Security Library for Open AT Application Framework

Development Guide

SIERRA WIRELESS

4111868
6.0
June 14, 2013

Important Notice

Due to the nature of wireless communications, transmission and reception of data can never be guaranteed. Data may be delayed, corrupted (i.e., have errors) or be totally lost. Although significant delays or losses of data are rare when wireless devices such as the Sierra Wireless modem are used in a normal manner with a well-constructed network, the Sierra Wireless modem should not be used in situations where failure to transmit or receive data could result in damage of any kind to the user or any other party, including but not limited to personal injury, death, or loss of property. Sierra Wireless accepts no responsibility for damages of any kind resulting from delays or errors in data transmitted or received using the Sierra Wireless modem, or for failure of the Sierra Wireless modem to transmit or receive such data.

Safety and Hazards

Do not operate the Sierra Wireless modem in areas where cellular modems are not advised without proper device certifications. These areas include environments where cellular radio can interfere such as explosive atmospheres, medical equipment, or any other equipment which may be susceptible to any form of radio interference. The Sierra Wireless modem can transmit signals that could interfere with this equipment. Do not operate the Sierra Wireless modem in any aircraft, whether the aircraft is on the ground or in flight. In aircraft, the Sierra Wireless modem **MUST BE POWERED OFF**. When operating, the Sierra Wireless modem can transmit signals that could interfere with various onboard systems.

Note: *Some airlines may permit the use of cellular phones while the aircraft is on the ground and the door is open. Sierra Wireless modems may be used at this time.*

The driver or operator of any vehicle should not operate the Sierra Wireless modem while in control of a vehicle. Doing so will detract from the driver or operator's control and operation of that vehicle. In some states and provinces, operating such communications devices while in control of a vehicle is an offence.

Limitations of Liability

This manual is provided "as is". Sierra Wireless makes no warranties of any kind, either expressed or implied, including any implied warranties of merchantability, fitness for a particular purpose, or noninfringement. The recipient of the manual shall endorse all risks arising from its use.

The information in this manual is subject to change without notice and does not represent a commitment on the part of Sierra Wireless. SIERRA WIRELESS AND ITS AFFILIATES SPECIFICALLY DISCLAIM LIABILITY FOR ANY AND ALL DIRECT, INDIRECT, SPECIAL, GENERAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS OR REVENUE OR ANTICIPATED PROFITS OR REVENUE ARISING OUT OF THE USE OR INABILITY TO USE ANY SIERRA WIRELESS PRODUCT, EVEN IF SIERRA WIRELESS AND/OR ITS AFFILIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR THEY ARE FORESEEABLE OR FOR CLAIMS BY ANY THIRD PARTY.

Notwithstanding the foregoing, in no event shall Sierra Wireless and/or its affiliates aggregate liability arising under or in connection with the Sierra Wireless product, regardless of the number of events, occurrences, or claims giving rise to liability, be in excess of the price paid by the purchaser for the Sierra Wireless product.

Customer understands that Sierra Wireless is not providing cellular or GPS (including A-GPS) services. These services are provided by a third party and should be purchased directly by the Customer.

SPECIFIC DISCLAIMERS OF LIABILITY: CUSTOMER RECOGNIZES AND ACKNOWLEDGES SIERRA WIRELESS IS NOT RESPONSIBLE FOR AND SHALL NOT BE HELD LIABLE FOR ANY DEFECT OR DEFICIENCY OF ANY KIND OF CELLULAR OR GPS (INCLUDING A-GPS) SERVICES.

Patents

This product may contain technology developed by or for Sierra Wireless Inc.

This product includes technology licensed from QUALCOMM®.

This product is manufactured or sold by Sierra Wireless Inc. or its affiliates under one or more patents licensed from InterDigital Group and MMP Portfolio Licensing.

Copyright

Trademarks

Sierra Wireless®, AirPrime®, AirLink®, AirVantage® and the Sierra Wireless logo are registered trademarks of Sierra Wireless.

Watcher® is a registered trademark of Netgear, Inc., used under license.

Windows® and Windows Vista® are registered trademarks of Microsoft Corporation.

Macintosh® and Mac OS X® are registered trademarks of Apple Inc., registered in the U.S. and other countries.

QUALCOMM® is a registered trademark of QUALCOMM Incorporated. Used under license.

Other trademarks are the property of their respective owners.

