve its small web page. Now start a web browser and open the URL "`https://127.0.0.1:8443/`". This example uses Mozilla Firefox:



Browser warning in Firefox

The warning shown by the browser indicates that the certificate presented is invalid. After clicking on *Advanced* we are presented with additional information:

 © 2015-2026 SEGGER Microcontroller GmbH



Browser warning in Firefox after clicking "Advanced"

According to the browser, the certificate is rejected because it is self-signed rather than a fully-authenticated certificate for a website on the Internet. This is expected for this sample application. In production use, it is of course possible to configure emSSL with a custom certificate that has been signed by a well-known certification authority, avoiding such warning messages for the end user.

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:

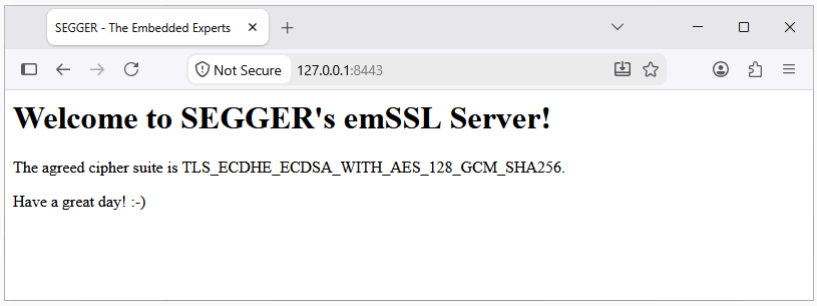
```
C:> SSL_SimpleWebServer.exe

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
emSSL Simple Secure Web Server V3.18.0 compiled Aug 27 2026 14:24:21

Waiting for connection on port 8443...
Connection made, attempting to upgrade to secure...
Can't negotiate a secure connection. Session closed.

Waiting for connection on port 8443...
_
```

To proceed to the sample web page, click *Proceed to 127.0.0.1:8443 (Risky)*. If you are using a different browser, the warning message may look slightly different, but should also allow you to proceed to the web page. You should then be greeted with a short web page served by emSSL on your PC:



Page served by emSSL

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`.

 © 2015-2026 SEGGER Microcontroller GmbH

To present this page, the browser made two additional connections to the server: one to gather the web page itself, and a second connection to load any favicon. Refreshing 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

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
emSSL Simple Secure Web Server V3.18.0 compiled Aug 27 2026 14:24:21

Waiting for connection on port 8443...
Connection made, attempting to upgrade to secure...
Can't negotiate a secure connection. Session closed.

Waiting for connection on port 8443...
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 8443...
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 8443...
^C
C:> _
```

 © 2015-2026 SEGGER Microcontroller GmbH

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.exe www.segger.com

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
emSSL TLS Scan V3.18.0 compiled Aug 27 2026 14:21:58

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

C02F ECDHE_RSA_WITH_AES_128_GCM_SHA256         TLS 1.2  RSA     147 ms
C030 ECDHE_RSA_WITH_AES_256_GCM_SHA384         TLS 1.2  RSA     155 ms
CCA8 ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256   TLS 1.2  RSA     149 ms

3 common cipher suites out of 70 tested

C:> _
```

This shows that emSSL has made 3 successful connections to `www.segger.com` out of 70 different cipher suites 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:> SSL_Scan.exe www.facebook.com

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
emSSL TLS Scan V3.18.0 compiled Aug 27 2026 14:21:58

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

C008 ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA         TLS 1.2  ECDSA   242 ms
C009 ECDHE_ECDSA_WITH_AES_128_CBC_SHA          TLS 1.2  ECDSA   190 ms
C02B ECDHE_ECDSA_WITH_AES_128_GCM_SHA256       TLS 1.2  ECDSA   182 ms
C00A ECDHE_ECDSA_WITH_AES_256_CBC_SHA          TLS 1.2  ECDSA   143 ms
C02C ECDHE_ECDSA_WITH_AES_256_GCM_SHA384       TLS 1.2  ECDSA   177 ms
CCA9 ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 TLS 1.2  ECDSA   160 ms
C012 ECDHE_RSA_WITH_3DES_EDE_CBC_SHA           TLS 1.2  RSA     146 ms
C013 ECDHE_RSA_WITH_AES_128_CBC_SHA            TLS 1.2  RSA     137 ms
C02F ECDHE_RSA_WITH_AES_128_GCM_SHA256         TLS 1.2  RSA     185 ms
C014 ECDHE_RSA_WITH_AES_256_CBC_SHA            TLS 1.2  RSA     153 ms
C030 ECDHE_RSA_WITH_AES_256_GCM_SHA384         TLS 1.2  RSA     145 ms
CCA8 ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256   TLS 1.2  RSA     139 ms

12 common cipher suites out of 70 tested

C:> _
```

Some 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.

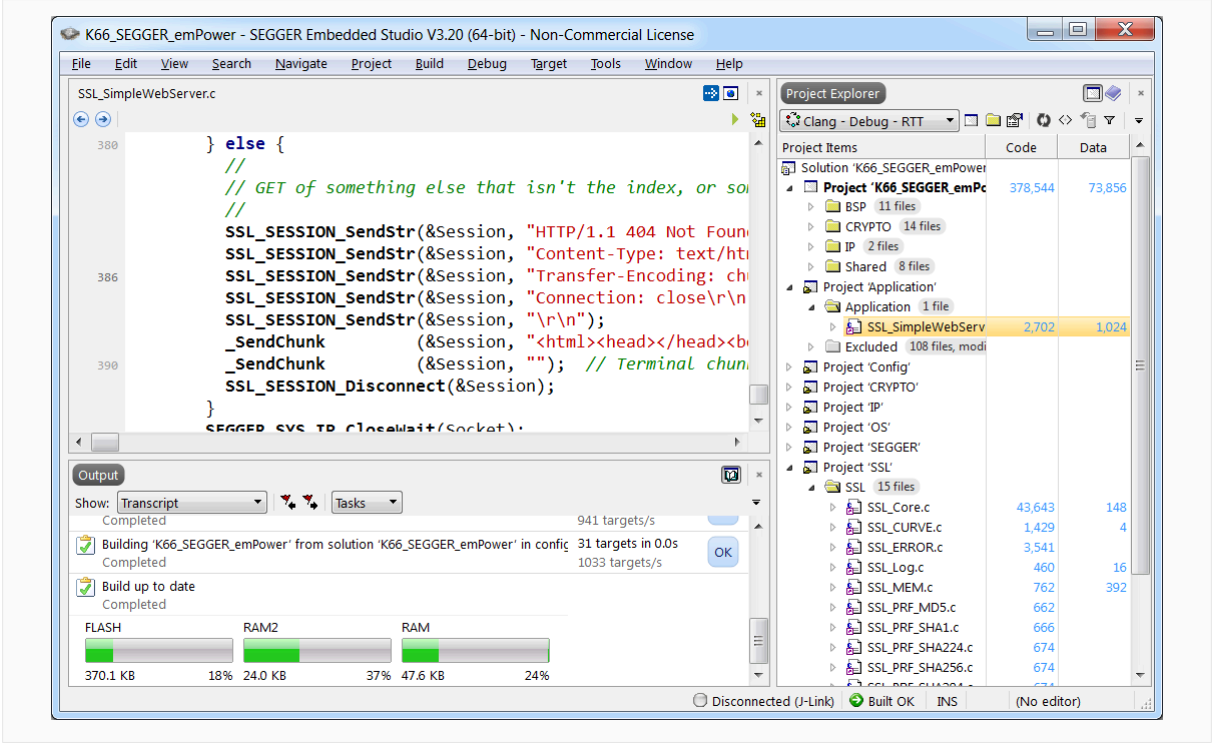
 © 2015-2026 SEGGER Microcontroller GmbH

2.3 Moving to embedded hardware

When starting to run emSSL on embedded hardware, we recommend that you use one of the "Start" projects supplied 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 a SEGGER emPower board which is supplied with Embedded Studio PRO or available separately from SEGGER and through authorized distributors.

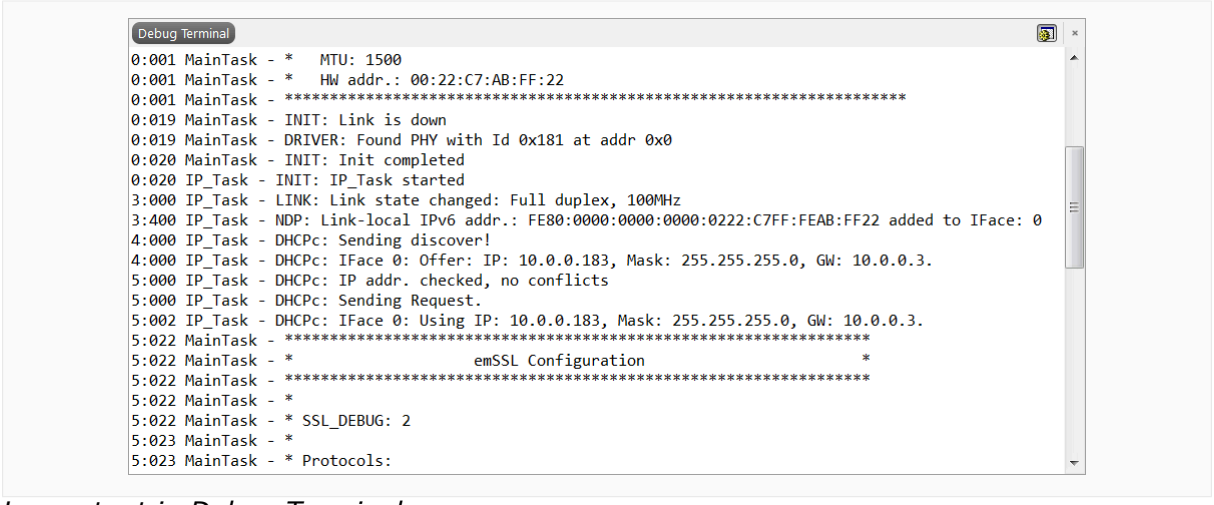
Start Embedded Studio and load the SEGGER emPower start project:



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.

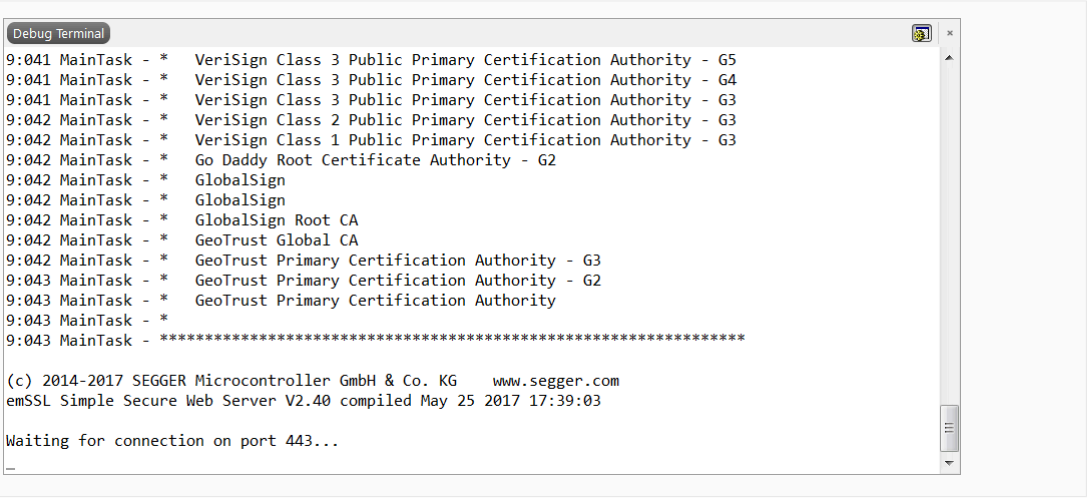
The **Debug Terminal** will show the configured IP address of the emPower board, and you will be able to use a browser to show web pages served from the embedded target in the same way as the PC application above. If you do not see the **Debug Terminal**, choose **View > Debug Terminal**. The terminal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



Log output in Debug Terminal

 © 2015-2026 SEGGER Microcontroller GmbH

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



emSSL initialized and waiting for a connection

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

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

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 is 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.

 © 2015-2026 SEGGER Microcontroller GmbH

3.2 What to expect

The following sections describe a client-server pair of applications that provide 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!
```

 © 2015-2026 SEGGER Microcontroller GmbH

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

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

 © 2015-2026 SEGGER Microcontroller GmbH

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 38.

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 for the application.

How you open a socket to the remote server depends on the underlying networking API. To make the 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 emSSL's capabilities is handled by `SSL_X_Config()`, which is called as part of the SSL initialization process performed 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 168.

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 is responsible for accepting connections:

 © 2015-2026 SEGGER Microcontroller GmbH

```
//
// 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, &SSLTransportAPI_TCP_Blocking);  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 on the stack.

Most notably, `SSL_SESSION_Prepare()` is provided a Transport API, a set of function pointers in a structure that point to appropriate send and receive functions for a socket. In this example, we use the sample functions for the SEGGER abstraction layer that ship with emSSL (see *Sample Transport APIs* on page 56) to provide socket services.

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 the client and server can agree on a common set of communication

 © 2015-2026 SEGGER Microcontroller GmbH

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 is 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 written 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 if an error occurred on the connection.

Note that the character count is stored in `Status` for later use, as well as any error status.

❸ Apply ROT13 transform

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

 © 2015-2026 SEGGER Microcontroller GmbH

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 handled separately and indicates that the connection has been closed by the client end. This 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 with 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.

 © 2015-2026 SEGGER Microcontroller GmbH

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';
    }
  }
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

3.3.6 SSL_ROT13Server.c complete listing

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

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

Purpose     : Simple server that provides a secure ROT13 service.

*/

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

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

#ifndef   ROT13_ADDR
  #define ROT13_ADDR          "0.0.0.0"       // IP_ANY
#endif
#ifndef   ROT13_PORT
  #define ROT13_PORT          "19000"
#endif

/*********************************************************************
*
*             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, char *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;
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
      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(char *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
*
**********************************************************************
*/
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
/*********************************************************************
*
*       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_BindEx(ROT13_ADDR, ROT13_PORT, SEGGER_SYS_IP_PROTOCOL_TCP);
  if (BoundSocket < 0) {
    SEGGER_SYS_IO_Printf("Cannot bind server socket\n");
    SEGGER_SYS_OS_Halt(100);
  }
  //
  for (;;) {
    //
    do {
      Socket = SEGGER_SYS_IP_AcceptEx(BoundSocket);
    } while (Socket < 0);
    //
    SSL_SESSION_Prepare(&Session, Socket, &SSLTransportAPI_TCP_Blocking);
    Status = SSL_SESSION_Accept(&Session);
    //
    if (Status < 0) {
      SEGGER_SYS_IO_Printf("Cannot 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_Close(Socket);
    }
  }
}

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

 © 2015-2026 SEGGER Microcontroller GmbH

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 44.

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 for the application and are identical to the server application.

❷ Initialize SSL component

The call to `SSL_Init()` initializes 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, &SSLTransportAPI_TCP_Blocking);  ❷
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
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 well, the function returns a socket handle. If it fails 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 the 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()`. An 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()` sends a request to the ROT13 server and accepts its response:

 © 2015-2026 SEGGER Microcontroller GmbH

```
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 remaining code is for processing errors and echoing data sent and received, and is not further explained.

 © 2015-2026 SEGGER Microcontroller GmbH

3.4.4 SSL_ROT13Client.c complete listing

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

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

Purpose     : Simple client that uses a secure ROT13 service.

*/

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

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

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

#ifndef   ROT13_SERVER
  #define ROT13_SERVER  "127.0.0.1"
#endif
#ifndef   ROT13_PORT
  #define ROT13_PORT    "19000"
#endif

/*********************************************************************
*
*       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;
    }
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
    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, 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_ConnectEx(ROT13_SERVER,
                                   ROT13_PORT,
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
                                    SEGGER_SYS_IP_PROTOCOL_TCP);
  if (Socket < 0) {
    SEGGER_SYS_IO_Printf("Cannot open %s:%s\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, &SSLTransportAPI_TCP_Blocking);
  if (SSL_SESSION_Connect(&Session, ROT13_SERVER) < 0) {
    SEGGER_SYS_IO_Printf("Cannot negotiate a secure connection to %s:%s\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 ***************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

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 certificate 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 an 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.

 © 2015-2026 SEGGER Microcontroller GmbH

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 3072 bits:

```
MacBook:~ paul$ openssl genrsa -out rsakey.pem 3072
Generating RSA private key, 3072 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 3072 bits.

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$ _
```

 © 2015-2026 SEGGER Microcontroller GmbH

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 52.

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 52.

