

smxSSL™

Strong security protection for networked embedded devices

Secure Socket Layer (SSL) provides seamless data security services that offer protection for TCP/IP protocols such as HTTP and SMTP and that can also be used to protect proprietary socket applications.

smxSSL is a small footprint embedded SSL implementation that provides client- and server-based secure communications for network sessions. smxSSL is integrated with the SMX RTOS and the smxNS TCP/IP stack to simplify application development. smxSSL provides C APIs for integration of either client or server SSL components into an embedded application. These highly efficient, robust, and small footprint implementations offer the core services required to protect critical computing resources across open communication networks while avoiding the substantial overhead of enterprise SSL (e.g. Open SSL) solutions. The protocol implementation is lightweight and fast and can be migrated to other operating systems and networking stacks with ease. The publicly available cryptographic libraries shipped with smxSSL have been used in thousands of embedded systems and provide secure and robust protection for sensitive data and control information.

smxSSL provides core SSLv3 Server and Client capabilities in a small, efficient implementation that targets the most common embedded SSL needs without the unnecessary overhead of features more suitable for enterprise applications. This product is sold in two forms – smxSSLs, which contains the base SSL library and the SSL server-specific APIs, and smxSSLC, which contains the base SSL library plus the SSL client-specific APIs.

Features

- Open standards based implementation.
- Independent SSL client and server components with simple C APIs.
- Lightweight, efficient implementations targeted for embedded systems use.
- Small code size (under 50 KB) per component.
- Proven public cryptographic libraries.
- smxSSL server integrated with smxNS web server to provide HTTPS.
- smxSSL client integrated with smxHTTP lite client to provide secure downloading from a central web server.
- Transport agnostic: integrated with smxNS but extensible to other network stacks.
- RTOS agnostic: integrated with SMX RTOS, but includes multithreading support for optimal performance with any RTOS.
- Processor agnostic: will run on any moderate to high performance processor.

smxSSL provides:

- A modular, easily extensible, configurable cryptographic library.
- Session resumption with configurable session cache.
- Multiple SSL servers.
- Key length configuration to optimize library size.
- Simple, straightforward application programming interface.

These are explained below.

Cryptographic Library

smxSSL supports a modular and easily extensible cryptographic library that, by default, incorporates the following cryptographic algorithms:

- RSA Key Exchange
- AES symmetric cipher (128-bit keys)
- 3DES symmetric cipher (168-bit keys)
- ARC4 symmetric cipher (128, 56, and 40-bit keys)
- SHA-1 hash
- MD5 hash

These algorithms are supported in the following combinations (called *cipher suites*):

- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_EXPORT1024_WITH_RC4_56_MD5
- SSL_RSA_EXPORT1024_WITH_RC4_56_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5

smxSSL provides configuration macros which permit disabling the AES and 3DES cipher algorithms when they are not in use, thus reducing memory requirements.

Additional cryptographic algorithms can be added without making changes to the core SSL implementation, by introducing the modules for them into the library and extending the tables that define available functions.

Session Resumption (smxSSLs only)

smxSSLs supports session resumption, which allows frequently connecting clients to submit their previously used SSL Session IDs. When smxSSLs finds a Session ID in the session cache, the computationally-intensive operation related to RSA Key Exchange is bypassed,

leading to quicker connection times for frequent clients (web browsers using HTTP, for example).

The session cache size can be adjusted to fine tune library memory use vs. system performance.

Multiple SSL Servers

smxSSLs is designed to protect multiple resources within a single application. It does this by implementing multiple smxSSLs servers, each with its own cache. Each smxSSLs server manages a unique certificate (including public/private key pair) allowing multiple clients to authenticate individual services from a single application. In order to maintain control over memory use, server cache sizes can be adjusted.

Configurable Key Length

Longer key lengths effectively strengthen cryptographic algorithms, but they do so at the expense of increased memory and processing time. Each time a client connects to a server, the client and the server exchange secure messages to establish a secret key. Once the key exchange completes, symmetric cipher algorithms encrypt and decrypt data using key lengths as noted in the supported list of algorithms. However, as previously noted, the length of the key significantly effects memory requirements for system caches and dynamic memory allocation, and also significantly effects computation time for cryptographic functions. For this reason, the largest expected key length can be specified, allowing the system to balance memory use and performance vs. an acceptable level of security.

Simple API

smxSSL provides a simple, straightforward API for configuring and managing smxSSL servers and connections. The API works with any reliable transport provided by the calling application to perform data transfer. The API

also provides functions for library management and private key translation.