Contact Information

Sales Desk:	Phone:	1-604-232-1488
	Hours:	8:00 AM to 5:00 PM Pacific Time
	E-mail:	sales@sierrawireless.com
Post:	Sierra Wireless 13811 Wireless Way Richmond, BC Canada V6V 3A4	
Technical Support:	support@sierrawireless.com	
RMA Support:	repairs@sierrawireless.com	
Fax:	1-604-231-1109	
Web:	www.sierrawireless.com	

Consult our website for up-to-date product descriptions, documentation, application notes, firmware upgrades, troubleshooting tips, and press releases: www.sierrawireless.com

Document History

Version	Date	Updates
1	May 6, 2008	Creation
2	February 22, 2010	Updated Reference Document section. Updated document to Sierra Wireless format.
3	July 20, 2010	Internal document updates (indexing)
4	January 7, 2012	Updated the Options accepted by getOpts section. Updated product naming conventions.
5.0	May 25, 2012	Updated legal boilerplate. Added WIP_COPT_SEED to the Function wip_SSLInitOpts and Configuration Examples sections. Removed inaccurately documented WIP_SSL_NULL option for WIP_COPT_ENCRYPTION and WIP_COPT_AUTHENTICATION in the Function wip_SSLInitOpts section.
6.0	June 14, 2013	WIPSSL 2.0 interface update. Major update of interface based on the update of the underlying OpenSSL Library. Added wip_SSLGetVersion(). Added Session Encoding and decoding functions: wipssl_EncodeSession and wipssl_DecodeSession. Added new possible error codes to wip_SSLInitOpts(). Added Automatic Session Handling support. Clarified the generic wip channel operations on SSL channel. Clarified that session handling supports both Session ID and Session Ticket. Removed some obsolete, insecure options and added other, more secure algorithms.

Contents

CONTENTS 5

1. INTRODUCTION 8
   1.1. Overview 8
   1.2. Related Documents 8
   1.3. Abbreviations and Glossary 8
2. PRINCIPLES 10
   2.1. Internet Library SSL API 10
   2.2. Supported Features 10
   2.3. Key, Certificates, Trusted Certificate Authorities 11
3. CONFIGURATION AND THE LIBRARY API 12
   3.1. Introduction 12
   3.2. SSL Library Initialization 12
      3.2.1. Function wip_SSLInit() 12
      3.2.2. Prototype 12
      3.2.3. Parameters 12
      3.2.4. Returned Values 12
   3.3. SSL Library Configuration 13
      3.3.1. Function wip_SSLInitOpts() 13
      3.3.2. Prototype 13
      3.3.3. Parameters 13
      3.3.4. Configuration Examples 14
      3.3.5. Returned Values 15
   3.4. SSL Library Termination 15
      3.4.1. Function wip_SSLClose() 15
      3.4.2. Prototype 15
      3.4.3. Parameters 15
      3.4.4. Returned Values 15
   3.5. Library Versions 16
      3.5.1. Function wip_SSLGetVersion() 16
      3.5.2. Prototype 16
      3.5.3. Parameters 16
      3.5.4. Example 16
      3.5.5. Returned Values 16
4. CONSTRUCTOR AND CHANNEL MANIPULATION 17
   4.1. Introduction 17
   4.2. Channels Creation 17
      4.2.1. Function wip_SSLClientCreate() 17
         4.2.1.1. Prototype 17
         4.2.1.2. Parameters 17
         4.2.1.3. Returned Values 17

4.2.2. Function wip_SSLClientCreateOpts() 18
4.2.2.1. Prototype 18
4.2.2.2. Parameters 18
4.2.2.3. Returned Values 18
4.3. SSL Channels API 18
4.3.1. The wip_read() function 19
4.3.1.1. Prototype 19
4.3.1.2. Parameters 19
4.3.1.3. Returned Values 19
4.3.2. The wip_readOpts() function 19
4.3.3. The wip_write() function 20
4.3.3.1. Prototype 20
4.3.3.2. Parameters 20
4.3.3.3. Returned Values 20
4.3.4. The wip_writeOpts() function 20
4.3.5. The wip_getOpts () function 20
4.3.5.1. Prototype 21
4.3.5.2. Parameters 21
4.3.5.3. Returned Values 24
4.3.5.4. Example 24
4.3.6. The wip_setOpts() function 24
4.3.6.1. Prototype 24
4.3.6.2. Parameters 24
4.3.6.3. Returned Values 24
4.3.6.4. Example 25
4.3.7. The wip_close() function 25
4.3.7.1. Prototype 25
4.3.7.2. Parameters 25
4.3.7.3. Returned Values 25
4.3.7.4. Example 25
4.3.8. The wip_abort() function 25
4.3.9. The wip_shutdown() function 26
5. SESSIONS 27
5.1. Introduction 27
5.2. Session Handling 27
5.3. Automatic Session Handling 27
5.4. Application Handled Sessions 28
5.4.1. wip_SSLSession_t wip_SSLGetSession(wip_channel_t ssl_channel) 28
5.4.2. wip_SSLSession_t wip_SSLGetSessionFromHTTPSDataChannel (wip_channel_t http_DataChannel) 28
5.4.3. void wip_SSLReleaseSession(wip_SSLSession_t ssl_session) 28
5.5. Saving Sessions across Reboots 29
5.5.1. wipssl_EncodeSession 29
5.5.1.1. Prototype 29
5.5.1.2. Parameters 29
5.5.1.3. Returned Values 29
5.5.2. wipssl_DecodeSession 29
5.5.2.1. Prototype 29
5.5.2.2. Parameters 29
5.5.2.3. Returned Values 30
6. INTERNET LIBRARY SSL USAGE 31
6.1. Typical Call Sequence 31
6.2. Sample 31

7. TROUBLESHOOTING 35
7.1. Internet Library Trace Service 35
7.2. Low Security Errors 35
APPENDIX A. INTERFACE UPDATE 36
A.1 Enums Removed 36
A.2 Enums Renaming 36
A.3 Default Values 37
A.4 Changed Accepted Parameters wip_setOpts() 37
INDEX 38

1. Introduction

1.1. Overview

This document describes Internet Library SSL API, which provides SSL-secured communication channels to the Internet Library. This document assumes the reader is already familiar with Internet Library. Please refer to the following section for a list of related documents.