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 in a two-stage process:

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

First, use OpenSSL to generate a set of DSA parameters using a strong 3072-bit prime:

```
MacBook:~ paul$ openssl dsaparam -out dsaparam.pem 3072
Generating DSA parameters, 3072 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 share 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 these 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 resulting file contains the private key used for signing:

```
MacBook:~ paul$ openssl gendsa -out dsakey.pem dsaparam.pem
Generating DSA key, 3072 bits
MacBook:~ paul$ _
```

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

 © 2015-2026 SEGGER Microcontroller GmbH

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 -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, as covered in *Installing certificates and keys* on page 52.

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 DSA 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 52.

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 eckey.pem -name secp224r1 -genkey
MacBook:~ paul$ _
```

This generates a set of ECDSA parameters and a 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).

 © 2015-2026 SEGGER Microcontroller GmbH

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 -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 52.

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 52.

 © 2015-2026 SEGGER Microcontroller GmbH

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 utility `PrintCert` supplied with emSSL will take a PEM- or DER-encoded certificate file and convert it to a C array:

```
C:> PrintCert rsacert.der > rsacert.c
C:> PrintCert rsakey.der > rsakey.c
C:> _
```

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

```
/*********************************************************************
*
*       X.509 Certificate
*
*  Subject
*    localhost
*
*  Issuer
*    PrintCert RSA Root CA
*
*  Serial number
*    42 08 59 2D DD 81 BC 00
*
*  Validity
*    Not before: 25-Aug-26 08:51:00
*    Not after:  25-Aug-27 08:51:00
*
*  RSA public key
*    2048 bit modulus
*/
static const unsigned char Certificate_DER[1043] = {
  /*0000*/   0x30, 0x82, 0x04, 0x0F, 0x30, 0x82, 0x02, 0xF7,   // 0...0...
  ...
};
```

Similarly, the file `rsakey.c` contains a C array definition called `RSAPrivateKey_PKCS1_DER` that represents the private key.

To integrate the certificate and private key into emSSL, you provide an implementation for the certificate API. An example is given in the sample file `SSL_X_Config.c` that ships with emSSL.

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_CERTIFICATE_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,
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
                           unsigned      Index,
                           const U8   ** ppData,
                           unsigned    * pLen) {
  //
  // We only support a single self-signed certificate and
  // corresponding private key.
  //
  if (Index == 0) {
    *ppData = Certificate_DER;
    *pLen   = sizeof(Certificate_DER);
    return 0;
  } else {
    *ppData = 0;
    *pLen   = 0;
    return -1;
  }
}
```

The same applies to 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   = RSAPrivateKey_PKCS1_DER;
  *pDataLen = sizeof(RSAPrivateKey_PKCS1_DER);
  return 0;
}
```

Make sure to either copy or include the contents of `rsacert.c` and `rsakey.c` into the C file containing the `_GetCertificate` and `_GetPrivateKey` functions, or make them available via `extern` declarations. Note that you need to remove the `static` keyword from the generated array definitions for external use. This can be done manually or by adding the `-x` switch to the `PrintCert` command line.

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) {
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
    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 a DER or PEM certificate to something that can be added to emSSL. `PrintCert` is part of the emSSL shipping package.

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

> **Warning**
>
> We highly recommend 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 using `PrintCert`:

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

Copyright (c) 2015-2026 SEGGER Microcontroller GmbH    www.segger.com
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
emSSL PrintCert V3.18.0 compiled Aug 24 2026 17:09:37

MacBook:~ paul$ _
```

The `PrintCert` utility generates a C-code representation of the root certificate. Add this representation to your code base. Next, insert the certificate into the emSSL trust store by adding the following line to the `SSL_X_Config()` function:

```
SSL_ROOT_CERTIFICATE_Add(&SSL_CERTIFICATE_GeoTrust_Primary_CA);
```

Depending on how you add the C-code representation to your code base, you may additionally need to add an `extern` declaration to the `SSL_X_Config.c` file:

```
extern SSL_ROOT_CERTIFICATE SSL_CERTIFICATE_GeoTrust_Primary_CA;
```

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` [options] <file>

emSSL PrintCert accepts the following command line options:

Option	Description
`-x`	Declare objects with external linkage instead of internal linkage (`static`).
`-r`	Print a root certificate.
`-p <str>`	Prefix object names with *str*.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8 Defining a transport API

emSSL is designed to be agnostic to the underlying hardware and the underlying data transport layer. The most common transport protocols are UDP and TCP (on top of IP), but emSSL can be used with other transports as well. The library is also independent of any concrete IP stack or transport layer implementations.

For this reason, it is necessary to couple emSSL with the concrete transport that is to be used on the target system. To do so, you need to define how data can be sent and received over the transport, and how certain characteristics of a connection can be queried by emSSL.

Specifically, a structure of type `SSL_TRANSPORT_API` must be defined and provided to the library when calling `SSL_SESSION_Prepare()`. The members of this structure are callback functions that emSSL calls when necessary.

You may use the sample transport API implementations that ship with emSSL or implement your own. The required callbacks are listed below and explained in detail in the following sections.

3.8.1 Sample Transport APIs

emSSL ships with two sets of sample transport APIs.

For PC: SEGGER_SYS_IP

The first set of sample transport APIs uses the `SEGGER_SYS_IP` abstraction layer, which couples emSSL with the network stacks of Windows and Linux. These transport APIs implement all necessary callback functions and can be used without further configuration on common PC platforms.

For embedded: emNet

The second set of sample transport APIs uses the SEGGER emNet network stack for transport. These transport APIs implement all necessary callback functions except `SSL_GET_TIME_FUNC`, which provides the current (wall clock) time as a UNIX timestamp. The implementation of this callback depends on the selected microcontroller and its peripherals, therefore no sensible default can be provided. Thus, the `SSL_GET_TIME_FUNC` callback must be implemented manually and then referenced in the compile-time definition `SSL_TRANSPORT_API_GET_TIME_FUNC` in `SSL_Conf.h`. See *SSL_GET_TIME_FUNC* on page 61, *The SSL_GET_TIME_FUNC callback* on page 64, and *SSL_TRANSPORT_API_GET_TIME_FUNC* on page 169 for more details.

Transport API variants

Each set of sample transport APIs comprises three API variants:

- `SSLTransportAPI_TCP_Blocking`, which uses TLS on blocking sockets.
- `SSLTransportAPI_TCP_Nonblocking`, which uses TLS on nonblocking sockets.
- `SSLTransportAPI_UDP_Nonblocking`, which uses DTLS on nonblocking sockets.

If you don’t know which of the three API variants is right for you, try `SSLTransportAPI_TCP_Blocking` first.

Using the sample transport APIs in a project

The sample transport APIs are provided in C files in the directory `Sample/TransportAPI` in the shipping package. To use one of them, simply add the corresponding C file to your project in your specific development environment. The header file `SSL.h` contains external declarations for all three variants, allowing them to be used directly in applications that include the main SSL header.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.2 Data types

Function	Description
SSL_TRANSPORT_API	Transport-level functions for TLS protocol.
Callbacks	
SSL_SEND_FUNC	Send data to transport layer.
SSL_RECV_FUNC	Receive data from transport layer.
SSL_GET_TIME_FUNC	Get current time.
SSL_IS_UDP_FUNC	Inquire if transport is UDP (datagram).
SSL_IS_BLOCKING_FUNC	Inquire if transport is in blocking or non-blocking mode.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.3 SSL_TRANSPORT_API

Description

Transport-level functions for TLS protocol.

Type definition

```
typedef struct SSL_TRANSPORT_API_tag {
  SSL_SEND_FUNC          * pfSend;
  SSL_RECV_FUNC          * pfReceive;
  SSL_GET_TIME_FUNC      * pfGetTime;
  SSL_IS_UDP_FUNC        * pfIsUDP;
  SSL_IS_BLOCKING_FUNC   * pfIsBlocking;
} SSL_TRANSPORT_API;
```

Structure members

Member	Description
pfSend	Send data to transport layer
pfReceive	Receive data from transport layer
pfGetTime	Get current time
pfIsUDP	Inquire if transport is UDP (datagram)
pfIsBlocking	Inquire if transport is in blocking or nonblocking mode

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.4 SSL_SEND_FUNC

Description

Send data to transport layer.

Type definition

```
typedef int SSL_SEND_FUNC(      int    Socket,
                          const char * pData,
                                int    Len,
                                int    Flags);
```

Parameters

Parameter	Description
Socket	Handle for socket to send to.
pData	Pointer to data to send.
Len	Number of octets to send.
Flags	Transport-specific flags; none defined by emSSL.

Return value

≥ 0	Success, number of bytes sent.
= SSL_ERROR_AGAIN	Nonblocking socket would block.
= SSL_ERROR_NETWORK_ERROR	Other error.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

Any implementation must adhere to the following rules:

- The function must handle any protocol that you intend to use (e.g., TCP and UDP) correctly.
- The function must handle sockets with any blocking behavior that you intend to use (e.g., blocking and non-blocking) correctly.
- The function must return SSL_ERROR_AGAIN when a non-blocking socket would block. SSL_ERROR_AGAIN may never be returned for blocking sockets.
- It is acceptable to send only a prefix of the supplied data. In this case, return the number of bytes that have been sent.
- Make sure to return the correct error codes specified above. Note that those are incompatible with a typical POSIX-style send function and usually need to be translated.

If you implement this function on top of a POSIX-style send API, note that data that does not fit into a single UDP datagram may be outright rejected. In this case, send will indicate a message size error without sending the data. The maximum size of an UDP datagram is typically limited by the MTU on the IP layer.

By default, emSSL produces fragments of 1024 bytes, which results in datagrams that are smaller than the standard MTU of around 1500 bytes in most networks. Should you expect smaller UDP datagram size limits, reduce the fragment size in emSSL (by setting SSL_MAX_APP_DATA_FRAGMENT_LEN at compile time or calling SSL_SESSION_SetRecordSizeLimit at runtime). Consult the manual section *Implementation hints and limitations* on page 64 for more information.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.5 SSL_RECV_FUNC

Description

Receive data from transport layer.

Type definition

```
typedef int SSL_RECV_FUNC(int    Socket,
                          char * pData,
                          int    Len,
                          int    Flags);
```

Parameters

Parameter	Description
Socket	Handle for socket to receive from.
pData	Pointer to object which receives the data.
Len	Maximum number of octets to receive.
Flags	Transport-specific flags; none defined by emSSL.

Return value

> 0	Success, number of bytes received.
= 0	Success, 0 bytes received (only acceptable for UDP sockets).
= SSL_ERROR_EOF	Socket closed gracefully.
= SSL_ERROR_AGAIN	Socket would block (only acceptable for non-blocking sockets).
= SSL_ERROR_NETWORK_ERROR	Other error.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

Any implementation must adhere to the following rules:

- The function must handle any protocol that you intend to use (e.g., TCP and UDP) correctly.
- The function must handle sockets with any blocking behavior that you intend to use (e.g., blocking and non-blocking) correctly.
- The function must return SSL_ERROR_AGAIN when a non-blocking socket would block. SSL_ERROR_AGAIN may never be returned for blocking sockets.
- It is acceptable to return fewer bytes than requested. In this case, return the number of bytes received.
- Only return 0 to indicate a 0-length UDP packets. Never return 0 otherwise. Specifically, never return 0 when receiving from a TCP socket.
- If a UDP packet is longer than Len, it is acceptable to discard any excess bytes (i.e., this function does not need to buffer excess data). This matches the behavior of typical POSIX-style recv functions.
- Make sure to return the correct error codes specified above. Note that those are incompatible with a typical POSIX-style recv function and usually need to be translated.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.6 SSL_GET_TIME_FUNC

Description

Get current time.

Type definition

```
typedef U32 SSL_GET_TIME_FUNC(void);
```

Return value

UNIX timestamp: The number of seconds elapsed since the UNIX epoch, i.e., 00:00:00 UTC on 1970-01-01.

Additional information

This defines a callback function to be referenced in a `SSL_TRANSPORT_API` struct.

It is strongly recommended to implement this function whenever possible. TLS security is weakened if the time of day is not available. Most importantly, the library cannot detect expired certificates without it.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.7 SSL_IS_UDP_FUNC

Description

Inquire if transport is UDP (datagram).

Type definition

```
typedef int SSL_IS_UDP_FUNC(int Socket);
```

Parameters

Parameter	Description
Socket	Transport socket to query.

Return value

= 0 Transport is TCP for TLS.
≠ 0 Transport is UDP for DTLS.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

This function allows the emSSL library to test whether a socket is datagram-based (typically UDP) or stream-based (typically TCP).

This function is called regularly throughout the library and should be implemented as efficiently as possible.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.8 SSL_IS_BLOCKING_FUNC

Description

Inquire if transport is in blocking or nonblocking mode.

Type definition

```
typedef int SSL_IS_BLOCKING_FUNC(int Socket);
```

Parameters

Parameter	Description
Socket	Transport socket to query.

Return value

= 0 Transport is nonblocking.
≠ 0 Transport is blocking.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

This function allows the emSSL library to test whether a socket is in blocking or nonblocking mode.

This function is called by the library during SSL_SESSION_Prepare().

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.9 Implementation hints and limitations

If you decide to implement your own SSL_TRANSPORT_API, consider the following hints and limitations.

3.8.9.1 The SSL_SEND_FUNC and SSL_RECV_FUNC callbacks

The SSL_SEND_FUNC and SSL_RECV_FUNC callbacks will often be implemented by calling POSIX-type send and recv system calls provided by an operating system or IP stack. Note that the POSIX API is not directly compatible with the emSSL SSL_TRANSPORT_API. Thus, send and recv *cannot* be used as callback functions directly and must be wrapped in a wrapper callback function.

In particular, emSSL requires the SSL_SEND_FUNC and SSL_RECV_FUNC callbacks to return specific error codes, which are not standardized across POSIX API implementations. For example, the Windows implementation of send and recv returns the generic error code -1 on error, and provides a more specific error code through the Windows-specific WSAGetLastError() function afterward. The callback function has to extract the specific error code and translate it to the appropriate emSSL equivalent, as specified in the API documentation (see *SSL_SEND_FUNC* on page 59 and *SSL_RECV_FUNC* on page 60).

3.8.9.2 The SSL_GET_TIME_FUNC callback

The callback SSL_GET_TIME_FUNC must return the current wall clock time as a U32 UNIX timestamp. Returning a system tick counter (e.g., clock cycles or milliseconds since last boot) is *not* acceptable. This callback is used by emSSL to verify the validity of certificates based on their *Not before* and *Not after* timestamps.

Certificates that are used before or after their validity period can only be rejected when the wall clock time is available. If this callback is not provided (i.e., set to NULL in the SSL_TRANSPORT_API structure), certificate timestamps are not checked, and expired certificates may be wrongly accepted.

3.8.9.3 The SSL_IS_UDP_FUNC callback

The callback SSL_IS_UDP_FUNC decides about the protocol variant that is used. TCP-style (stream-based) connections use TLS, while UDP-style (datagram-based) connections use DTLS. If you use a different transport than TCP/UDP, return 0 to use the TLS protocol and any other value to use the DTLS protocol for the custom transport.

If this callback is not provided (i.e., set to NULL in the SSL_TRANSPORT_API structure), a TCP-style transport is assumed and the TLS protocol is used.

3.8.9.4 The SSL_IS_BLOCKING_FUNC callback

Another important property of SSL_SEND_FUNC/SSL_RECV_FUNC is the blocking behavior. These callbacks can operate with either blocking or non-blocking behavior.

With blocking behavior, SSL_SEND_FUNC/SSL_RECV_FUNC stall the current thread when they are waiting for a network operation to complete (such as waiting for more data to be received or waiting for data to be accepted for transmission). With non-blocking behavior, these functions return immediately (with error code SSL_ERROR_AGAIN) when such a condition occurs.

In most IP stacks, send and recv functions can be configured to show either blocking or non-blocking behavior, which propagates to SSL_SEND_FUNC/SSL_RECV_FUNC.

The preferred behavior depends on the application that uses emSSL. emSSL generally supports both variants, but needs to know whether blocking or non-blocking behavior is expected. The library calls the SSL_IS_BLOCKING_FUNC callback to query this information.

If this callback is not provided (i.e., set to NULL in the SSL_TRANSPORT_API structure), blocking behavior is assumed.

 © 2015-2026 SEGGER Microcontroller GmbH

3.8.9.5 The Socket parameter

Most of the callback functions introduced above take a `Socket` parameter. From the perspective of the emSSL library, this parameter is just a generic identifier for a communication endpoint without any deeper meaning. You can provide any integer that helps the transport layer identify a particular communication endpoint. In most IP stacks, this is a numeric socket handle, but it can be any other identifier for other protocols.

3.8.9.6 UDP datagram size

When using UDP/DTLS, additional considerations must be made regarding the UDP datagram size.

Application data is wrapped in packets on the DTLS record layer. When an application provides more data to the `SSL_SESSION_Send()` function than a configurable maximum length, emSSL will fragment the data across multiple record layer packets.

The DTLS specification mandates that every such packet must fit into a single datagram. UDP datagrams may be limited in size, often by the network's MTU (usually around 1500 bytes). Thus, to ensure that packets fit the datagram size, it is important to set a reasonable fragment length for the network environment that emSSL operates in.

The maximum amount of application data to be put into a single record layer packet (i.e., the length of a fragment) can be configured in two ways: First, the default maximum size can be set through the compile-time definition `SSL_MAX_APP_DATA_FRAGMENT_LEN`. Second, the maximum size can be changed by the application at runtime by calling `SSL_SESSION_SetRecordSizeLimit()`.

Note that a record layer packet will be slightly larger than the configured fragment length, as this number does not include the overhead of the record layer protocol. The precise overhead depends on the selected cipher suite. Some estimates are listed below:

- Typical overhead for an AEAD cipher: circa 37 bytes
- Typical overhead for a block cipher: circa 94 bytes
- Theoretical maximum overhead (for a block cipher with maximum padding): 333 bytes

Thus, `SSL_MAX_APP_DATA_FRAGMENT_LEN` should be set well below the network's MTU when using UDP. It defaults to 1024 bytes.

3.8.9.7 Limitations

Consider the following limitations when using emSSL on a UDP-style transport (i.e., with the DTLS protocol):

- Blocking UDP sockets (or more generally, blocking datagram-based connections) are currently not supported and will be rejected by the library upon calling `SSL_SESSION_Accept()` or `SSL_SESSION_Connect()`.
- UDP sockets are further limited by the fact that emSSL currently does not fragment record layer packets during the DTLS handshake; only the application data sent after a successful handshake is fragmented. This can cause issues when handshake messages exceed the maximum allowed size of a UDP datagram, which is especially likely when large certificate chains are used. Network stacks may limit the size of a UDP datagram to the network's MTU (often around 1500 bytes) and reject longer UDP packets. If you observe this issue, it is recommended to allow fragmentation of those packets on the IP layer as a workaround.

 © 2015-2026 SEGGER Microcontroller GmbH

3.9 Audit events

The optional audit subsystem notifies the application about certain events. It is disabled by default and can be enabled by setting the compile-time configuration macro SSL_AUDIT_ENABLED non-zero.

If the audit subsystem is enabled, the application must implement a callback function named SSL_X_Audit() with external linkage. This function is called by emSSL whenever an audit event occurs. The function takes a pointer to an SSL_AUDIT_EVENT structure as a single parameter, which describes the event.

Event contents

Every audit event includes the following information:

- The type of the event (as SSL_AUDIT_EVENT_TYPE)
- A pointer to the current SSL session

The session pointer can be used to extract additional information about the event. We note, however, that application developers should refrain from accessing any members of the SSL_SESSION structure directly. This structure is considered an internal implementation detail and is subject to change. Only the getter functions listed in the section *Session control functions* on page 120 may be used for this purpose. For instance, the application can call SSL_SESSION_GetSocket() to retrieve the socket handle of the connection that identifies the communication peer.

We further note that any pointers returned by the SSL_SESSION_Get*() functions are pointers into the session structure, which changes as the session progresses. The application must not use those pointers to alter any session data. The pointers must not be freed. Any session information that the application needs to persist must be copied to a separate memory location before returning from SSL_X_Audit().

Event types

The following event types are currently reported:

- SSL_AUDIT_EVENT_INTEGRITY_ERROR: A message integrity error was encountered on the record layer, i.e., an invalid MAC or a tag mismatch (for AEAD ciphers) occurred.
- SSL_AUDIT_EVENT_INVALID_SESSION_ID: An emSSL server received a ClientHello message that contains a non-empty, unknown session ID. The offending session ID and its octet length can be retrieved from the session pointer through SSL_SESSION_GetSessionID().

Sample callback function SSL_X_Audit()

A sample callback function is presented below. This function handles events of the types SSL_AUDIT_EVENT_INVALID_SESSION_ID and SSL_AUDIT_EVENT_INTEGRITY_ERROR and prints related information when they occur.