Server Configuration Functions

ltrxSSLSystemConfig(time, seed)
ltrxSSLServerConfig(handle, certificate,
certificate_size, key, key_size)
ltrxSSLServerConfigRelease(handle)

Connection Management Functions

ltrxSSLAccept(handle, configuration)
ltrxSSLClose(handle)
ltrxSSLConnect(handle, callback, ciphers,
num_ciphers)

ltrxSSLConnectionGet(handle, socket,
transport, timeout, buffer, size)
ltrxSSLReconnect(handle, callback, id, ciphers,
num_ciphers)

Data Transfer Functions

ltrxSSLSend(connection, buffer, size)
ltrxSSLBufferedSend(connection, buffer, size)
ltrxSSLRecv(connection, buffer, size)

Code Footprint and Memory Requirements

smxSSL was designed to be small and efficient with configurable options to optimize memory requirements with regard to features and performance.

smxSSLs Memory Footprints

The code footprint varies depending on the use of the AES and 3DES ciphers, and also the maximum key length the library supports as follows:

Component	ARM7TDMI*	ColdFire v2**
Core	23528	24682
Debug Code	1048	1132
AES Cipher	4520	5334
3DES Cipher	2492	2576
Total for 512-bit Library	31588	33724
For 1024-bit library, add	64	42
For 2048-bit library, add	296	128
For 4096-bit library, add	552	128

* With IAR ARM v5.20 (ARM Mode)

** With CodeWarrior 7.1

Memory requirements depend upon configurations, as follows:

Component	R.O. Data	R/W Data
smxSSLs Core	636	2688
Handshake Buffer (app supplied)		858
512-bit RSA Private Key (app supplied)	330	
Sample 512-bit Certificate (app supplied)	848	
Config when using AES and 3DES	64	8
Additional Server Cache*		(1152 * v)
Additional Connection Cache*		(1196 * c)
Additional Session Cache*		(108 * s)
AES Cipher	10304	
3DES Cipher	688	4892
Debug Data	888	132
Block Cipher (with AES, 3DES, or Debug)		2048
Default Total**	13758	14538

* The library requires one instance each of the Server, Connection, and Session cache objects, which are included in the smxSSLs Core number.

** Total is for v = 0, c = 3, s = 3, which are the defaults.

In the minimum configuration, a 512-bit library does not include additional Server, Connection, or Session cache objects nor the AES and 3DES Ciphers. When disabling debugging, the library saves an additional 2K of memory because a buffer shared by the AES and 3DES ciphers with debugging is no longer required:

Component	R.O. Data	R/W Data
smxSSLs Core	636	2688
Handshake Buffer (app supplied)		858
512-bit RSA Private Key (app supplied)	330	
Sample 512-bit Certificate (app supplied)	848	
Minimum Total	1814	3546

As the maximum key length grows, the memory requirements increase for certain items, as shown in the following table. Storage for the RSA private key and certificate must grow to include the longer bit length and the application-supplied SSL handshake buffer must also grow so it is large enough to handle the SSL handshake. Items not shown in this list are not affected by maximum key length:

Component	512-bit		1024-bit		2048-bit		4096-bit	
	R.O.	R/W	R.O.	R/W	R.O.	R/W	R.O.	R/W
smxSSLs Core	636	2688	764	3712	1020	5760	1532	9856
Handshake Buffer		858		966		1217		1739
RSA Private Key		330		312		1400		2800
Sample Certificate	848		956		1227		1729	
Each Server Cache		1152		2176		4224		8320

The stack space and heap requirements are also related to key size. These figures represent maximum measured additional stack and heap requirements for the smxSSLs HTTPS and echo server demonstrations. These figures do not include the total used by SMX and smxNS.

Max Key Size	HTTPS Max Stack		Echo Server Max Stack		Max Heap Use
	ARM7	ColdFire v2	ARM7	ColdFire v2	
512-bit	1.4	1.7	1.2	1.9	5.3 KB
1024-bit	3.5	4.3	4416	4.2	11.3 KB
2048-bit	8.0	9.0	8.1	9.0	25.3 KB
4096-bit	17.1	18.6	17.1	18.5	51.3 KB

smxSSLc Memory Footprints

The SSL client demo in ssldemo.c has the following memory requirements when compiled for an ARM9 processor using IAR Workbench 5.2. These values are the incremental additional memory required when SSL client support is added to a base network build. Note that in this build, the task stack is allocated out of the heap, so R.W. Data plus Heap yields the incremental RAM requirement.

Max Key Size	R.O. Code	R.O. Data	R/W Data	Max Stack	Heap
512	40288	15428	14854	3064	4048
1024	40316	15556	17158	4352	5552
2048	40504	15812	21766	6888	8048
4096	40764	16324	30982	12016	13056