1.2. Related Documents

Please refer to the following documentation for more information regarding the Security Library and peripherals.

Document	Reference
[1] Internet Library Connectivity Development Guide	4111845
[2] Internet Application AT Commands User Guide	4111846
[3] ADL User Guide for Open AT Framework OS	4111844
[4] Security AT Commands for Open AT Framework – Interface Guide	4112704

Note: Internet Library SSL is based upon OpenSSL, and a clear idea of how OpenSSL works is recommended. To this end, learning more about the general workings of OpenSSL or more via OpenSSL-dedicated documentation is recommended.

1.3. Abbreviations and Glossary

Abbreviations	Definition
3DES	Triple DES.
AES	Advanced Encryption Standard.
API	Application Programming Interface.
DES	Data Encryption Standard.
DSA	Digital Signature Algorithm.
DSS	Digital Signature Standard.
HTTPS	HTTP over SSL.
MAC	Message Authentication Code.
MD5	Message Digest version 5.
OpenSSL	A free SSL toolkit. Please refer to http://www.openssl.org/ for more information.
RSA	The RSA algorithm. R, S, A are the first letter of the surnames of the three creators.
SHA1	Secure Hash Algorithm, version 1.
SHA2	Secure Hash Algorithm, version 2, is a collective name for SHA256, SHA384, and SHA512.

Abbreviations	Definition
SHA256	Secure Hash Algorithm, version 2, with digest length of 256 bits.
SSL	Secure Socket Layer. See TLS.
TLS	Transport Layer Security. The successor to SSL version. Sometimes refered to as SSL when talking generally about TLS/SSL.

2. Principles

2.1. Internet Library SSL API

Internet Library SSL is based on OpenSSL. It provides Internet Library channels, with a regular Internet Library channel API, which communicate through SSL. Because they keep Internet Library API, the only differences between using a regular TCP channel and using an Internet Library SSL channel are as follows:

- Internet Library SSL channels are created by calling `wip_SSLClientCreate()` instead of `wip_TCPClientCreate()`.
- Before using these constructors, the SSL library must be configured by a call to `wip_SSLInit()` or `wip_SSLInitOpts()`, describe authorized protocols (keys, authentication, encryption, MAC, SSL version, private keys, certificates, trusted certification authorities)

Once the channel constructor has been called, SSL channels behave just as their regular TCP counterparts:

- Client channels emit `WIP_CEV_OPEN` once they are ready to communicate. They signal data arrival after data starving with `WIP_CEV_READ`, write buffer liberation after it had been filled with `WIP_CEV_WRITE`, peer shutdown with `WIP_CEV_PEER_CLOSE`, and problems with `WIP_CEV_ERROR`. They support the usual `wip_read()`, `wip_readOpts()`, `wip_write()`, `wip_writeOpts()`, `wip_shutdown()`, `wip_abort()`, `wip_close()` APIs. They also support `wip_setOpts()` and `wip_getOpts()`, although they don't accept exactly the same set of options as TCP clients.

2.2. Supported Features

Internet Library SSL supports the following features.

- Cipher keys:
  - RSA key exchange
  - Diffie-Hellman key agreement, with ephemeral (temporary) keys
- Authentication schemes:
  - RSA
  - DSS
  - NULL (no authentication)
- Encryption algorithms:
  - DES
  - triple DES
  - RC2
  - RC4
  - AES
  - NULL (no encryption)
- MAC message authentication:
  - MD5
  - SHA1
  - SHA2 (256)
- SSL versions 3 and TLS 1.0, TLS1.1, TLS1.2

- Client mode

This configuration is application-wide, i.e. at a given instant, all SSL clients accept the same set of encryption methods, authentication schemes etc.

The number of client keys, certificates etc. is only limited by the amount of RAM available.

Please note that all combinations are not valid, example some algorithms are is not supported by TLS1.2.

2.3. Key, Certificates, Trusted Certificate Authorities

Because Security Library does not necessarily rely on a file system, these data are to be stored directly in memory (flash or RAM) as '\0'-terminated strings. It is the application's responsibility to provide them, as C constants, flash objects, or downloaded data.

With the new arrival of File system support in Open AT, it is possible to store the certificates as files, but the data must be read into a C variable and it is up to the applications to manage the files.