```
void SSL_X_Audit(const SSL_AUDIT_EVENT *pEvent) {
  SEGGER_SYS_IO_Printf("AUDIT EVENT:\n");
  switch (pEvent->Type) {
    case SSL_AUDIT_EVENT_INVALID_SESSION_ID: {
      const U8 * pSID;
      unsigned   SIDLen;
      unsigned   i;
      //
      SSL_SESSION_GetSessionID(pEvent->pSession, &pSID, &SIDLen);
      //
      SEGGER_SYS_IO_Printf("  Invalid session ID:\n");
      SEGGER_SYS_IO_Printf("    Socket:    %d\n",
                           SSL_SESSION_GetSocket(pEvent->pSession));
      SEGGER_SYS_IO_Printf("    SessionId: ");
      for (i = 0; i < SIDLen; i++) {
        SEGGER_SYS_IO_Printf("%.2x ", pSID[i]);
      }
      SEGGER_SYS_IO_Printf("\n");
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
      break;
    }
    case SSL_AUDIT_EVENT_INTEGRITY_ERROR:
      SEGGER_SYS_IO_Printf("  Integrity error:\n");
      SEGGER_SYS_IO_Printf("    Socket: %d\n",
                           SSL_SESSION_GetSocket(pEvent->pSession));
      break;
    default:
      SEGGER_SYS_IO_Printf("  Unknown audit event\n");
      break;
  }
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

3.9.1 Data types

Type	Description
SSL_AUDIT_EVENT	Audit event.
SSL_AUDIT_EVENT_TYPE	Audit event types.

 © 2015-2026 SEGGER Microcontroller GmbH

3.9.1.1 SSL_AUDIT_EVENT

Description

Audit event.

Type definition

```
typedef struct {
  SSL_AUDIT_EVENT_TYPE   Type;
  const SSL_SESSION    * pSession;
} SSL_AUDIT_EVENT;
```

Structure members

Member	Description
Type	Type of the audit event.
pSession	Pointer to SSL session to extract additional information from (via SSL_SESSION_Get*() functions).

 © 2015-2026 SEGGER Microcontroller GmbH

3.9.1.2 SSL_AUDIT_EVENT_TYPE

Description

Audit event types.

Type definition

```
typedef enum {
  SSL_AUDIT_EVENT_INVALID_SESSION_ID,
  SSL_AUDIT_EVENT_INTEGRITY_ERROR
} SSL_AUDIT_EVENT_TYPE;
```

Enumeration constants

Constant	Description
SSL_AUDIT_EVENT_INVALID_SESSION_ID	Received ClientHello with invalid session ID.
SSL_AUDIT_EVENT_INTEGRITY_ERROR	Integrity error on the record layer.

 © 2015-2026 SEGGER Microcontroller GmbH

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.

 © 2015-2026 SEGGER Microcontroller GmbH

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    32000
```

 © 2015-2026 SEGGER Microcontroller GmbH

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
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
#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
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
#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_AES_128_GCM_SHA256                         0x1301
#define SSL_SUITE_ID_AES_256_GCM_SHA384                         0x1302
#define SSL_SUITE_ID_CHACHA20_POLY1305_SHA256                   0x1303
#define SSL_SUITE_ID_AES_128_CCM_SHA256                         0x1304
#define SSL_SUITE_ID_AES_128_CCM_8_SHA256                       0x1305
#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
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
#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
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
#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
#define SSL_SUITE_ID_PSK_WITH_CHACHA20_POLY1305_SHA256            0xCCAB
#define SSL_SUITE_ID_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256      0xCCAC
#define SSL_SUITE_ID_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256        0xCCAD
#define SSL_SUITE_ID_RSA_PSK_WITH_CHACHA20_POLY1305_SHA256        0xCCAE
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_128_GCM_SHA256          0xD001
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_256_GCM_SHA384          0xD002
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_128_CCM_8_SHA256        0xD003
#define SSL_SUITE_ID_ECDHE_PSK_WITH_AES_128_CCM_SHA256          0xD005
```

 © 2015-2026 SEGGER Microcontroller GmbH

4.1.3 Session flags

Description

Flags set by the user or by emSSL.

Definition

```
#define SSL_SESSION_FLAG_REQUEST_CERTIFICATE                  0x0001u
#define SSL_SESSION_FLAG_CERTIFICATE_RECEIVED                 0x0002u
#define SSL_SESSION_FLAG_CERTIFICATE_REQUEST_RECEIVED         0x0004u
#define SSL_SESSION_FLAG_CERTIFICATE_SENT                     0x0008u
#define SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_ID            0x0010u
#define SSL_SESSION_FLAG_REQUEST_RESUME_SESSION_TICKET        0x0020u
#define SSL_SESSION_FLAG_REQUEST_RESUME_CRITICAL              0x0040u
#define SSL_SESSION_FLAG_RESUME_GRANTED                       0x0080u
#define SSL_SESSION_FLAG_DISABLE_SESSION_TICKET               0x0100u
#define SSL_SESSION_FLAG_SESSION_TICKET_EXPECTED              0x0200u
#define SSL_SESSION_FLAG_SESSION_TICKET_EXTENSION_ADVERTISED  0x0400u
#define SSL_SESSION_FLAG_SESSION_TICKET_RECEIVED              0x0800u
#define SSL_SESSION_FLAG_DISABLE_RSA_PMS_VERSION_CHECK        0x1000u
#define SSL_SESSION_FLAG_REQUEST_PREVENT_FALLBACK             0x2000u
#define SSL_SESSION_FLAG_PREFER_SERVER_ORDER                  0x4000u
#define SSL_SESSION_FLAG_REQUIRE_STRICT_TLS_CLOSE             0x8000u
#define SSL_SESSION_FLAG_NONBLOCKING                          0x10000u
```

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.

 © 2015-2026 SEGGER Microcontroller GmbH

Definition	Description
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].
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.
SSL_SESSION_FLAG_NONBLOCKING	Set by emSSL: Underlying socket is in nonblocking mode.

 © 2015-2026 SEGGER Microcontroller GmbH

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_GENERAL          (1uL << 13)
#define SSL_LOG_CONFIG           (1uL << 14)
#define SSL_LOG_ALERT            (1uL << 15)
#define SSL_LOG_API              (1uL << 16)
#define SSL_LOG_CRYPTO           (1uL << 17)
#define SSL_LOG_DERIVE           (1uL << 18)
#define SSL_LOG_HANDSHAKE_MSG    (1uL << 19)
#define SSL_LOG_APP              (1uL << 31)
#define SSL_LOG_MESSAGES         0
```

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 hash when computing verification data.
SSL_LOG_GENERAL	Log general information
SSL_LOG_CONFIG	Log emSSL configuration on startup.
SSL_LOG_ALERT	Log received alert messages.
SSL_LOG_API	Log API calls and returns
SSL_LOG_CRYPTO	Log crypto-related information

 © 2015-2026 SEGGER Microcontroller GmbH

Definition	Description
SSL_LOG_DERIVE	Log low-level secret derivation information
SSL_LOG_HANDSHAKE_MSG	Log handshake messages
SSL_LOG_APP	Log application messages.
SSL_LOG_MESSAGES	No longer used, retained for compatibility

Additional information

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

 © 2015-2026 SEGGER Microcontroller GmbH

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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2 Data types

The table below lists the types provided by the emSSL API. A detailed description of each type is found in the sections that follow.

Function	Description
SSL_SEND_FUNC	Send data to transport layer.
SSL_RECV_FUNC	Receive data from transport layer.
SSL_GET_TIME_FUNC	Get current time.
SSL_IS_UDP_FUNC	Inquire if transport is UDP (datagram).
SSL_TRANSPORT_API	Transport-level functions for TLS protocol.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.1 SSL_SEND_FUNC

Description

Send data to transport layer.

Type definition

```
typedef int SSL_SEND_FUNC(      int    Socket,
                          const char * pData,
                                int    Len,
                                int    Flags);
```

Parameters

Parameter	Description
Socket	Handle for socket to send to.
pData	Pointer to data to send.
Len	Number of octets to send.
Flags	Transport-specific flags; none defined by emSSL.

Return value

≥ 0	Success, number of bytes sent.
= SSL_ERROR_AGAIN	Nonblocking socket would block.
= SSL_ERROR_NETWORK_ERROR	Other error.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

Any implementation must adhere to the following rules:

- The function must handle any protocol that you intend to use (e.g., TCP and UDP) correctly.
- The function must handle sockets with any blocking behavior that you intend to use (e.g., blocking and non-blocking) correctly.
- The function must return SSL_ERROR_AGAIN when a non-blocking socket would block. SSL_ERROR_AGAIN may never be returned for blocking sockets.
- It is acceptable to send only a prefix of the supplied data. In this case, return the number of bytes that have been sent.
- Make sure to return the correct error codes specified above. Note that those are incompatible with a typical POSIX-style send function and usually need to be translated.

If you implement this function on top of a POSIX-style send API, note that data that does not fit into a single UDP datagram may be outright rejected. In this case, send will indicate a message size error without sending the data. The maximum size of an UDP datagram is typically limited by the MTU on the IP layer.

By default, emSSL produces fragments of 1024 bytes, which results in datagrams that are smaller than the standard MTU of around 1500 bytes in most networks. Should you expect smaller UDP datagram size limits, reduce the fragment size in emSSL (by setting SSL_MAX_APP_DATA_FRAGMENT_LEN at compile time or calling SSL_SESSION_SetRecordSizeLimit at runtime). Consult the manual section *Implementation hints and limitations* on page 64 for more information.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.2 SSL_RECV_FUNC

Description

Receive data from transport layer.

Type definition

```
typedef int SSL_RECV_FUNC(int    Socket,
                          char * pData,
                          int    Len,
                          int    Flags);
```

Parameters

Parameter	Description
Socket	Handle for socket to receive from.
pData	Pointer to object which receives the data.
Len	Maximum number of octets to receive.
Flags	Transport-specific flags; none defined by emSSL.

Return value

> 0	Success, number of bytes received.
= 0	Success, 0 bytes received (only acceptable for UDP sockets).
= SSL_ERROR_EOF	Socket closed gracefully.
= SSL_ERROR_AGAIN	Socket would block (only acceptable for non-blocking sockets).
= SSL_ERROR_NETWORK_ERROR	Other error.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

Any implementation must adhere to the following rules:

- The function must handle any protocol that you intend to use (e.g., TCP and UDP) correctly.
- The function must handle sockets with any blocking behavior that you intend to use (e.g., blocking and non-blocking) correctly.
- The function must return SSL_ERROR_AGAIN when a non-blocking socket would block. SSL_ERROR_AGAIN may never be returned for blocking sockets.
- It is acceptable to return fewer bytes than requested. In this case, return the number of bytes received.
- Only return 0 to indicate a 0-length UDP packets. Never return 0 otherwise. Specifically, never return 0 when receiving from a TCP socket.
- If a UDP packet is longer than Len, it is acceptable to discard any excess bytes (i.e., this function does not need to buffer excess data). This matches the behavior of typical POSIX-style recv functions.
- Make sure to return the correct error codes specified above. Note that those are incompatible with a typical POSIX-style recv function and usually need to be translated.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.3 SSL_GET_TIME_FUNC

Description

Get current time.

Type definition

```
typedef U32 SSL_GET_TIME_FUNC(void);
```

Return value

UNIX timestamp: The number of seconds elapsed since the UNIX epoch, i.e., 00:00:00 UTC on 1970-01-01.

Additional information

This defines a callback function to be referenced in a `SSL_TRANSPORT_API` struct.

It is strongly recommended to implement this function whenever possible. TLS security is weakened if the time of day is not available. Most importantly, the library cannot detect expired certificates without it.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.4 SSL_IS_UDP_FUNC

Description

Inquire if transport is UDP (datagram).

Type definition

```
typedef int SSL_IS_UDP_FUNC(int Socket);
```

Parameters

Parameter	Description
Socket	Transport socket to query.

Return value

= 0 Transport is TCP for TLS.
≠ 0 Transport is UDP for DTLS.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

This function allows the emSSL library to test whether a socket is datagram-based (typically UDP) or stream-based (typically TCP).

This function is called regularly throughout the library and should be implemented as efficiently as possible.

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.5 SSL_IS_BLOCKING_FUNC

Description

Inquire if transport is in blocking or nonblocking mode.

Type definition

```
typedef int SSL_IS_BLOCKING_FUNC(int Socket);
```

Parameters

Parameter	Description
Socket	Transport socket to query.

Return value

= 0 Transport is nonblocking.
≠ 0 Transport is blocking.

Additional information

This defines a callback function to be referenced in a SSL_TRANSPORT_API struct.

This function allows the emSSL library to test whether a socket is in blocking or nonblocking mode.

This function is called by the library during SSL_SESSION_Prepare().

 © 2015-2026 SEGGER Microcontroller GmbH

4.2.6 SSL_TRANSPORT_API

Description

Transport-level functions for TLS protocol.

Type definition

```
typedef struct SSL_TRANSPORT_API_tag {
  SSL_SEND_FUNC          * pfSend;
  SSL_RECV_FUNC          * pfReceive;
  SSL_GET_TIME_FUNC      * pfGetTime;
  SSL_IS_UDP_FUNC        * pfIsUDP;
  SSL_IS_BLOCKING_FUNC   * pfIsBlocking;
} SSL_TRANSPORT_API;
```

Structure members

Member	Description
pfSend	Send data to transport layer
pfReceive	Receive data from transport layer
pfGetTime	Get current time
pfIsUDP	Inquire if transport is UDP (datagram)
pfIsBlocking	Inquire if transport is in blocking or nonblocking mode

 © 2015-2026 SEGGER Microcontroller GmbH

4.3 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.3.1 SSL_GetVersionText()

Description

Get emSSL version as printable string.

Prototype

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

Return value

Zero-terminated version string.

 © 2015-2026 SEGGER Microcontroller GmbH

4.3.2 SSL_GetCopyrightText()

Description

Get emSSL copyright as printable string.

Prototype

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

Return value

Zero-terminated copyright string.

 © 2015-2026 SEGGER Microcontroller GmbH

4.4 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.4.1 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.4.2 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5 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_ROOT_CERTIFICATE_AddDER()	Add root certificate from DER-encoded certificate.
SSL_ROOT_CERTIFICATE_LoadDER()	Load root certificate from DER-encoded certificate.
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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.1 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 174.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.2 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.3 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 cryptographic algorithm library offers more curves, the SSL protocol only supports a subset of all standardized elliptic curves.

See also

Adding elliptic curves on page 177.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.4 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.5 SSL_CURVE_GetName()

Description

Return the standard name for an elliptic curve.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.6 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 175.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.7 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 268.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.8 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.9 SSL_MEM_GetContext()

Description

Get the default memory allocation context for the SSL module.

Prototype

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

Parameters

Parameter	Description
ppMem	Pointer to pointer that receives the address of the default memory context.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.10 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 177

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.11 SSL_PROTOCOL_GetText()

Description

Decode the TLS protocol to a textual representation.

Prototype

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

Parameters

Parameter	Description
Protocol	TLS protocol ID.

Return value

Nonzero pointer to the protocol name.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.12 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 54.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.13 SSL_ROOT_CERTIFICATE_AddDER()

Description

Add root certificate from DER-encoded certificate.

Prototype

```
int SSL_ROOT_CERTIFICATE_AddDER(const U8       * pData,
                                unsigned   DataLen);
```

Parameters

Parameter	Description
pData	Pointer to DER-encoded certificate data.
DataLen	Octet length of the DER-encoded certificate data.

Return value

≥ 0 Certificate successfully loaded.
< 0 Error loading certificate.

Additional information

The DER-encoded certificate data is NOT copied from pData. You must ensure that the certificate data at pData remain allocated and accessible until the SSL stack is shut down.

The DER-encoded certificate length must be exactly DataLen octets in size: data beyond the end of the valid DER-encoded certificate is not ignored and is considered an error.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.14 SSL_ROOT_CERTIFICATE_LoadDER()

Description

Load root certificate from DER-encoded certificate.

Prototype

```
int SSL_ROOT_CERTIFICATE_LoadDER(      SSL_ROOT_CERTIFICATE * pRootCert,
                                 const U8                   * pData,
                                       unsigned               DataLen);
```

Parameters

Parameter	Description
pRootCert	Pointer to object that receives the root certificate.
pData	Pointer to DER-encoded certificate data.
DataLen	Octet length of the DER-encoded certificate data.

Return value

≥ 0 Certificate successfully loaded.
< 0 Error loading certificate.

Additional information

The DER-encoded certificate data is NOT copied from pData. You must ensure that the certificate data at pData remain allocated and accessible until the SSL stack is shut down.

The DER-encoded certificate length must be exactly DataLen octets in size: data beyond the end of the valid DER-encoded certificate is not ignored and is considered an error.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.15 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 137.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.16 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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.17 SSL_SIGNATURE_ALGORITHM_Add()

Description

Add signature algorithm to emSSL.

Prototype

```
void SSL_SIGNATURE_ALGORITHM_Add(SSL_SIGNATURE_SCHEME_ID 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 176.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.18 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 175.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.19 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 175.

 © 2015-2026 SEGGER Microcontroller GmbH

4.5.20 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;
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
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;
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
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;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_RC4_128_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_3DES_EDE_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_128_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_256_CBC_SHA;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_128_GCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_256_GCM_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_128_CCM_8_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_AES_128_CCM_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256;
extern const SSL_SUITE SSL_SUITE_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384;
```

See also

Adding cipher suites on page 171.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6 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_GetRecordSizeLimit()	Get the record size limit.
SSL_SESSION_GetSessionID()	Get the Session ID.
SSL_SESSION_GetSocket()	Get the Socket used.
SSL_SESSION_GetSuite()	Retrieve the active suite for an SSL session.
SSL_SESSION_Prepare()	Prepare SSL session before connection.
SSL_SESSION_Process()	Run SSL state machine (nonblocking mode).
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_SetCertificateAPI()	Set the API to use to handle PKI certificates and keys.
SSL_SESSION_SetFlags()	Set session-related flags.
SSL_SESSION_SetProtocolRange()	Set the protocol range supported by the SSL connection.
SSL_SESSION_SetPSK()	Set the PSK to use for this connection.
SSL_SESSION_SetRecordSizeLimit()	Sets the record size limit.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.1 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 139.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.2 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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.3 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 or IP address of server we wish to connect to.

Return value

≥ 0	Success.
= SSL_ERROR_AGAIN	Operation in progress, call SSL_SESSION_Process() to continue.
< 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 and checked against the Common Name or Subject Alternative Name in the presented X.509 certificate.

If sServerName is a valid dotted-decimal IPv4 address, it is checked against the IP address presented in the Subject Alternative Name extension in the presented X.509 certificate.

If sServerName is the null pointer, the Server Name Indication extension is not included in the SSL handshake and no name check is performed against the presented X.509 certificate.

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

In nonblocking mode, call SSL_SESSION_Process() to run the SSL state machine and process incoming data on the socket.

See also

SSL_SESSION_SetProtocolRange on page 139.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.4 SSL_SESSION_Disconnect()

Description

Disconnect a client or server SSL connection.

Prototype

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

Parameters

Parameter	Description
pSelf	Pointer to session context.

Return value

≥ 0	Success.
= SSL_ERROR_AGAIN	Operation in progress, call SSL_SESSION_Process() to continue.
< 0	Processing error.

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.

In nonblocking mode with return status SSL_ERROR_AGAIN, call SSL_SESSION_Process() until it returns a status other than SSL_ERROR_AGAIN.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.5 SSL_SESSION_GetRecordSizeLimit()

Description

Get the record size limit.

Prototype

```
unsigned SSL_SESSION_GetRecordSizeLimit(const SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session.

Return value

Record size limit.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.6 SSL_SESSION_GetSessionID()

Description

Get the Session ID.

Prototype

```
void SSL_SESSION_GetSessionID(const SSL_SESSION  * pSelf,
                              const U8          ** ppSID,
                                    unsigned     * pSIDLen);
```

Parameters

Parameter	Description
pSelf	Pointer to session.
ppSID	Receives a pointer to the session ID.
pSIDLen	Receives the length of the session ID.

Additional information

This function returns a pointer to the current session ID and its length. Note that the session ID is not copied; if the caller needs a persistent copy, they should create one themselves. The caller may not modify or free the session ID.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.7 SSL_SESSION_GetSocket()

Description

Get the Socket used.

Prototype

```
int SSL_SESSION_GetSocket(const SSL_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to session.

Return value

Socket.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.8 SSL_SESSION_GetSuite()

Description

Retrieve the active suite for an SSL session.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.9 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 139.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.10 SSL_SESSION_Process()

Description

Run SSL state machine (nonblocking mode).

Prototype

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

Parameters

Parameter	Description
pSelf	Pointer to session context.

Return value

≥ 0	Completed, success.
= SSL_ERROR_AGAIN	Operation in progress, call SSL_SESSION_Process() to continue.
< 0	Processing error.

Additional information

This function runs the SSL state machine to complete an operation with the session set to nonblocking mode. Do not call this function if the session is set to blocking mode.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.11 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

In blocking mode, 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.

In nonblocking mode, all immediately available data, up to the number of bytes requested, are copied into the receiving object and the number of bytes copied returned. If no data are immediately available, SSL_ERROR_AGAIN is returned. All other negative values indicate a protocol error or socket closure.

Do not call SSL_SESSION_Process() upon reception of an SSL_ERROR_AGAIN status, simply call SSL_SESSION_Receive() again for more data.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.12 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_CERTIFICATE	SSL server received a valid certificate from the client during connection setup.
SSL_SESSION_FLAG_REQUEST_PREVENT_FALLBACK	SSL client adds the Fallback signalling ciphersuite to the Client Hello message (if required) to prevent version rollback.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.13 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	Success.
= SSL_ERROR_AGAIN	Operation in progress, call SSL_SESSION_Process() to continue.
< 0	Processing error.

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.

In nonblocking mode with return status SSL_ERROR_AGAIN, call SSL_SESSION_Process() until it returns a status other than SSL_ERROR_AGAIN.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.14 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	Success.
= SSL_ERROR_AGAIN	Operation in progress, call SSL_SESSION_Process() to continue.
< 0	Processing error.

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.

In nonblocking mode with return status SSL_ERROR_AGAIN, call SSL_SESSION_Process() until it returns a status other than SSL_ERROR_AGAIN.

See also

SSL_SESSION_Send on page 133.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.15 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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.16 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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.17 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-

 © 2015-2026 SEGGER Microcontroller GmbH

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

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.18 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.

Note that this function sets the TLS version to use, independent of the transport layer. For reference, DTLS 1.0 uses the TLS 1.1 protocol and DTLS 1.2 uses the TLS 1.2 protocol.

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);
```

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.19 SSL_SESSION_SetPSK()

Description

Set the PSK to use for this connection.

Prototype

```
void SSL_SESSION_SetPSK(      SSL_SESSION * pSelf,
                        const U8          * pPSK,
                              unsigned      PSKLen);
```

Parameters

Parameter	Description
pSelf	Pointer to active session.
pPSK	Pointer to PSK octet string.
PSKLen	Octet length of the PSK octet string.

Additional information

This function must be called, when using PSK cipher suites, to set the PSK to use.

 © 2015-2026 SEGGER Microcontroller GmbH

4.6.20 SSL_SESSION_SetRecordSizeLimit()

Description

Sets the record size limit. Must be in range [64, 16384].

Prototype

```
int SSL_SESSION_SetRecordSizeLimit(SSL_SESSION * pSelf,
                                   unsigned      RecordSizeLimit);
```

Parameters

Parameter	Description
pSelf	Pointer to session.
RecordSizeLimit	The record size limit value.

Return value

≥ 0 Success.
< 0 Failure (value out of range).

Additional information

The record size limit has two effects. First, it sets the maximum payload size for application packets that this device *sends* on the record layer. Second, it is used during the handshake to indicate the maximum record size that this device is willing to *receive* (via the Record Size Limit Extension, RFC8449).

To impact the value that is used during the handshake, this function must be called after initializing the session through SSL_SESSION_Prepare() and before making or accepting connections using SSL_SESSION_Connect() or SSL_SESSION_Accept(). The function can also be called at a later point to limit the size of outgoing packets only.

In both cases, the total size of the final record layer packet on the wire will be slightly larger, as this count does not include padding and such (the exact overhead depends on the selected TLS version and cipher suite).

Reducing the record size limit can help reduce message size errors (EMSGSIZE) when using emSSL with UDP sockets.

If not set, the record size limit defaults to the compile-time definition SSL_MAX_APP_DATA_FRAGMENT_LEN.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7 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_GetIanaName()	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?

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.1 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.2 SSL_SUITE_FindByID()

Description

Find an installed cipher suite by IANA ID.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.3 SSL_SUITE_FindByIndex()

Description

Find an installed cipher suite by preference index.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.4 SSL_SUITE_GetCipherName()

Description

Get the cipher name for a cipher ID.

Prototype

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

Parameters

Parameter	Description
ID	Cipher ID.

Return value

Zero-terminated string describing ID.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.5 SSL_SUITE_GetIanaName()

Description

Get IANA name of a SSL suite ID.

Prototype

```
const char *SSL_SUITE_GetIanaName(unsigned ID);
```

Parameters

Parameter	Description
ID	TLS suite ID.

Return value

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

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.6 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

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.7 SSL_SUITE_GetKeyExchangeName()

Description

Get the name of a key exchange ID.

Prototype

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

Parameters

Parameter	Description
ID	Key exchange ID.

Return value

Zero-terminated string describing ID.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.8 SSL_SUITE_GetMACAlgorithmName()

Description

Get the algorithm name for a MAC algorithm ID.

Prototype

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

Parameters

Parameter	Description
ID	MAC algorithm ID.

Return value

Zero-terminated string describing ID.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.9 SSL_SUITE_GetPKAlgorithmName()

Description

Get the name of a public key algorithm ID.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.10 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.11 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.12 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 return false.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.13 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.7.14 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

 © 2015-2026 SEGGER Microcontroller GmbH

4.8 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.1 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 81, *SSL_RemoveLogFilter* on page 161.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.2 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 164 for a list of warnings.
- *SSL_RemoveWarnFilter* on page 162.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.3 SSL_ERROR_GetText()

Description

Decode an SSL error code.

Prototype

```
const 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.4 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_RemoveLogFilter(SSL_LOG_ERROR);
Status   = SSL_SESSION_Connect(&Session, "www.segger.com");
SSL_SetLogFilter(Previous);
```

See also

Logging flags on page 81, *SSL_AddLogFilter* on page 158.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.5 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 warning filter.

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.6 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 81
- *SSL_AddLogFilter* on page 158
- *SSL_RemoveLogFilter* on page 161

 © 2015-2026 SEGGER Microcontroller GmbH

4.8.7 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 83
- *SSL_AddWarnFilter* on page 159.
- *SSL_RemoveWarnFilter* on page 162.

 © 2015-2026 SEGGER Microcontroller GmbH

4.9 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.

 © 2015-2026 SEGGER Microcontroller GmbH

4.10 API evolution

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

Version 3.10.0

The following are the API changes in this version:

- Renamed `SSL_SUITE_GetIANASuiteName()` to `SSL_SUITE_GetIanaName()` with no change in function. A macro maintains backward source code compatibility.
- Added `SSL_LOG_API` to control logging of significant API entries and exits.

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_GetIanaName()`.
- `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.

 © 2015-2026 SEGGER Microcontroller GmbH

- Added `SSL_SESSION_SetFlags()` to set per-session configuration flags.
- Added capability to request and respond with client certificates.

 © 2015-2026 SEGGER Microcontroller GmbH

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 271 for details on best practices when configuring SSL and TLS systems.

 © 2015-2026 SEGGER Microcontroller GmbH

5.1 Compile-time configuration

Some aspects of emSSL can be configured at compile time. The following compile-time configuration macros can be set in the file `SSL_Conf.h`. Their default values are defined in the file `SSL_ConfDefaults.h`.

5.1.1 SSL_DEBUG

Description

Debug level for emSSL. Supported values are:

- `0`: No checks (smallest and fastest code).
- `1`: Warnings & Panic checks.
- `2`: Warnings, logs, & panic checks (significantly bigger code).

Definition

```
#define SSL_DEBUG    0
```

5.1.2 SSL_LOG_BUFFER_SIZE

Description

This symbol defines the maximum size of the emSSL log buffer, which determines the maximum length of a single log message. Only relevant when logging is enabled. The maximum value is only limited by the amount of memory available on the target system.

Definition

```
#define SSL_LOG_BUFFER_SIZE    320
```

5.1.3 SSL_MAX_SESSION_TICKET_LEN

Description

This 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.

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. In practice, ticket sizes vary considerably in length, from a few hundred bytes up to two kilobytes. The theoretical maximum according to RFC 5077 is 65535 bytes.

If this is set to 0, support for session tickets is disabled. The maximum value is only limited by the amount of memory available on the target system.

Definition

```
#define SSL_MAX_SESSION_TICKET_LEN    0
```

5.1.4 SSL_TRANSPORT_API_GET_TIME_FUNC

Description

Set this symbol to configure a function that returns the current (wall clock) time as a U32 UNIX timestamp. This function is used in the sample `SSL_TRANSPORT_API` implementations for embedded systems.

The function must match the signature of `SSL_GET_TIME_FUNC` in SSL.h.

If you use your own `SSL_TRANSPORT_API`, this configuration has no effect. Set `pfGetTime` there instead.

 © 2015-2026 SEGGER Microcontroller GmbH

The wall clock time is required to detect expired certificates. If it is not available (i.e., set to NULL), the expiry time cannot be checked and expired certificates may be accepted.