Warning: *Though there is no foolproof way to secure private keys security, you are strongly advised to find and determine ideal methods of customizing your keys in order to obfuscate them. Doing so and thereby making your key more obscure and randomized will greatly increase security levels.*

3. Configuration and the Library API

3.1. Introduction

Configuring Internet Library SSL only has to be done when first working with and possibly updating the API; it does not need to be configured on a regular basis.

3.2. SSL Library Initialization

3.2.1. Function wip_SSLInit()

Initializes the SSL library with default settings if not stated otherwise below:

- No private key, no certificate, no trusted certificate authority. Please note that there is no check of the servers certificate, since there is no CA.
- All key exchange algorithms accepted
- All authentication schemes accepted, except NULL
- All encryption schemes accepted, except NULL
- All SSL/TLS versions accepted

3.2.2. Prototype

```
int wip_SSLInit( void );
```

3.2.3. Parameters

None.

3.2.4. Returned Values

This function returns

- 0 if the SSL Library has been successfully initialize
- In case of an error, the function returns a negative error code that corresponds to an error value in wip_error_t or wipssl_error_t.

3.3. SSL Library Configuration

3.3.1. Function wip_SSLInitOpts()

The `wip_SSLInitOpts()` function is used to initialize the SSL library with the options given by the user. It works like wip_SSLInit(), but gives the possibility to change options.

3.3.2. Prototype

```
int wip_SSLInitOpts( int optid, ..., WIP_COPT_END);
```

3.3.3. Parameters

The available option identifiers are listed below and where applicable their default values.

Option ID	Option Values	Description
WIP_COPT_CERT	<const ascii*>	Client certificate.
WIP_COPT_CERT_AUTHORITY	<const ascii*>	CA Certificate.
WIP_COPT_PRIVATE_KEY	<const ascii*>	Client private key.
WIP_COPT_SEED	<const ascii*>	String random content, used to ensure that the key generation is secure. Must be null terminated.
WIP_COPT_VERIFY	WIP_SSL_NEVER	No certificate checking.
	WIP_SSL_ONCE	Check certificate once, will not necessarily recheck during renegotiation.
	WIP_SSL_ALWAYS	Always check certificate even during renegotiation. (Default value)
WIP_COPT_KEY	WIP_SSL_KEY_RSA	RSA key exchange
	WIP_SSL_KEY_DH_EPH	Temporary DH key, no DH cert
	WIP_SSL_KEY_ALL	All available keys. (Default value).
WIP_COPT_AUTHENTICATION	WIP_SSL_AUTH_RSA	
	WIP_SSL_AUTH_DSS	
	WIP_SSL_AUTH_NULL	NULL authentication. Used only for debug. Must be set explicitly.
	WIP_SSL_AUTH_ALL	All authentication schemes except NULL. (Default)
WIP_COPT_ENCRYPTION	WIP_SSL_ENC_DES	
	WIP_SSL_ENC_3DES	
	WIP_SSL_ENC_RC2	
	WIP_SSL_ENC_RC4	
	WIP_SSL_ENC_AES	

Option ID	Option Values	Description
	WIP_SSL_ENC_NULL	NULL encryption. Used only for debug. Must be set explicitly.
	WIP_SSL_ENC_ALL	All encryption algorithms except NULL. (Default)
WIP_COPT_MAC	WIP_SSL_MAC_MD5	
	WIP_SSL_MAC_SHA1	
	WIP_SSL_MAC_SHA2	Supports the SHA256.
	WIP_SSL_MAC_ALL	All hashes (Default)
WIP_COPT_VERSION	WIP_SSL_V3	Only allow SSL3.
	WIP_SSL_TLSV1	Only allow TLSv1.0
	WIP_SSL_TLSV1_2	Only allow TLSv1.2
	WIP_SSL_VER_ALL	All SSL protocol versions (Default)
WIP_COPT_SESSION_AUTO	WIP_SSL_SESSION_AUTO_ON	Activates the Automatic Session handling, valid until the library is closed.
	WIP_SSL_SESSION_AUTO_OFF	Does not start the Automatic Session handling, valid until the library is closed. (Default).
WIP_COPT_END	-	End of option list.

Note that an option id can appear more than once, e.g. the following line would allow AES and RC4 encryptions:

```
wip_SSLInitOpts(   WIP_COPT_ENCRYPTION, WIP_SSL_ENC_AES,
                   WIP_COPT_ENCRYPTION, WIP_SSL_ENC_RC4,
                   WIP_COPT_END);
```

3.3.4. Configuration Examples

SSL library initialization with a root certificate, own certificate and own private key. In addition to that, we specify that the library must verify validity of the peer certificate. Moreover, we impose use of RSA algorithm for authentication, AES algorithm for encryption and we only allow the TLS version 1.2 of the TLS/SSL protocol. Through the random_string entropy, the seed is added.

```
wip_SSLInitOpts(   WIP_COPT_CERT_AUTHORITY,         CA_CERTIFICATE,
                   WIP_COPT_CERT,                   MY_CERTIFICATE,
                   WIP_COPT_PRIVATE_KEY,            MY_PRIVATE_KEY,
                   WIP_COPT_VERIFY,                 WIP_SSL_ALWAYS,
                   WIP_COPT_AUTHENTICATION,         WIP_SSL_AUTH_RSA,
                   WIP_COPT_ENCRYPTION,             WIP_SSL_ENC_AES,
                   WIP_COPT_VERSION,                WIP_SSL_TLSV1_2,
                   WIP_COPT_SEED,                   random_string,
                   WIP_COPT_END);
```

3.3.5. Returned Values

This function returns:

- 0 if the SSL socket has been successfully initialize
- In case of an error, the function returns a negative error code that corresponds to an error value in wip_error_t.

3.4. SSL Library Termination

3.4.1. Function wip_SSLClose()

Close the whole SSL library and release associated resources. Note that this command will not accomplish the closure while SSL channels are still open; be sure to close all SSL channels before issuing this command.

3.4.2. Prototype

```
int wip_SSLClose ( void );
```

3.4.3. Parameters

None.

3.4.4. Returned Values

The `wip_SSLCLose()` function will return error `WIP_CERR_RESOURCES` unless all SSL channels are fully closed; as it takes some time after the call to `wip_close()` before the channel is actually closed (a cryptographic shutdown has to be exchanged), you cannot call `wip_SSLClose()` immediately after calling `wip_close()` on your last SSL channel. The best way to close the library is therefore to use a finalizer.

Finalizers provide the user with the ability to monitor the completion of a channel closure, as they are a function which is called after the channel has been released. For complete information regarding Finalizers, refer to document [1] Internet Library Connectivity Development Guide.

Caution: *Never use a wip_channel_t function in any way after wip_close() has been called on it; doing so may incur unspecified problems, including possible reboot and memory corruptions.*

```
static void try_close_lib( void *ctx);
static void close_SSL_channel_and_lib( wip_channel_t c) {
   wip_setOpts( c,
     WIP_COPT_FINALIZER, try_close_lib,
     WIP_COPT_END);
   wip_close( c);
}
```

```
static void try_close_lib( void *ctx) {
   if( OK != wip_SSLClose()) {
     wip_debug( "Couldn't close the SSL lib,"
                "maybe there are still some open channels?");
   }
}
```

3.5. Library Versions

3.5.1. Function wip_SSLGetVersion()

The `wip_SSLGetVersion()` function is used to get the versions of the Security Library, the underlying OpenSSL version and also the highest supported SSL version.

3.5.2. Prototype

```
const char* wip_SSLGetVersion (int type);
```

3.5.3. Parameters

Parameter name	Parameter Value	Description
type	0	The Security Library version.
	1	The underlying OpenSSL version.
	2	The highest possible SSL version capability of the current compilation of the Security library. This is regardless of the dynamic configuration of the system.

3.5.4. Example

Getting the underlying OpenSSL version string.

```
const char * versing_string_p = wip_SSLGetVersion( 1 );
```

3.5.5. Returned Values

This function returns a constant string

- Containing the version.
- Empty string in the error case.

4. Constructor and Channel Manipulation

4.1. Introduction

This chapter describes how to create channels for the Internet Library SSL to operate over, and addresses the constructor functionality. Note that conceptually this constructor and channels are designed to work virtually exactly like TCP or Internet Library channels do, as covered in document [1] Internet Library Connectivity Development Guide.

4.2. Channels Creation

SSL channel creation can only be requested after `wip_SSLInit()` or `wip_SSLInitOpts()` have been called, so that the library is properly configured.

4.2.1. Function wip_SSLClientCreate()

4.2.1.1. Prototype

```
wip_channel_t wip_SSLClientCreate( ascii *server, u16 port,
                                   wip_eventHandler_f evh, void *ctx);
```

4.2.1.2. Parameters

server:

In: Address of the SSL server.

port:

In: Port number of the SSL server socket.

evh:

In: Event handler attached to the channel.

ctx:

In: User data to be passed to the event handler every time it is called.

4.2.1.3. Returned Values

The SSL channel, or NULL on immediate failure.

4.2.2. Function wip_SSLClientCreateOpts()

4.2.2.1. Prototype

```
wip_channel_t wip_SSLClientCreateOpts (   ascii *server, u16 port,
                                    wip_eventHandler_f evh, void *ctx, ..);
```

4.2.2.2. Parameters

server:

In: Address of the SSL server.

port:

In: Port number of the SSL server socket.

evh:

In: Event handler attached to the channel.

ctx:

In: User data to be passed to the event handler every time it is called.

...:

A list of parameter pairs, end marker is as usual WIP_COPT_END.

This constructor also accepts the following options:

Option id	Option values	Description
WIP_COPT_SESSION	<wip_SSLSession_t>	All Session data including eventual Session ID and Session Ticket. The session data has to be recovered from a previously open connection through wip_SSLGetSession API:s.