Definition

```
#define SSL_TRANSPORT_API_GET_TIME_FUNC    NULL
```

5.1.5 SSL_MAX_APP_DATA_FRAGMENT_LEN

Description

Set this 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.

Also consider the limitations w.r.t. UDP transports mentioned in *UDP datagram size* on page 65.

Definition

```
#define SSL_MAX_APP_DATA_FRAGMENT_LEN    1024
```

5.1.6 SSL_DTLS_CONTENT_SIZE

Description

Set this symbol to define the maximum size of a DTLS fragment accepted by emSSL. The default is 16384, the maximum TLS fragment size. The valid range is from 1 to 16384 inclusive, though smaller values risk that incoming data is truncated.

Definition

```
#define SSL_DTLS_CONTENT_SIZE    16384
```

5.1.7 SSL_SESSION_CACHE_SIZE

Description

Set this symbol to define the number of entries in the session cache. Setting this symbol to zero disables the session cache. The maximum value is only limited by the amount of memory available on the target system.

Definition

```
#define SSL_SESSION_CACHE_SIZE    0
```

5.1.8 SSL_AUDIT_ENABLED

Description

Set this symbol non-zero to enable the emSSL audit subsystem, which notifies the application developer about certain events.

See *Audit events* on page 66 for more information.

Definition

```
#define SSL_AUDIT_ENABLED    0
```

 © 2015-2026 SEGGER Microcontroller GmbH

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 171.
- Bulk encryption ciphers, such as DES and AES. See *Adding ciphers* on page 174.
- Hash and MAC algorithms, such as SHA and MD5. See *Adding MACs* on page 175.
- Signature verification schemes required. See *Adding public key signature verifiers* on page 175.
- Message signing schemes required. See *Adding public key message signers* on page 175.
- Signature schemes offered. See *Adding signature algorithms* on page 176.
- Elliptic curves, if using elliptic curve cryptography. See *Adding elliptic curves* on page 177.
- TLS protocol versions that you wish to support. See *Adding TLS protocols* on page 177.

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 175), bulk ciphers (*Adding ciphers* on page 174), and message authentication (*Adding MACs* on page 175).

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);
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

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

 © 2015-2026 SEGGER Microcontroller GmbH

- 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

 © 2015-2026 SEGGER Microcontroller GmbH

- 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 215).

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 emCrypt* on page 190 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);
```

 © 2015-2026 SEGGER Microcontroller GmbH

5.2.4 Adding MACs

You must add the 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 emCrypt* on page 190 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 emCrypt* on page 190 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 (according to the cipher suites you select) using SSL_SIGNATURE_SIGN_Add(). The public key message signer implementations are:

- SSL_SIGNATURE_SIGN_RSA_API
- SSL_SIGNATURE_SIGN_ECDSA_API

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

 © 2015-2026 SEGGER Microcontroller GmbH

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 emCrypt* on page 190 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_SIGN_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 emCrypt* on page 190 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);
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
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 similarly burdened, 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 279 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 versions 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`

 © 2015-2026 SEGGER Microcontroller GmbH

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, which 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. Setting aside the configuration of the cipher suite and other items, and focusing on the PRF, this 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 while retaining the single TLS 1.0 cipher suite requires adding PRF-SHA256 — this is because the MAC for the TLS version 1.0 cipher suite 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 adding 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 in which you add components to emSSL does not matter; you can add them in whatever order feels most natural. When `SSL_X_Config()` returns, `SSL_Init()` will validate the configuration and warn you about anything that can be removed, and it will halt if a required component has not been configured.

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

 © 2015-2026 SEGGER Microcontroller GmbH

5.3 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.

5.3.1 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.
SSL_OS_GetTime32()	Return the current system time in ms.
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.

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.1.1 SSL_OS_Init()

Description

Initialize SSL stack.

Prototype

```
void SSL_OS_Init(void);
```

Additional information

Creates and initializes all objects required for task synchronization.

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.1.2 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.

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.1.3 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.

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.1.4 SSL_OS_GetTime32()

Description

Return the current system time in ms. The value will wrap around after app. 49.7 days.

Prototype

```
U32 SSL_OS_GetTime32(void);
```

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.2 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();
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
/*********************************************************************
*
*       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);
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

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

 © 2015-2026 SEGGER Microcontroller GmbH

5.3.3 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 */
}

/*********************************************************************
*
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
*       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";
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

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

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 6

Configuring emCrypt

 © 2015-2026 SEGGER Microcontroller GmbH

6.1 Overview

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

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

There are three parts to emCrypt configuration:

- *Runtime configuration*, where cryptographic services are installed when initializing the cryptographic library. Runtime configuration happens in a `CRYPTO_X_Config()` function that is implemented by your application.
- *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. Compile-time flags are configured through macros in the `CRYPTO_Conf.h` file.
- *Operating system bindings* define how emCrypt locks hardware resources against simultaneous access by different threads. This is especially relevant when hardware acceleration is used in a multithreaded environment.

The following sections describe the configuration of emCrypt in detail.

 © 2015-2026 SEGGER Microcontroller GmbH

6.2 Runtime configuration

Your application must provide two functions with external linkage: `CRYPTO_X_Config()` and `CRYPTO_X_Panic()`.

You use `CRYPTO_X_Config()` to configure the capabilities of emCrypt. This function is called internally by the initialization function. The function `CRYPTO_X_Panic()` defines what happens if something unforeseen happens.

Sample implementations of `CRYPTO_X_Panic()` and `CRYPTO_X_Config()` can be found in the `Sample/Config` directory in the shipping package. Additional samples that demonstrate the use of hardware acceleration are discussed in the chapter *Hardware acceleration* on page 230.

6.2.1 CRYPTO_X_Config()

In general, you need to do three things in the `CRYPTO_X_Config()` function:

1. Install the cryptographic algorithm implementations you want to use.
2. Install a source of random numbers.
3. Install the modular exponentiation implementation to use.

6.2.1.1 Installing cryptographic algorithm implementations

For some algorithms, emCrypt allows you to choose between multiple implementations. For the SHA256 hash function, for example, emCrypt offers a generic software implementation that runs on most systems, and also various hardware-accelerated implementations that only run on specific microcontrollers (see *Hardware acceleration* on page 230).

Therefore, for each such algorithm, you need to define a preferred and a fallback implementation. The fallback implementation may be `NULL`. For instance:

```
void CRYPTO_X_Config(void) {
  // ...
  CRYPTO_<algorithm>_Install(&CRYPTO_CIPHER_<algorithm>_SW, NULL);
  // ...
}
```

The concrete installation procedure for the software implementation of an algorithm is described in the *emCrypt reference* on page 211 (e.g., *SHA256* on page 212). In addition, the chapter *Hardware acceleration* contains examples that show how to install the respective hardware implementations (e.g., *Installing CRYP hardware support* on page 245 for the STM32 CRYP coprocessor).

6.2.1.2 Installing a random number source

In order to guarantee the privacy of communications, it is vital that emSSL can call upon a stream of random numbers that cannot be predicted. For example, unpredictable random numbers are required to generate unique cryptographic keys. If an attacker could predict or guess the sequence of random numbers used for key generation, they can easily deduce the generated keys, allowing them to decrypt messages or impersonate users or devices.

To prevent such attacks, it is crucial to use a good source 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 uses 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.

 © 2015-2026 SEGGER Microcontroller GmbH

For devices that have no true random source, a typical approach is 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.

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.

You typically call `CRYPTO_RNG_Install()` in your `CRYPTO_X_Config()` function.

Hardware-only random sources

emCrypt has add-on drivers for the selected hardware RNG peripherals. See chapter *Hardware acceleration* on page 230. You can install the hardware source as both the random bit generator and the source of entropy, for example:

```
void CRYPTO_X_Config(void) {
  // ...
  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.

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:

```
void CRYPTO_X_Config(void) {
  // ...
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                       &CRYPTO_RNG_HW_Kinetis_RNGA);
  // ...
}
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.2.1.3 Installing modular exponentiation algorithms

Some public-key crypto algorithms rely on modular exponentiation operations. There are various ways to implement this operation, and you need to choose one implementation to use. To do so, call the functions `CRYPTO_MPI_SetPublicModExp()` and `CRYPTO_MPI_SetPrivateModExp()`. For example:

```
void CRYPTO_X_Config(void) {
  // ...

  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}
```

Our general recommendation is to choose `CRYPTO_MPI_ModExp_Basic_Fast`, unless we provide a sample configuration file for your specific MCU that selects a different implementation.

Refer to the emCrypt manual for a list of available alternatives.

6.2.2 CRYPTO_X_Panic()

This function is called by the library when unexpected conditions occur, such as a failing assertion (`CRYPTO_ASSERT()`). Note that assertions are typically only evaluated during development (specifically, when the compile-time macro `DEBUG` is defined and nonzero, or if the macro *`CRYPTO_API_CHECKS`* is set to `1`).

Most samples react to a panic by entering an empty infinite loop. Alternative implementations could emit an error message or, on PC platforms, exit the application with an error code.

6.2.3 Example runtime configurations

6.2.3.1 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.
*/

/*********************************************************************
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
*
*       #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
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  };
  //
  // 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 ***************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.3 Compile-time configuration

For many compute-intensive algorithms, emCrypt offers multiple implementation variants that vary in code size and performance. Smaller implementations are typically slower, while larger implementations tend to be faster. In order to configure how compute-intensive cryptographic algorithms are compiled, you can set several compile-time configurations in the configuration file `CRYPTO_Conf.h`.

By default, emCrypt is configured to select a *balanced* configuration that provides reasonable performance at a reasonable code size for most applications. It is recommended to try the balanced default configuration first and assess its performance. If more performance is needed or the code size needs to be reduced, the configuration can be adapted as described below.

6.3.1 Optimization goals

Algorithms are selected based on the *optimization goal* configuration. An optimization goal specifies the application developer's preference in the tradeoff between code size and performance.

Optimization goals can be configured at two levels:

- *Algorithm-specific* optimization goals specify the preferred goal for a particular algorithm.
- The *global* optimization goal sets the general preference. It is applied to all algorithms that do not have an algorithm-specific optimization goal set (fallback).

There are seven optimization goals to choose from:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`	Lowest resource requirements without performance restriction. Should be used on constrained systems only.
`SEGGER_OPT_GOAL_SIZE_SMALL`	Favor low resource requirements at reasonable performance.
`SEGGER_OPT_GOAL_SIZE`	Favor lower resource requirements at slightly lower performance.
`SEGGER_OPT_GOAL_BALANCED`	Balance between performance and resource requirements. The common case default.
`SEGGER_OPT_GOAL_SPEED`	Favor performance at slightly higher resource requirements.
`SEGGER_OPT_GOAL_SPEED_HIGH`	High performance with reasonable resource requirements.
`SEGGER_OPT_GOAL_SPEED_MAX`	Maximum performance without resource restriction. May fit only larger systems.

By default, the global optimization goal is set to `SEGGER_OPT_GOAL_BALANCED`, and no algorithm-specific optimization goals are defined. If you want to change the global optimization goal, set the `CRYPTO_OPT_GOAL_GLOBAL` compile-time macro in `CRYPTO_Conf.h`. For example:

```
#define CRYPTO_OPT_GOAL_GLOBAL  SEGGER_OPT_GOAL_SPEED
```

Any algorithm-specific optimization goals can be configured in the `CRYPTO_Conf.h` file as well.

Many algorithms have fewer than seven different implementations. In that case, a single implementation maps to multiple goals, ensuring that every optimization goal is a valid selection for each algorithm. The characteristics of the respective algorithms at each optimization goal level are explained in this manual (e.g., *SHA256* on page 221).

 © 2015-2026 SEGGER Microcontroller GmbH

Regardless of the selected implementation, the actual code size and performance depend on various factors, such as the MCU's architecture, the chosen compiler, compiler version, and flags.

> **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`.

6.3.2 Lookup table storage location

Some algorithm implementations use lookup tables to speed up computations. Those tables live in flash memory by default, but can also be copied into RAM. RAM-based tables may yield better performance on some MCUs.

Just like the optimization goals, lookup table storage locations can also be configured at two levels:

- The *algorithm-specific* storage location specifies the preferred location for a particular algorithm.
- The *global* storage location sets the general preference. It is applied to all algorithms that do not have an algorithm-specific storage location set (fallback).

The global storage location can be configured by setting the `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL` compile-time macro in `CRYPTO_Conf.h`:

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL  0
```

- Value `0` indicates that tables are stored in flash. This is the default.
- Value `1` indicates that tables are stored in RAM.

The algorithm-specific storage location configuration macros can be set in `CRYPTO_Conf.h` as well.

6.3.3 Compile-time configuration reference

The following compile-time configuration macros can be set in the file `CRYPTO_Conf.h`. Their default values are defined in the file `CRYPTO_ConfDefaults.h`.

6.3.3.1 CRYPTO_API_CHECKS

Description

Configure whether emCrypt checks parameters of API function calls.

Setting `CRYPTO_API_CHECKS` nonzero, typically to `1`, will enable parameter checks at the API level and we strongly suggest that you set this to `1` when developing. Setting `CRYPTO_API_CHECKS` to `0` disables all API parameter checks and assumes that clients will only pass valid parameters to exposed API functions. If you violate this condition, the behavior of the crypto library is undefined.

In general, this ensures that MPIs that are written to must indeed be writable and not in ROM.

In the default configuration, `CRYPTO_API_CHECKS` is set to `1` if the macro `DEBUG` is defined and nonzero. It is set to `0` otherwise.

Definition

```
#define CRYPTO_API_CHECKS    1
#define CRYPTO_API_CHECKS    0
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.3.3.2 CRYPTO_MPI_BITS_PER_LIMB

Description

Configure number of bits per limb for multiprecision integers (MPIs). MPIs store and manipulate big numbers. If a number is too big to fit into a single register, it has to be broken into multiple "limbs". This configuration determines the size of each limb in bits.

Supported configurations are:

- `32`, which 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 support 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--you would need to run a benchmark on your target and compare encryption and decryption speeds for 8-bit and 16-bit limb sizes to pick an optimal configuration.

Definition

```
#define CRYPTO_MPI_BITS_PER_LIMB    32
```

6.3.3.3 CRYPTO_CONFIG_BYTE_ORDER

Description

Configure the endianness of the target. By default, the endianness is determined automatically based on compile-time definitions (see file `SEGGER_UTIL.h`). Setting it manually overrides the automatically determined endianness.

Supported configurations are:

- `-1`: Little endian and misaligned loads/stores never fault
- `0`: Endian neutral or misaligned loads/stores may fault
- `1`: Big endian and misaligned loads/stores never fault

Definition

```
#define CRYPTO_CONFIG_BYTE_ORDER    SEGGER_UTIL_CONFIG_BYTE_ORDER
```

6.3.3.4 CRYPTO_OPT_GOAL_GLOBAL

Description

For some algorithms, emCrypt offers multiple implementations that vary in code size and performance. Smaller implementations are typically slower, while larger implementations tend to be faster. The algorithm is selected at compile time based on the optimization goal configuration.

This configuration sets the GLOBAL preference. It defaults to BALANCED and is applied to all algorithms that do not have an algorithm-specific optimization goal set (fallback).

There are seven optimization goals:

- `SEGGER_OPT_GOAL_SIZE_MIN` Lowest resource requirements without performance restriction. Should be used on constrained systems only.
- `SEGGER_OPT_GOAL_SIZE_SMALL` Favor low resource requirements at reasonable performance.
- `SEGGER_OPT_GOAL_SIZE` Favor lower resource requirements at slightly lower performance.
- `SEGGER_OPT_GOAL_BALANCED` Balance between performance and resource requirements. Should be the common case default.
- `SEGGER_OPT_GOAL_SPEED` Favor performance at slightly higher resource requirements.
- `SEGGER_OPT_GOAL_SPEED_HIGH` High performance with reasonable resource requirements.

 © 2015-2026 SEGGER Microcontroller GmbH

- SEGGER_OPT_GOAL_SPEED_MAX Maximum performance without resource restriction. May fit only larger systems.

Definition

```
#define CRYPTO_OPT_GOAL_GLOBAL     SEGGER_OPT_GOAL_BALANCED
```

6.3.3.5 CRYPTO_OPT_GOAL_AES

Description

Algorithm-specific optimization goal for the AES software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_AES     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.6 CRYPTO_OPT_GOAL_CAMELLIA

Description

Algorithm-specific optimization goal for the Camellia software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_CAMELLIA     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.7 CRYPTO_OPT_GOAL_SEED

Description

Algorithm-specific optimization goal for the SEED software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_SEED     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.8 CRYPTO_OPT_GOAL_GCM

Description

Algorithm-specific optimization goal for the software implementation of the GCM cipher mode.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

 © 2015-2026 SEGGER Microcontroller GmbH

Definition

```
#define CRYPTO_OPT_GOAL_GCM     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.9 CRYPTO_OPT_GOAL_MD5

Description

Algorithm-specific optimization goal for the MD5 software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_MD5     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.10 CRYPTO_OPT_GOAL_SHA1

Description

Algorithm-specific optimization goal for the SHA1 software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_SHA1     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.11 CRYPTO_OPT_GOAL_SHA256

Description

Algorithm-specific optimization goal for the SHA256 software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_SHA256     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.12 CRYPTO_OPT_GOAL_SHA512

Description

Algorithm-specific optimization goal for the SHA512 software implementation.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_SHA512     CRYPTO_OPT_GOAL_GLOBAL
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.3.3.13 CRYPTO_OPT_GOAL_ECDSA_MULTIPLY

Description

Algorithm-specific optimization goal for the multiplication operation during ECDSA verification.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_ECDSA_MULTIPLY     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.14 CRYPTO_OPT_GOAL_MPI_BIT_OPS

Description

Algorithm-specific optimization goal for the bit operations on multiprecision integers (MPIs).