4.2.2.3. Returned Values

The SSL channel, or NULL on immediate failure.

4.3. SSL Channels API

Since SSL client channels implements the wip_channel API, it supports the normal channel functions `wip_read()`, `wip_write()`, `wip_close()`. The behavior is specified below.

Please also refer to the documentation in [1] Internet Library Connectivity Development Guide

They also support the `wip_xxxOpts()` family of extended functions, although with a specific set of accepted options, which are summarized below.

4.3.1. The wip_read() function

4.3.1.1. Prototype

```
int wip_read( wip_channel_t c,
              void *buffer,
              u32 buf_len );
```

4.3.1.2. Parameters

c:

In: The channel to read from

buffer:

Out: Pointer to the buffer where read data must be put

buf_len:

In: Size of the buffer

4.3.1.3. Returned Values

This function returns

- number of bytes actually read
- In case of an error, a negative error code as described below

Option	Definition
WIP_CERR_NOT_SUPPORTED	This channel does not support data reading, or it has been provided with an option it does not support.
WIP_CERR_INVALID	Some option has been passed with an invalid value.

4.3.2. The wip_readOpts() function

There are no specific options supported for reading for an SSL channel. Please use wip_read().

4.3.3. The wip_write() function

4.3.3.1. Prototype

```
int wip_write( wip_channel_t c,
               void *buffer,
               u32 buf_len );
```

4.3.3.2. Parameters

c:

In: The channel to write to

buffer:

Out: Pointer to the buffer where write is to be found

buf_len:

In: Size of the buffer

4.3.3.3. Returned Values

This function returns

- number of bytes actually written
- In case of an error, a negative error code as described below

Option	Definition
WIP_CERR_NOT_SUPPORTED	This channel does not support data reading, or it has been provided with an option it does not support.
WIP_CERR_INVALID	Some option has been passed with an invalid value.

4.3.4. The wip_writeOpts() function

There are no specific options supported when writing to an SSL channel. Please use wip_write().

4.3.5. The wip_getOpts () function

Via wip_getOpts() it is possible to read some characteristics for the SSL channel. Note the difference between operations on the local certificate and the server certificate.

4.3.5.1. Prototype

```
int wip_getOpts( wip_channel_t c,
                 ...);
```

4.3.5.2. Parameters

c:

In: The channel to get options from.

...:

Out: [Option, OutBuffer, bufferSize] .The list must be terminated by WIP_COPT_END.

The recommended buffer size is the size of the buffer needed to extract the field successfully. The size is based on RFC:s and the underlying code, but local variations that require bigger buffers may be encountered.

If a field does not exist in a certificate or if the buffer provided is too small the error code returned is WIP_CERR_INVALID.

Option	Buffer Type and size	Definition
WIP_COPT_FINALIZER	Function pointer.	pointer to the callback function
WIP_COPT_VERSION	u32	local certificate version. Possible values : 0 = Version1 1 = Version2 2 = Version3
WIP_COPT_SERIAL	ascii buf[64]	local certificate serial number assigned by the CA. Returns the serial number as a null terminated string. Possible values :
WIP_COPT_SIG_ALGO	u32	local certificate algorithm identifier for the algorithm used by the CA to sign the certificate. 6 = rsaEncryption, 7 = md2WithRSAEncryption 8 = md5WithRSAEncryption 65 = sha1WithRSAEncryption 113= dsaWithSHA1

Option	Buffer Type and size	Definition
WIP_COPT_NOT_BEFORE	ascii buf[64]	local certificate validity- not valid before this date. The date string can best be described by this printf printf("%s %2d %02d:%02d:%02d %d%s", monthWithThreeFirstLettersInEnglish[M-1], day, hour, minute, second, year,(gmt)?" GMT":""); Example: “Sep 4 12:16:22 2018 GMT”
WIP_COPT_NOT_AFTER	ascii buf[64]	local certificate validity- not valid after this date. For format please see WIP_COPT_NOT_BEFORE.
WIP_COPT_PUB_KEY_ALGO	ascii buf[64]	local certificate public key algorithm
WIP_COPT_RSA_NUM_BITS	U32	local certificate number of bytes for the key if the algorithm is RSA
WIP_COPT_PUB_KEY	ascii buf[512]	local certificate public key
WIP_COPT_EXT_CRITICAL	Not Supported	local certificate critical extension
WIP_COPT_CA_SIG_ALGO	64 bytes	local certificate algorithm identifier for the algorithm used by the CA to sign the certificate
WIP_COPT_KEY_ID	ascii buf [1024]	local certificate a means to identify certificates that contain a particular public key
WIP_COPT_SUBJECT	-	see WIP_COPT_SUBJECT_CN.
WIP_COPT_ISSUER	-	see WIP_COPT_ISSUER_CN.
WIP_COPT_ISSUER_C	ascii buf[64]	local certificate issuer country.
WIP_COPT_ISSUER_ST	ascii buf[128]	local certificate issuer state or province name
WIP_COPT_ISSUER_L	ascii buf[128]	local certificate issuer locality
WIP_COPT_ISSUER_O	ascii buf[64]	local certificate issuer organizational name
WIP_COPT_ISSUER_OU	ascii buf[64]	local certificate issuer organizational unit name
WIP_COPT_ISSUER_CN	ascii buf[64]	local certificate issuer common name
WIP_COPT_SUBJECT_C	ascii buf[64]	local certificate subject country
WIP_COPT_SUBJECT_ST	ascii buf[128]	local certificate subject state or province name
WIP_COPT_SUBJECT_L	ascii buf[128]	local certificate subject locality
WIP_COPT_SUBJECT_O	ascii buf[64]	local certificate subject organizational name
WIP_COPT_SUBJECT_OU	ascii buf[64]	local certificate subject organizational unit name
WIP_COPT_SUBJECT_CN	ascii buf[64]	local certificate subject common name
WIP_COPT_PEER_CERT_VERSION	U32	connected peer certificate version . See WIP_COPT_VERSION
WIP_COPT_PEER_CERT_SERIAL	ascii buf[64]	connected peer certificate serial