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_MPI_BIT_OPS     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.15 CRYPTO_OPT_GOAL_AES_HW

Description

Algorithm-specific optimization goal for some AES hardware implementations.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_AES_HW     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.16 CRYPTO_OPT_GOAL_MD5_HW

Description

Algorithm-specific optimization goal for some MD5 hardware implementations.

Supported configurations are any of the seven optimization goals or CRYPTO_OPT_GOAL_GLOBAL.

By default, this configuration inherits the global optimization goal (CRYPTO_OPT_GOAL_GLOBAL).

Definition

```
#define CRYPTO_OPT_GOAL_MD5_HW     CRYPTO_OPT_GOAL_GLOBAL
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.3.3.17 CRYPTO_OPT_GOAL_SHA1_HW

Description

Algorithm-specific optimization goal for some SHA1 hardware implementations.

Supported configurations are any of the seven optimization goals or `CRYPTO_OPT_GOAL_GLOBAL`.

By default, this configuration inherits the global optimization goal (`CRYPTO_OPT_GOAL_GLOBAL`).

Definition

```
#define CRYPTO_OPT_GOAL_SHA1_HW     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.18 CRYPTO_OPT_GOAL_SHA256_HW

Description

Algorithm-specific optimization goal for some SHA256 hardware implementations.

Supported configurations are any of the seven optimization goals or `CRYPTO_OPT_GOAL_GLOBAL`.

By default, this configuration inherits the global optimization goal (`CRYPTO_OPT_GOAL_GLOBAL`).

Definition

```
#define CRYPTO_OPT_GOAL_SHA256_HW     CRYPTO_OPT_GOAL_GLOBAL
```

6.3.3.19 CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL

Description

Some algorithm implementations use lookup tables to speed up certain computations. These tables live in flash memory by default, but can also be copied into RAM. RAM-based tables may yield better performance on some MCUs.

This configuration sets the GLOBAL preference. It defaults to `0` and is applied to all algorithms that do not have an algorithm-specific configuration set (fallback).

Supported configurations are:

- `0`: Tables are stored in Flash
- `1`: Tables are stored in RAM

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL     0
```

6.3.3.20 CRYPTO_CONFIG_TABLES_IN_RAM_AES

Description

Algorithm-specific lookup table storage location configuration for the AES software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

 © 2015-2026 SEGGER Microcontroller GmbH

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_AES    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.21 CRYPTO_CONFIG_TABLES_IN_RAM_ARIA

Description

Algorithm-specific lookup table storage location configuration for the ARIA software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_ARIA    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.22 CRYPTO_CONFIG_TABLES_IN_RAM_CAMELLIA

Description

Algorithm-specific lookup table storage location configuration for the CAMELLIA software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_CAMELLIA    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.23 CRYPTO_CONFIG_TABLES_IN_RAM_DES

Description

Algorithm-specific lookup table storage location configuration for the DES software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_DES    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.24 CRYPTO_CONFIG_TABLES_IN_RAM_PRESENT

Description

Algorithm-specific lookup table storage location configuration for the PRESENT software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

 © 2015-2026 SEGGER Microcontroller GmbH

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_PRESENT    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.25 CRYPTO_CONFIG_TABLES_IN_RAM_SEED

Description

Algorithm-specific lookup table storage location configuration for the SEED software implementation.

Supported configurations are `0` (flash), `1` (RAM), or `CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`.

By default, this configuration inherits the global configuration (`CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL`).

Definition

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_SEED    CRYPTO_CONFIG_TABLES_IN_RAM_GLOBAL
```

6.3.3.26 CRYPTO_CONFIG_FORTUNA_POOL_MAX

Description

The number of pools for entropy acquisition by the Fortuna pseudorandom number generator (PRNG). Supported configurations are the natural numbers in range [4, 32].

Definition

```
#define CRYPTO_CONFIG_FORTUNA_POOL_MAX    32
```

6.3.3.27 CRYPTO_CONFIG_FORTUNA_SOURCE_MAX

Description

The maximum number of entropy sources for the Fortuna PRNG. Supported configurations are the natural numbers in range [1, 255].

Definition

```
#define CRYPTO_CONFIG_FORTUNA_SOURCE_MAX    2
```

6.3.3.28 CRYPTO_CONFIG_FORTUNA_RESEED_MAX

Description

Maximum number of pseudorandom data bytes generated by the Fortuna PRNG between reseed events. Supported configurations are the natural numbers in range [1, 1048576].

Definition

```
#define CRYPTO_CONFIG_FORTUNA_RESEED_MAX    (1024u*1024)
```

6.3.3.29 CRYPTO_CONFIG_FORTUNA_GET_MAX

Description

Maximum number of bytes that a single “get” request to the Fortuna PRNG can provide. Supported configurations are the natural numbers in range [1, 1048576].

Definition

```
#define CRYPTO_CONFIG_FORTUNA_GET_MAX    (1024u*1024)
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.4 CRYPTO-OS integration

In a threaded execution environment, individual hardware resources must be protected from simultaneous use by more than one thread. emCrypt handles this by surrounding the use of hardware resources with calls to an OS binding layer.

The functions `CRYPTO_OS_Init()`, `CRYPTO_OS_Claim()`, `CRYPTO_OS_Request()` and `CRYPTO_OS_Unclaim()` must be provided by the application.

To use a shared resource, emCrypt 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, emCrypt uses the resource and then calls `CRYPTO_OS_Unclaim()` to release it. In the case where access to the resource is not granted, emCrypt 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 emCrypt 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 emCrypt initialization.

6.4.1 Sample bindings

emSSL ships with two sample CRYPTO-OS bindings:

- A sample binding for SEGGER embOS, `CRYPTO_OS_embOS.c`.
- A sample binding for a bare metal system that has no tasking, `CRYPTO_OS_None.c`. This file provides empty bindings that can be used when no locking of hardware resources is required.

6.4.2 CRYPTO-OS API

Function	Description
`CRYPTO_OS_Init()`	Initialize CRYPTO binding to OS.
`CRYPTO_OS_Claim()`	Claim a hardware resource.
`CRYPTO_OS_Request()`	Request a hardware resource.
`CRYPTO_OS_Unclaim()`	Release claim on a hardware resource.

 © 2015-2026 SEGGER Microcontroller GmbH

6.4.2.1 CRYPTO_OS_Init()

Description

Initialize CRYPTO binding to OS.

Prototype

```
void CRYPTO_OS_Init(void);
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.4.2.2 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.4.2.3 CRYPTO_OS_Request()

Description

Request 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.4.2.4 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5 emCrypt reference

The following sections are extracted from the full emCrypt documentation for reference.

6.5.1 Runtime API

6.5.1.1 Hashes

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

See also

See *SHA-1* on page 220 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.1.1.2 SHA-256

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_SW;
```

Description

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

Installation

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

See also

See *SHA-256* on page 221 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.1.1.3 SHA-512

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA512_SW;
```

Description

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

Installation

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

See also

See *SHA-512* on page 222 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

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

See also

See *MD5* on page 223 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.1.2 Ciphers

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

See also

See *AES* on page 224 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

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

See also

See *DES* on page 225 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

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

See also

See *SEED* on page 226 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

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

See also

See *ARIA* on page 227 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

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

See also

See *Camellia* on page 228 for details on how to configure the performance and footprint of this algorithm.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2 Compile-time API

6.5.2.1 Hashes

6.5.2.2 SHA-1

Default

```
#define CRYPTO_OPT_GOAL_SHA1              SEGGER_OPT_GOAL_BALANCED
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	Loop-based implementation
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Loops unrolled, increased ROM footprint

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	1.01	0.01	120
`SEGGER_OPT_GOAL_SIZE_SMALL`	1.01	0.01	120
`SEGGER_OPT_GOAL_SIZE`	1.01	0.01	120
`SEGGER_OPT_GOAL_BALANCED`	1.01	0.01	120
`SEGGER_OPT_GOAL_SPEED`	4.91	0.01	120
`SEGGER_OPT_GOAL_SPEED_HIGH`	4.91	0.01	120
`SEGGER_OPT_GOAL_SPEED_MAX`	4.91	0.01	120

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.3 SHA-256

Default

```
#define CRYPTO_OPT_GOAL_SHA256              SEGGER_OPT_GOAL_BALANCED
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	Loop-based implementation
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Loops unrolled, increased ROM footprint

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	1.34	0.01	128
`SEGGER_OPT_GOAL_SIZE_SMALL`	1.34	0.01	128
`SEGGER_OPT_GOAL_SIZE`	1.34	0.01	128
`SEGGER_OPT_GOAL_BALANCED`	1.34	0.01	128
`SEGGER_OPT_GOAL_SPEED`	7.79	0.01	128
`SEGGER_OPT_GOAL_SPEED_HIGH`	7.79	0.01	128
`SEGGER_OPT_GOAL_SPEED_MAX`	7.79	0.01	128

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.4 SHA-512

Default

```
#define CRYPTO_OPT_GOAL_SHA512              SEGGER_OPT_GOAL_BALANCED
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	Loop-based implementation
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`	Loops partially unrolled, increased ROM footprint
`SEGGER_OPT_GOAL_SPEED_MAX`	Loops fully unrolled, increased ROM footprint

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	2.26	0.01	224
`SEGGER_OPT_GOAL_SIZE_SMALL`	2.26	0.01	224
`SEGGER_OPT_GOAL_SIZE`	2.26	0.01	224
`SEGGER_OPT_GOAL_BALANCED`	2.26	0.01	224
`SEGGER_OPT_GOAL_SPEED`	7.39	0.01	224
`SEGGER_OPT_GOAL_SPEED_HIGH`	7.39	0.01	224
`SEGGER_OPT_GOAL_SPEED_MAX`	31.70	0.01	224

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.5 MD5

Default

```
#define CRYPTO_OPT_GOAL_MD5              SEGGER_OPT_GOAL_BALANCED
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`	Loop-based implementation
`SEGGER_OPT_GOAL_BALANCED`, `SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Loops unrolled, increased ROM footprint

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	1.31	0.01	112
`SEGGER_OPT_GOAL_SIZE_SMALL`	1.31	0.01	112
`SEGGER_OPT_GOAL_SIZE`	1.31	0.01	112
`SEGGER_OPT_GOAL_BALANCED`	2.58	0.01	112
`SEGGER_OPT_GOAL_SPEED`	2.58	0.01	112
`SEGGER_OPT_GOAL_SPEED_HIGH`	2.58	0.01	112
`SEGGER_OPT_GOAL_SPEED_MAX`	2.58	0.01	112

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.6 Ciphers

6.5.2.7 AES

Default

```
#define CRYPTO_OPT_GOAL_AES             SEGGER_OPT_GOAL_BALANCED
#define CRYPTO_CONFIG_TABLES_IN_RAM_AES 0
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`	Minimal lookup tables
`SEGGER_OPT_GOAL_BALANCED`	Medium-sized lookup tables
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Large lookup tables

The lookup tables used by this algorithm can be stored in flash or RAM, as explained in *Lookup table storage location* on page 198. Set the `CRYPTO_CONFIG_TABLES_IN_RAM_AES` configuration macro to 1 to move the tables to RAM.

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	LUT Storage	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	Flash	2.42	0.01	256
`SEGGER_OPT_GOAL_SIZE_SMALL`	Flash	2.42	0.01	256
`SEGGER_OPT_GOAL_SIZE`	Flash	2.42	0.01	256
`SEGGER_OPT_GOAL_BALANCED`	Flash	10.27	0.01	256
`SEGGER_OPT_GOAL_SPEED`	Flash	14.37	0.01	256
`SEGGER_OPT_GOAL_SPEED_HIGH`	Flash	14.37	0.01	256
`SEGGER_OPT_GOAL_SPEED_MAX`	Flash	14.37	0.01	256
`SEGGER_OPT_GOAL_SIZE_MIN`	RAM	2.42	0.56	256
`SEGGER_OPT_GOAL_SIZE_SMALL`	RAM	2.42	0.56	256
`SEGGER_OPT_GOAL_SIZE`	RAM	2.42	0.56	256
`SEGGER_OPT_GOAL_BALANCED`	RAM	10.27	8.75	256
`SEGGER_OPT_GOAL_SPEED`	RAM	14.37	12.85	256
`SEGGER_OPT_GOAL_SPEED_HIGH`	RAM	14.37	12.85	256
`SEGGER_OPT_GOAL_SPEED_MAX`	RAM	14.37	12.85	256

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.8 DES

Default

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_DES     0
```

Override

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

Description

The lookup tables used by this algorithm can be stored in flash or RAM, as explained in *Lookup table storage location* on page 198. Set the `CRYPTO_CONFIG_TABLES_IN_RAM_DES` configuration macro to 1 to move the tables to RAM.

Memory footprint

The following table lists the approximate memory footprint of each configuration.

LUT Storage	ROM (KB)	RAM (KB)	Context size (B)
Flash	3.64	0.01	400
RAM	3.64	2.19	400

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.9 SEED

Default

```
#define CRYPTO_OPT_GOAL_SEED              SEGGER_OPT_GOAL_BALANCED
#define CRYPTO_CONFIG_TABLES_IN_RAM_SEED 0
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	Implementation with small lookup tables
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Implementation with large lookup tables

The lookup tables used by this algorithm can be stored in flash or RAM, as explained in *Lookup table storage location* on page 198. Set the `CRYPTO_CONFIG_TABLES_IN_RAM_SEED` configuration macro to 1 to move the tables to RAM.

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	LUT Storage	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	Flash	1.35	0.01	136
`SEGGER_OPT_GOAL_SIZE_SMALL`	Flash	1.35	0.01	136
`SEGGER_OPT_GOAL_SIZE`	Flash	1.35	0.01	136
`SEGGER_OPT_GOAL_BALANCED`	Flash	1.35	0.01	136
`SEGGER_OPT_GOAL_SPEED`	Flash	4.84	0.01	136
`SEGGER_OPT_GOAL_SPEED_HIGH`	Flash	4.84	0.01	136
`SEGGER_OPT_GOAL_SPEED_MAX`	Flash	4.84	0.01	136
`SEGGER_OPT_GOAL_SIZE_MIN`	RAM	1.35	0.58	136
`SEGGER_OPT_GOAL_SIZE_SMALL`	RAM	1.35	0.58	136
`SEGGER_OPT_GOAL_SIZE`	RAM	1.35	0.58	136
`SEGGER_OPT_GOAL_BALANCED`	RAM	1.35	0.58	136
`SEGGER_OPT_GOAL_SPEED`	RAM	4.84	4.17	136
`SEGGER_OPT_GOAL_SPEED_HIGH`	RAM	4.84	4.17	136
`SEGGER_OPT_GOAL_SPEED_MAX`	RAM	4.84	4.17	136

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.10 ARIA

Default

```
#define CRYPTO_CONFIG_TABLES_IN_RAM_ARIA      0
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

The lookup tables used by this algorithm can be stored in flash or RAM, as explained in *Lookup table storage location* on page 198. Set the `CRYPTO_CONFIG_TABLES_IN_RAM_ARIA` configuration macro to 1 to move the tables to RAM.

Memory footprint

The following table lists the approximate memory footprint of each configuration.