Option	Buffer Type and size	Definition
WIP_COPT_PEER_CERT_SIG_ALGO	U32	connected peer certificate algorithm identifier for the algorithm used by the CA to sign the certificate
WIP_COPT_PEER_CERT_ISSUER_C	ascii buf[64]	connected peer certificate issuer country
WIP_COPT_PEER_CERT_ISSUER_ST	ascii buf[128]	connected peer certificate issuer state or province name
WIP_COPT_PEER_CERT_ISSUER_L	ascii buf[128]	connected peer certificate issuer locality
WIP_COPT_PEER_CERT_ISSUER_O	ascii buf[64]	connected peer certificate issuer organizational name
WIP_COPT_PEER_CERT_ISSUER_OU	ascii buf[64]	connected peer certificate issuer organizational unit name
WIP_COPT_PEER_CERT_ISSUER_CN	ascii buf[64]	connected peer certificate issuer common name
WIP_COPT_PEER_CERT_NOT_BEFORE	ascii buf[64]	connected peer certificate validity- not valid before this date. For format please see WIP_COPT_NOT_BEFORE.
WIP_COPT_PEER_CERT_NOT_AFTER	ascii buf[64]	connected peer certificate validity- not valid after this date. For format please see WIP_COPT_NOT_BEFORE.
WIP_COPT_PEER_CERT_SUBJECT_C	ascii buf[64]	connected peer certificate subject country
WIP_COPT_PEER_CERT_SUBJECT_ST	ascii buf[128]	connected peer certificate subject state or province name
WIP_COPT_PEER_CERT_SUBJECT_L	ascii buf[128]	connected peer certificate subject locality
WIP_COPT_PEER_CERT_SUBJECT_O	ascii buf[64]	connected peer certificate subject organizational name
WIP_COPT_PEER_CERT_SUBJECT_OU	ascii buf[64]	connected peer certificate subject organizational unit name
WIP_COPT_PEER_CERT_SUBJECT_CN	ascii buf[64]	connected peer certificate subject common name
WIP_COPT_PEER_CERT_PUB_KEY_ALGO	U32	connected peer certificate public key algorithm. See WIP_COPT_SIG_ALGO.
WIP_COPT_PEER_CERT_RSA_NUM_BITS	U32	connected peer certificate number of bytes for the key if the algorithm is RSA
WIP_COPT_PEER_CERT_PUB_KEY	ascii buf[1024]	connected peer certificate public key
WIP_COPT_PEER_CERT_EXT_OBJECTS	Not Supported	connected peer certificate extension objects
WIP_COPT_PEER_CERT_EXT_CRITICAL	Not Supported	connected peer certificate critical extensions
WIP_COPT_PEER_CERT_CA_SIG_ALGO	ascii buf[64]	connected peer certificate algorithm identifier for the algorithm used by the CA to sign the certificate
WIP_COPT_PEER_CERT_KEY_ID	ascii buf[1024]	connected peer certificate a means to identify certificates that contain a particular public key
WIP_COPT_PEER_CERT_FIRST_EXT	ascii buf[256]	connected peer certificate first extension
WIP_COPT_PEER_CERT_NEXT_EXT	ascii buf[256]	connected peer certificate next extension. Call repeatidly to get all available lines. Please notice that there is a limitation for the Authority Key Identifier, only the keyid is available.
WIP_COPT_END		end of options list

4.3.5.3. Returned Values

0 for OK or a negative error code indicating error.

Error Code	Description
WIP_CERR_INVALID	A field does not exist in a certificate or if the buffer provided is too small.
WIP_CERR_NOT_SUPPORTED	The function has been provided with an option it does not support.