LUT Storage	ROM (KB)	RAM (KB)	Context size (B)
Flash	3.25	0.01	288
RAM	3.25	1.08	288

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.11 Camellia

Default

```
#define CRYPTO_OPT_GOAL_CAMELLIA              SEGGER_OPT_GOAL_BALANCED
#define CRYPTO_CONFIG_TABLES_IN_RAM_CAMELLIA 0
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

You can select from different implementation variants of this algorithm by setting an optimization goal. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	S-boxes are expanded at runtime to reduce ROM footprint
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	S-boxes are expanded at compile time for better runtime efficiency

The lookup tables used by this algorithm can be stored in flash or RAM, as explained in *Lookup table storage location* on page 198. Set the `CRYPTO_CONFIG_TABLES_IN_RAM_CAMELLIA` configuration macro to 1 to move the tables to RAM.

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	LUT Storage	ROM (KB)	RAM (KB)	Context size (B)
`SEGGER_OPT_GOAL_SIZE_MIN`	Flash	3.87	0.01	288
`SEGGER_OPT_GOAL_SIZE_SMALL`	Flash	3.87	0.01	288
`SEGGER_OPT_GOAL_SIZE`	Flash	3.87	0.01	288
`SEGGER_OPT_GOAL_BALANCED`	Flash	3.87	0.01	288
`SEGGER_OPT_GOAL_SPEED`	Flash	6.89	0.01	288
`SEGGER_OPT_GOAL_SPEED_HIGH`	Flash	6.89	0.01	288
`SEGGER_OPT_GOAL_SPEED_MAX`	Flash	6.89	0.01	288
`SEGGER_OPT_GOAL_SIZE_MIN`	RAM	3.87	1.03	288
`SEGGER_OPT_GOAL_SIZE_SMALL`	RAM	3.87	1.03	288
`SEGGER_OPT_GOAL_SIZE`	RAM	3.87	1.03	288
`SEGGER_OPT_GOAL_BALANCED`	RAM	3.87	1.03	288
`SEGGER_OPT_GOAL_SPEED`	RAM	6.89	4.10	288
`SEGGER_OPT_GOAL_SPEED_HIGH`	RAM	6.89	4.10	288
`SEGGER_OPT_GOAL_SPEED_MAX`	RAM	6.89	4.10	288

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.2.12 ECDSA

6.5.2.13 ECDSA

Default

```
#define CRYPTO_OPT_GOAL_ECDSA_MULTIPLY              SEGGER_OPT_GOAL_BALANCED
```

Override

To define a non-default value, define the macro in `CRYPTO_Conf.h`.

Description

The ECDSA signature verification step can be accelerated by using Twin point scalar multiplication. The usage of this faster multiplication is controlled by the optimization goal configuration. Refer to *Optimization goals* on page 197 for general guidance on optimization goals.

This algorithm implements the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`	Simple point multiplication is used
`SEGGER_OPT_GOAL_BALANCED`, `SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Twin point multiplication is used

Memory footprint

The following table lists the approximate memory footprint of each configuration.

Optimization Goal	ROM (KB)	RAM (KB)
`SEGGER_OPT_GOAL_SIZE_MIN`	6.56	0.00
`SEGGER_OPT_GOAL_SIZE_SMALL`	6.56	0.00
`SEGGER_OPT_GOAL_SIZE`	6.56	0.00
`SEGGER_OPT_GOAL_BALANCED`	7.29	0.00
`SEGGER_OPT_GOAL_SPEED`	7.29	0.00
`SEGGER_OPT_GOAL_SPEED_HIGH`	7.29	0.00
`SEGGER_OPT_GOAL_SPEED_MAX`	7.29	0.00

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3 Hardware acceleration

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. emSSL can utilize such capabilities on supported devices.

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

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.1 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.

6.5.3.1.1 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);
}
```

6.5.3.1.2 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.1.3 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 ******************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.2 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 managing the cipher mode.

6.5.3.2.1 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.

6.5.3.2.2 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.2.3 Configuration

You can set the following compile-time macros in `CRYPTO_Conf.h` to change the performance characteristics of the code that interacts with the Kinetis CAU hardware unit:

- `CRYPTO_OPT_GOAL_AES_HW`
- `CRYPTO_OPT_GOAL_MD5_HW`
- `CRYPTO_OPT_GOAL_SHA1_HW`
- `CRYPTO_OPT_GOAL_SHA256_HW`

The performance characteristics are configured by means of optimization goals, as explained in the section *Optimization goals* on page 197.

In particular, you can choose from the following levels of optimization:

Optimization goal	Description
`SEGGER_OPT_GOAL_SIZE_MIN`, `SEGGER_OPT_GOAL_SIZE_SMALL`, `SEGGER_OPT_GOAL_SIZE`, `SEGGER_OPT_GOAL_BALANCED`	Loop-based implementation
`SEGGER_OPT_GOAL_SPEED`, `SEGGER_OPT_GOAL_SPEED_HIGH`, `SEGGER_OPT_GOAL_SPEED_MAX`	Loops unrolled, increased ROM footprint

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.2.4 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.
  //
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  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);
  CRYPTO_SM4_Install      (&CRYPTO_CIPHER_SM4_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 ***************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.3 SAMA5D2 cryptographic coprocessors (Add-on)

The Microchip SAMA5D2 features hardware accelerators for cryptographic algorithms, especially the AES and SHA units, as well as a true random number generator (TRNG).

emSSL has support for the following cryptographic primitives:

- Encryption/description using AES-128, AES-192, and AES-256 in ECB, CBC, and GCM modes, including incremental GCM mode.
- The SHA-256 and SHA-224 hash functions.
- The SAMA5D2 true random number generator.

6.5.3.3.1 Installing SAMA5D2 hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_SAMA5D2_AES;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA224_HW_SAMA5D2_SHA;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_SAMA5D2_SHA;
```

You can install hardware-accelerated AES as the *only* implementation using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_SAMA5D2_AES, NULL);
}
```

To install a software fallback as well, use:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_SAMA5D2_AES,
                      &CRYPTO_CIPHER_AES_SW);
}
```

For hash functions, a software fallback always needs to be installed. Use the following code:

```
void CRYPTO_X_Config(void) {
  CRYPTO_SHA224_Install
  (&CRYPTO_HASH_SHA224_HW_SAMA5D2_SHA, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_SAMA5D2_SHA, &CRYPTO_HASH_SHA256_SW);
}
```

6.5.3.3.2 Additional configuration

In order to further increase performance, the following configuration options can be added to the `CRYPTO_Conf.h` file. These options specify the endianess of the system and allow emSSL to use more efficient ways to copy data from and to the hardware accelerators.

```
#define SEGGER_UTIL_CONFIG_BYTE_ORDER_U16    -1
#define SEGGER_UTIL_CONFIG_BYTE_ORDER_U32    -1
#define SEGGER_UTIL_CONFIG_BYTE_ORDER_U64     0
```

6.5.3.3.3 Enabling hardware support

The cryptographic units are clocked through the Power Management Controller (PMC), which needs to be configured first. The following code is sufficient to configure it:

```
volatile U32 *pReg;
pReg = (volatile U32*)0xF0014010u;  // PMC_PCER0
*pReg |= (1u << 9)                  // PMC_PCER0.PID9=1  (AES)
      |  (1u << 12);                // PMC_PCER0.PID12=1 (SHA)
pReg = (volatile U32*)0xF0014100u;  // PMC_PCER1
*pReg |= (1u << (47 - 32));         // PMC_PCER1.PID47=1 (TRNG)
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.3.4 Performance

6.5.3.3.4.1 SHA256

The output of the benchmark CRYPTO_Bench_SHA256 is shown below. It compares the performance of the hardware-accelerated SHA256 implementation with the fastest (but largest) software implementation.

```
emCrypt SHA-256 Benchmark compiled May 28 2026 10:49:04
Copyright (c) 2014-2026 SEGGER Microcontroller GmbH    www.segger.com

Compiler: SEGGER cc 20.1.3
Config:   CRYPTO_VERSION            = 25400 [2.54.0]
Config:   CRYPTO_OPT_GOAL_SHA256    = SEGGER_OPT_GOAL_SPEED_MAX
Config:   CRYPTO_OPT_GOAL_SHA256_HW = SEGGER_OPT_GOAL_SPEED_MAX

+--------------+-----------+
| Algorithm    | Hash MB/s |
+--------------+-----------+
| SHA-224 (SW) |     12.73 |
| SHA-224 (HW) |     27.55 |
| SHA-256 (SW) |     12.73 |
| SHA-256 (HW) |     27.55 |
+--------------+-----------+

Benchmark complete

STOP.
```

6.5.3.3.4.2 AES

The output of the benchmark CRYPTO_Bench_AES is shown below. It compares the performance of the hardware-accelerated AES implementation with the fastest (but largest) software implementation.

```
emCrypt AES Benchmark compiled May 28 2026 10:44:39
Copyright (c) 2014-2026 SEGGER Microcontroller GmbH    www.segger.com

Compiler: SEGGER cc 20.1.3
Config:   CRYPTO_VERSION                  = 25400 [2.54.0]
Config:   CRYPTO_OPT_GOAL_AES             = SEGGER_OPT_GOAL_SPEED_MAX
Config:   CRYPTO_OPT_GOAL_AES_HW          = SEGGER_OPT_GOAL_SPEED_MAX
Config:   CRYPTO_OPT_GOAL_GCM             = SEGGER_OPT_GOAL_SPEED_MAX
Config:   CRYPTO_CONFIG_TABLES_IN_RAM_AES = 1

+--------------+------+-----------------+-----------------+
|              |      | ECB        MB/s | CBC        MB/s |
| Cipher       | Bits |     Enc     Dec |     Enc     Dec |
+--------------+------+-----------------+-----------------+
| AES          |  128 |    5.02    4.90 |    4.76    4.54 |
| AES (HW)     |  128 |   15.48   15.48 |   15.37   15.35 |
| AES          |  192 |    4.19    4.10 |    4.01    3.85 |
| AES (HW)     |  192 |   15.43   15.43 |   15.32   15.31 |
| AES          |  256 |    3.62    3.54 |    3.49    3.35 |
| AES (HW)     |  256 |   13.94   13.94 |   13.92   13.91 |
+--------------+------+-----------------+-----------------+
|              |      | GCM        MB/s | CCM        MB/s |
| Cipher       | Bits |     Enc     Dec |     Enc     Dec |
+--------------+------+-----------------+-----------------+
| AES          |  128 |    2.73    2.72 |    2.33    2.33 |
| AES (HW)     |  128 |   15.33   15.21 |    3.85    3.84 |
| AES          |  192 |    2.45    2.45 |    1.97    1.97 |
| AES (HW)     |  192 |   15.28   15.17 |    3.47    3.47 |
| AES          |  256 |    2.23    2.23 |    1.71    1.71 |
| AES (HW)     |  256 |   13.85   13.77 |    3.04    3.04 |
+--------------+------+-----------------+-----------------+
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
Benchmark AES-GCM incremental performance

+-------+-----------+-----------+-----------+-----------+-----------+-----------+
| Chunk |  AES-128  |  AES-192  |  AES-256  |  AES-128  |  AES-192  |  AES-256  |
| Bytes | (SW) MB/s | (SW) MB/s | (SW) MB/s | (HW) MB/s | (HW) MB/s | (HW) MB/s |
+-------+-----------+-----------+-----------+-----------+-----------+-----------+
|   1   |      1.62 |      1.52 |      1.43 |      2.29 |      2.29 |      2.28 |
|   9   |      2.57 |      2.33 |      2.12 |      4.86 |      4.85 |      4.85 |
|  17   |      2.69 |      2.43 |      2.20 |      5.30 |      5.30 |      5.29 |
|  25   |      2.76 |      2.49 |      2.25 |      5.59 |      5.59 |      5.57 |
|  33   |      2.80 |      2.52 |      2.28 |      5.72 |      5.71 |      5.70 |
|  41   |      2.83 |      2.54 |      2.30 |      5.85 |      5.84 |      5.83 |
|  49   |      2.85 |      2.56 |      2.31 |      5.92 |      5.91 |      5.90 |
|  57   |      2.86 |      2.57 |      2.32 |      5.97 |      5.96 |      5.95 |
|  65   |      2.87 |      2.57 |      2.32 |      6.00 |      5.99 |      5.97 |
|  73   |      2.87 |      2.58 |      2.33 |      6.02 |      6.01 |      6.00 |
|  81   |      2.88 |      2.58 |      2.33 |      6.05 |      6.04 |      6.03 |
|  89   |      2.89 |      2.59 |      2.33 |      6.08 |      6.07 |      6.06 |
|  97   |      2.89 |      2.59 |      2.34 |      6.09 |      6.08 |      6.07 |
| 105   |      2.89 |      2.60 |      2.34 |      6.12 |      6.11 |      6.10 |
| 113   |      2.89 |      2.59 |      2.34 |      6.11 |      6.10 |      6.09 |
| 121   |      2.90 |      2.60 |      2.34 |      6.14 |      6.13 |      6.12 |
+-------+-----------+-----------+-----------+-----------+-----------+-----------+

Benchmark complete

STOP.
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.3.5 Sample SAMA5D2 setup

The following is the cryptographic setup for a SAMA5D2-based board:

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

-------------------------- END-OF-HEADER -----------------------------
File        : CRYPTO_X_Config_SAMA5D2.c
Purpose     : Configure full cryptography with hardware accelerators
              for SHA224, SHA256, and AES for SAMA5D2. Also add the
              hardware RNG for ATSAMA5Dxx devices.

*/

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

#include "CRYPTO.h"

/*********************************************************************
*
*       Preprocessor definitions
*
**********************************************************************
*/

#define CRYPTO_SAMA5D2_PMC_BASE (0xF0014000u)
#define CRYPTO_SAMA5D2_PMC_PCER0 (CRYPTO_SAMA5D2_PMC_BASE + 0x0010u)
#define CRYPTO_SAMA5D2_PMC_PCER0_FLAG_AES (1u << 9)  // PID9  = AES
#define CRYPTO_SAMA5D2_PMC_PCER0_FLAG_SHA (1u << 12) // PID12 = SHA
#define CRYPTO_SAMA5D2_PMC_PCER1 (CRYPTO_SAMA5D2_PMC_BASE + 0x0100u)
#define CRYPTO_SAMA5D2_PMC_PCER1_FLAG_TRNG (1u << (47 - 32)) // PID47 = TRNG

/*********************************************************************
*
*       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.
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pPMC_PCER0;
  volatile U32 *pPMC_PCER1;
  //
  // TRNG and hardware accelerators for AES and SHA256 are clocked
  // through the Power Management Controller (PMC), so enable and
  // configure it. Peripheral IDs are listed in Table 18.9 of the
  // datasheet ("Peripheral Identifiers").
  //
  pPMC_PCER0 = (volatile U32*)CRYPTO_SAMA5D2_PMC_PCER0;
  *pPMC_PCER0 |= CRYPTO_SAMA5D2_PMC_PCER0_FLAG_AES |
                 CRYPTO_SAMA5D2_PMC_PCER0_FLAG_SHA;
  pPMC_PCER1 = (volatile U32*)CRYPTO_SAMA5D2_PMC_PCER1;
  *pPMC_PCER1 |= CRYPTO_SAMA5D2_PMC_PCER1_FLAG_TRNG;
  //
  // Install 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_HW_SAMA5D2_SHA, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_SAMA5D2_SHA, &CRYPTO_HASH_SHA256_SW);
  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_SM4_Install      (&CRYPTO_CIPHER_SM4_SW,       NULL);
  CRYPTO_AES_Install
  (&CRYPTO_CIPHER_AES_HW_SAMA5D2_AES, &CRYPTO_CIPHER_AES_SW);
  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 Hash_DRBG-SHA-256 with TRNG entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_SAMA5D2_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 **************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.4 iMX RT10xx data coprocessor (Add-on)

The iMX RT10xx Data Coprocessor (DCP) is a programmable cryptographic 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.

6.5.3.4.1 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);
}
```

6.5.3.4.2 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.4.3 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);
  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);
  //
  // Software ciphers.
  //
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  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 ***************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.5 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.

6.5.3.5.1 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);
}
```

6.5.3.5.2 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
```

6.5.3.5.3 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.5.4 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.
  //
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  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 ****************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.6 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.

6.5.3.6.1 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);
}
```

6.5.3.6.2 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
```

6.5.3.6.3 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.7 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.

6.5.3.7.1 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);
}
```

6.5.3.7.2 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);
```

6.5.3.7.3 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 206 for further details.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.7.4 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.
  //
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  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 ****************************/
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8 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

6.5.3.8.1 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);
}
```

6.5.3.8.2 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 206 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).

6.5.3.8.3 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.1 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.2 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.3 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.4 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.5 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.6 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.7 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.8 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.9 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.3.10 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; result 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.

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.4 Performance

6.5.3.8.4.1 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_OPT_GOAL_SHA1    = SEGGER_OPT_GOAL_SPEED_MAX
Config:   CRYPTO_OPT_GOAL_SHA1_HW = SEGGER_OPT_GOAL_SPEED_MAX

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

Benchmark complete
```

 © 2015-2026 SEGGER Microcontroller GmbH

6.5.3.8.5 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.
  //
```

 © 2015-2026 SEGGER Microcontroller GmbH

```
  HFBUSCLKEN0 = (void *)0x400E40B0UL;
  *HFBUSCLKEN0 |= 1UL << 1;  // Turn on clock to CRYPTO unit
}

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

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 7

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.

 © 2015-2026 SEGGER Microcontroller GmbH

7.1 Memory footprint

emSSL is designed to cater to 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.

7.1.1 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

7.1.2 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	

 © 2015-2026 SEGGER Microcontroller GmbH

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

7.1.3 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.

7.1.3.1 Static overhead

emSSL requires approximately 0.5 KB of RAM as a fixed overhead to manage its operation.

7.1.3.2 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 on 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 the cipher suite, but typically about 500 bytes is used.

7.1.3.3 Public key algorithms

Temporary memory is required to execute appropriate public key algorithms when keys are exchanged. The amount of memory required depends on the public key algorithm and key sizes. However, typically P-256 curves and 2048-bit RSA public keys require approximately 5 KB of memory to run.

 © 2015-2026 SEGGER Microcontroller GmbH

7.1.3.4 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.

7.1.3.5 Calculating RAM requirements

The maximum amount of RAM used has to be provided in `SSL_X_Config()` while initializing emSSL and must remain available until emSSL is closed.

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
```

 © 2015-2026 SEGGER Microcontroller GmbH

7.2 Minimal system configuration

emSSL can be configured to match nearly every system requirement.

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.

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 8

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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.1 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.2 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.3 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.4 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.5 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()`.

 © 2015-2026 SEGGER Microcontroller GmbH

8.6 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 for `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 for 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 for 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 for 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 for 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.7 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.8 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.

 © 2015-2026 SEGGER Microcontroller GmbH

8.9 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.

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 9

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 do 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 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.

 © 2015-2026 SEGGER Microcontroller GmbH

9.1 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 on one cipher suite to use for the communication.

Supporting all cipher suites defined for TLS is not required, but as a client you should support the most common ones to communicate with any server.

A cipher suite is a combination of four components:

- The authentication algorithm, which authenticates the message provider (the server)
- The key exchange algorithm, used to exchange the necessary keys for the communication
- The encryption cipher, used to bulk encrypt the actual data with the derived session keys
- A message authentication code (MAC), used to authenticate the message

 © 2015-2026 SEGGER Microcontroller GmbH

9.2 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)

https://datatracker.ietf.org/doc/html/rfc2246

The Transport Layer Security (TLS) Protocol Version 1.1 (RFC 4346)

https://datatracker.ietf.org/doc/html/rfc4346

The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)

https://datatracker.ietf.org/doc/html/rfc5246

 © 2015-2026 SEGGER Microcontroller GmbH

9.3 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)

https://datatracker.ietf.org/doc/html/rfc3546

Pre-Shared Key Ciphersuites for Transport Layer Security (RFC 4279)

https://datatracker.ietf.org/doc/html/rfc4279

Transport Layer Security (TLS) Extensions: Extension Definitions (RFC 6066)

https://datatracker.ietf.org/doc/html/rfc6066

Transport Layer Security (TLS) Authorization Extensions (RFC 5878)

https://datatracker.ietf.org/doc/html/rfc5878

Transport Layer Security (TLS) Renegotiation Indication Extension (RFC 5746)

https://datatracker.ietf.org/doc/html/rfc5746

Prohibiting Secure Sockets Layer (SSL) Version 2.0 (RFC 6176)

https://datatracker.ietf.org/doc/html/rfc6176

TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks

https://datatracker.ietf.org/doc/html/draft-ietf-tls-downgrade-scsv-00

 © 2015-2026 SEGGER Microcontroller GmbH

9.4 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)

https://datatracker.ietf.org/doc/html/rfc6655

AES Galois Counter Mode (GCM) Cipher Suites for TLS (RFC 5288)

https://datatracker.ietf.org/doc/html/rfc5288

Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (RFC 4492)

https://datatracker.ietf.org/doc/html/rfc4492

Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (RFC 7027)

https://datatracker.ietf.org/doc/html/rfc7027

Elliptic Curve Cryptography Subject Public Key Information (RFC 5480)

https://datatracker.ietf.org/doc/html/rfc5480

Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality (SP 800-38C)

This is a NIST specification, not 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

 © 2015-2026 SEGGER Microcontroller GmbH

9.5 Hardware acceleration

9.5.1 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)

9.5.2 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.

9.5.3 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.

9.5.4 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.

9.5.5 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.

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 10

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 to the end-user equipment to restrict the export of strong cryptography. emSSL uses strong cryptography for signing and encryption. Since export controls depend on what you are exporting, the purpose of the cryptographic device, and the type of cryptography your device uses, it is impossible to provide statements that cover all situations and devices.

> **Warning**
>
> We strongly advise you to conduct your own research and consult legal advisors regarding the deployment of SSL (and ECC cipher suites in particular) in your devices.
>
> The 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.

 © 2015-2026 SEGGER Microcontroller GmbH

10.1 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.

 © 2015-2026 SEGGER Microcontroller GmbH

10.2 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 the United States of America and appropriate patents were made available worldwide on a royalty-free basis.

 © 2015-2026 SEGGER Microcontroller GmbH

10.3 Using ECC cipher suites

ECC is the most problematic public key cryptosystem because of active patents assigned to Certicom (at the time of writing, November 19, 2026). 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 any *implementation* patent.

Any contributor to an IETF standard document must 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

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 uses algorithms from antiquity.

Warning

We stress again that patents are issued covering both different geographic domains and fields of use. You are responsible for ensuring that your end products (which contain emSSL) do not infringe patents in the locations where you produce and sell that equipment.

The 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 and to seek independent legal advice.

 © 2015-2026 SEGGER Microcontroller GmbH

10.4 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.

Warning

We stress again that it is impossible to provide any warranties for 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.

The 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.

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 11

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 produces the same sequence of bits given the same seed as input, but the output passes 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).

 © 2015-2026 SEGGER Microcontroller GmbH

DTLS
Datagram Transport Layer Security. A protocol 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. Like ECDH but using ephemeral keys, providing perfect forward secrecy (PFS).

ECDSA
Elliptic Curve Digital Signature Algorithm. A standard for digital signatures using elliptic curve cryptography instead of discrete logarithm cryptography. It is the elliptic curve analog of the DSA signature scheme.

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
Hash-based Message Authentication Code. A MAC computed using a cryptographic hash function and 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 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. A U.S. organization responsible for standardizing a wide range of technologies used in 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.

 © 2015-2026 SEGGER Microcontroller GmbH

PSK
Preshared Key. A shared private key held by two entities agreed in advance of communication.

RFC
Request for Comments. The standard document series used by the IETF to publish Internet standards and specifications.

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 that provides confidential and authenticated data transmission over insecure channels.

 © 2015-2026 SEGGER Microcontroller GmbH

Chapter 12

Indexes

 © 2015-2026 SEGGER Microcontroller GmbH

12.1 Function index

CRYPTO_AES_Install, 196, 215, 231, 231, 232, 233, 235, 237, 237, 241, 242, 243, 245, 247, 248, 248, 251, 264
CRYPTO_ARIA_Install, 196, 218, 236, 241, 244, 247, 251
CRYPTO_CAMELLIA_Install, 196, 219, 236, 241, 244, 247, 251
CRYPTO_MD5_Install, 196, 214, 232, 233, 235, 241, 244, 247, 249, 251, 264
CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO, **253**
CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO, **258**
CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO, **254**
CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO, **259**
CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO, **255**
CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO, **260**
CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO, **256**
CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO, **261**
CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO, **257**
CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO, **262**
CRYPTO_OS_Claim, **208**
CRYPTO_OS_Init, **207**
CRYPTO_OS_Request, **209**
CRYPTO_OS_Unclaim, **210**
CRYPTO_SEED_Install, 196, 217, 236, 241, 244, 247, 251
CRYPTO_SHA1_Install, 196, 211, 232, 233, 235, 241, 242, 243, 244, 247, 249, 251, 252, 264
CRYPTO_SHA256_Install, 196, 212, 232, 233, 235, 237, 241, 242, 243, 244, 247, 249, 251, 264
CRYPTO_SHA512_Install, 196, 213, 232, 236, 241, 244, 247, 251, 264
CRYPTO_TDES_Install, 196, 216, 232, 233, 235, 241, 243, 245, 247, 251, 264
SSL_AddLogFilter, **158**, 158
SSL_AddWarnFilter, **159**, 159
SSL_CIPHER_Add, **98**, 174
SSL_CLIENT_ConfigMutualAuth, **99**
SSL_CURVE_Add, **100**, 177, 177, 177, 177, 177
SSL_CURVE_GetName, **102**
SSL_CURVE_Remove, **101**
SSL_ERROR_GetText, 35, 35, 39, 39, 43, 43, 45, 45, **160**
SSL_Exit, 42, 46, **95**
SSL_GetCopyrightText, 40, 45, **93**
SSL_GetVersionText, **92**
SSL_Init, 33, 40, 41, 45, **96**
SSL_MAC_Add, **103**, 175, 176
SSL_MEM_Add, **104**
SSL_MEM_ConfigSystem, **105**
SSL_MEM_GetContext, **106**
SSL_OS_DisableInterrupt, 184, 187
SSL_OS_EnableInterrupt, 185, 187
SSL_OS_GetTaskName, 185, 188
SSL_OS_GetTime32, **183**, 185, 188
SSL_OS_Init, **180**, 184, 187
SSL_OS_Lock, **181**, 185, 188
SSL_OS_Unlock, **182**, 185, 188
SSL_PROTOCOL_Add, **107**, 178, 178, 178, 178, 178, 178, 178
SSL_PROTOCOL_GetText, **108**
SSL_RemoveLogFilter, **161**, 161
SSL_RemoveWarnFilter, **162**
SSL_ROOT_CERTIFICATE_Add, 55, **109**
SSL_ROOT_CERTIFICATE_AddDER, **110**
SSL_ROOT_CERTIFICATE_LoadDER, **111**
SSL_SERVER_ConfigMutualAuth, **112**
SSL_SESSION_Accept, 34, 40, **121**
SSL_SESSION_ClrFlags, **122**
SSL_SESSION_Connect, 42, 46, **123**, 158, 159, 161
SSL_SESSION_Disconnect, 34, 40, 42, 46, **124**
SSL_SESSION_GetRecordSizeLimit, **125**
SSL_SESSION_GetSessionID, 66, **126**
SSL_SESSION_GetSocket, 66, 67, **127**
SSL_SESSION_GetSuite, 54, 54, **128**
SSL_SESSION_Prepare, 34, 40, 41, 46, **129**, 139
SSL_SESSION_Process, **130**
SSL_SESSION_QueryFlags, **132**
SSL_SESSION_Receive, 36, 38, 44, **131**
SSL_SESSION_Send, 35, 39, **133**
SSL_SESSION_SendStr, 43, 45, **134**
SSL_SESSION_SetAllowedSuites, **135**
SSL_SESSION_SetCertificateAPI, 53, **136**
SSL_SESSION_SetFlags, **137**
SSL_SESSION_SetProtocolRange, **139**, 139
SSL_SESSION_SetPSK, **140**

 © 2015-2026 SEGGER Microcontroller GmbH

SSL_SESSION_SetRecordSizeLimit, **141**
SSL_SetDefaultCertificateAPI, **113**
SSL_SetLogFilter, 158, 161, **163**, 163
SSL_SetWarnFilter, 159, **164**, 164
SSL_SIGNATURE_ALGORITHM_Add, **114**, 176
SSL_SIGNATURE_SIGN_Add, **115**, 176
SSL_SIGNATURE_VERIFY_Add, **116**, 175, 177
SSL_SUITE_Add, **117**, 171, 178, 178, 178, 178
SSL_SUITE_CopyName, **143**
SSL_SUITE_FindByID, **144**
SSL_SUITE_FindByIndex, **145**
SSL_SUITE_GetCipherName, **146**
SSL_SUITE_GetIanaName, **147**
SSL_SUITE_GetID, **148**
SSL_SUITE_GetKeyExchangeName, **149**
SSL_SUITE_GetMACAlgorithmName, **150**
SSL_SUITE_GetPKAlgorithmName, **151**
SSL_SUITE_QueryNull, **152**
SSL_SUITE_QueryPKAlgorithm, **153**
SSL_SUITE_QueryRequiresECC, 54, 54, **154**
SSL_SUITE_QueryRequiresPSK, **155**
SSL_SUITE_QueryValidity, **156**

 © 2015-2026 SEGGER Microcontroller GmbH

12.2 Subject index

AES,
  configuration,
    compile-time, 224
    runtime, 215
  memory footprint, 224
ARIA,
  configuration,
    compile-time, 227
    runtime, 218
  memory footprint, 227

Camellia,
  configuration,
    compile-time, 228
    runtime, 219
  memory footprint, 228
Cipher,
  adding to emSSL, 174
  AES, 215, 224
  ARIA, 218, 227
  Camellia, 219, 228
  DES, 216, 225
  SEED, 217, 226
Cipher API,
  implementation,
    CRYPTO_CIPHER_AES_HW_Kinetis_CAU, 233
    CRYPTO_CIPHER_AES_HW_LPC_ROM, 231
    CRYPTO_CIPHER_AES_HW_RT10xx_DCP, 242
    CRYPTO_CIPHER_AES_HW_SAMA5D2_AES, 237
    CRYPTO_CIPHER_AES_HW_STM32_AES, 248
    CRYPTO_CIPHER_AES_HW_STM32_CRYP, 245
    CRYPTO_CIPHER_AES_SW, 215
    CRYPTO_CIPHER_ARIA_SW, 218
    CRYPTO_CIPHER_CAMELLIA_SW, 219
    CRYPTO_CIPHER_SEED_SW, 217
    CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, 233
    CRYPTO_CIPHER_TDES_HW_STM32_CRYP, 245
    CRYPTO_CIPHER_TDES_SW, 216
Cipher suite,
  adding to emSSL, 171
  functions,
    SSL_SUITE_Add(), 117
    SSL_SUITE_CopyName(), 143
    SSL_SUITE_FindByID(), 144
    SSL_SUITE_FindByIndex(), 145
    SSL_SUITE_GetCipherName(), 146
    SSL_SUITE_GetIanaName(), 147
    SSL_SUITE_GetID(), 148
    SSL_SUITE_GetKeyExchangeName(), 149
    SSL_SUITE_GetMACAlgorithmName(), 150
    SSL_SUITE_GetPKAlgorithmName(), 151
    SSL_SUITE_QueryNull(), 152
    SSL_SUITE_QueryPKAlgorithm(), 153
    SSL_SUITE_QueryRequiresECC(), 154
    SSL_SUITE_QueryRequiresPSK(), 155
    SSL_SUITE_QueryValidity(), 156
  preference order, 174
Configuration,
  SSL_X_Config(), 171

DES,
  configuration,
    compile-time, 225
    runtime, 216
  memory footprint, 225

ECDSA,
  configuration,
    compile-time, 229
  memory footprint, 229
Elliptic curve,
  adding to emSSL, 177

Hash algorithm,
  MD5, 214, 223
  SHA-1, 211, 220
  SHA-256, 212, 221
  SHA-512, 213, 222
Hash API,
  implementation,
    CRYPTO_HASH_MD5_HW_Kinetis_CAU, 233
    CRYPTO_HASH_MD5_HW_STM32_HASH, 249
    CRYPTO_HASH_MD5_SW, 214
    CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, 252
    CRYPTO_HASH_SHA1_HW_Kinetis_CAU, 233
    CRYPTO_HASH_SHA1_HW_RT10xx_DCP, 242
    CRYPTO_HASH_SHA1_HW_STM32_HASH, 249
    CRYPTO_HASH_SHA1_SW, 211
    CRYPTO_HASH_SHA224_HW_Kinetis_CAU, 233
    CRYPTO_HASH_SHA224_HW_SAMA5D2_SHA, 237
    CRYPTO_HASH_SHA224_HW_STM32_HASH, 249
    CRYPTO_HASH_SHA256_HW_Kinetis_CAU, 233
    CRYPTO_HASH_SHA256_HW_RT10xx_DCP, 242
    CRYPTO_HASH_SHA256_HW_SAMA5D2_SHA, 237
    CRYPTO_HASH_SHA256_HW_STM32_HASH, 249
    CRYPTO_HASH_SHA256_SW, 212
    CRYPTO_HASH_SHA512_SW, 213

MAC,
  adding to emSSL, 175
MD5,
  configuration,
    compile-time, 223
    runtime, 214
  memory footprint, 223
Message signers,
  adding to emSSL, 175
Message signing,
  functions,
    SSL_SIGNATURE_SIGN_Add(), 115

PrintCert utility, 55

SEED,
  configuration,
    compile-time, 226
    runtime, 217
  memory footprint, 226
SHA-1,
  configuration,
    compile-time, 220
    runtime, 211
  memory footprint, 220
SHA-256,
  configuration,
    compile-time, 221
    runtime, 212
  memory footprint, 221
SHA-512,
  configuration,
    compile-time, 222
    runtime, 213
  memory footprint, 222
Signature algorithm,
  adding to emSSL, 176
Signature verifier,
  adding to emSSL, 175
  functions,
    SSL_SIGNATURE_VERIFY_Add(), 116

TLS protocol,
  adding to emSSL, 177

 © 2015-2026 SEGGER Microcontroller GmbH