4.3.5.4. Example

Getting the before date. Notice that the size of the variable also has to be passed.

```
char beforedate[64]={0};
ret = wip_getOpts(CHANNEL,
        WIP_COPT_NOT_BEFORE, &beforedate, sizeof( beforedate ),
        WIP_COPT_END);
```

4.3.6. The wip_setOpts() function

4.3.6.1. Prototype

```
int wip_getSetOpts( wip_channel_t c,
                    ...);
```

4.3.6.2. Parameters

c:

In: The channel to set options to.

...:

Out: [Option, optionValue] .The list must be terminated by WIP_COPT_END.

These are the options accepted by wip_setOpts() for a SSL channel.

Option	Definition
WIP_COPT_FINALIZER	pointer to the callback function
WIP_COPT_END	end of options list

4.3.6.3. Returned Values

0 for OK or a negative error code indicating error.

Error Code	Description
WIP_CERR_INVALID	A field does not exist in a certificate or if the buffer provided is too small.

4.3.6.4. Example

Setting the callback function that is called when the channel is closed.

```
wip_setOpts (sslChannel,
        WIP_COPT_FINALIZER, wipssl_try_close_lib,
        WIP_COPT_END);
```

4.3.7. The wip_close() function

The wip_close function instructs a channel to close.
The actual closing takes some time. Please see WIP_COPT_FINALIZER in the wip_setOpts, which described how to define a callback function to be called at the exact time the channel is closed.

4.3.7.1. Prototype

```
int wip_close( wip_channel_t c);
```

4.3.7.2. Parameters

c:

In: The channel to close.

4.3.7.3. Returned Values

0 for OK or a negative error code indicating error.

Error Code	Description
WIP_CERR_INVALID	NULL channel specified

4.3.7.4. Example

Closing a SSL channel.

```
wip_close( sslChannel );
```

4.3.8. The wip_abort() function

The SSL channel does not support the wip_abort function. Please use wip_close() to close the channel.

4.3.9. The wip_shutdown() function

The SSL channel does not support the wip_shutdown function. Please use wip_close() to close the channel.

5. Sessions

5.1. Introduction

When connecting to a SSL server, there is an initial handshake procedure.
There exist two features in the TLS/SSL protocol to reuse the initial handshake for future connections to the same server called (SSL) Session ID and (TLS) Session Ticket.

The principle is that both server and client keep the data from the initial handshake in a Session cache. When the client reconnects to the server, it presents the Session ID or Ticket from the previous connection.

If for some reason the Session ID or Ticket is not valid, the handshake is simply redone in the normal way, so there should not be big penalty if the wrong Session ID or Ticket would be presented to the server. Also note that most SSL server has a limited cache for Session IDs and Session Tickets, so that cache misses are not uncommon.

Sessions can be handled manually or automatically. Please carefully read the notes of the automatic session handling to determine which is appropriate for your application.
If both automatic mode and manual mode are used at the same time, the manual mode is used.

It is possible to encode a Session to a byte string, save this byte string in the file system across reboot and then reencode it to a Session. Please consider your servers session time out and the security impacts before doing this. See wipssl_EncodeSession for more information.

Sessions are handled internally by reference counting; every time a session is gathered with `wip_SSLGetSession()`, its reference counter is increased. Therefore, as soon as one has finished using a Session, one should release it with `wip_SSLReleaseSession()`.

5.2. Session Handling

Sessions can be reused: when a client and a server have been communicating together, they can keep the temporary keys. If both of them agree to reuse it, the protocol avoids renegotiating new keys, thus saving time and bandwidth.

Session are reference-counted: Internet Library SSL keeps track of how many sessions, programs etc. are interested by a given session, and releases it automatically when there is nobody left interested by it. This means that when a program wants to keep a sessions data, it must register itself as interested by the session, so that it won't be collected by Internet Library SSL. When the program stops being interested, it must tell the library that it isn't interested anymore by the session: this way, if there is no channel left using this session, the library will be able to collect them.

Therefore, in a sound program, there has to be as many calls to `wip_SSLGetSession()` as calls to `wip_SSLReleaseSession()`.

To reuse a session with a new connection, one must gather it from an already running session with `wip_SSLGetSession()`. Then, when creating a new connection to the same server (possibly after the previous one has closed), an option `WIP_COPT_SESSION, <wip_SSLSession_t>` must be passed to its `wip_SSLClientCreateOpts()` constructor.

5.3. Automatic Session Handling

The automatic session handling is either activated or deactivated at the initialization of the library.
To do this the library is initialized via wip_SSLInitOpts() with the option WIP_COPT_SESSION_AUTO set to WIP_SSL_SESSION_AUTO_ON.

The library then saves a reference to the Session variable and tries to reuse it at the next connection using the Session ID or Session Ticket stored in the Session.

Please note that the Automatic Session Handling does not check that the next connection is to the same server. So if the application connects to different servers, please use the manual handling.

The Session used by the Automatic Session Handling is released when closing the library with wip_SSLClose().

The priority between manual and automatic, is that manual prevails. So if a Session variable is provided manually to wip_SSLClientCreateOpts() when the automatic session handling is in active, the manually provided Session variable will be used.

5.4. Application Handled Sessions

If the application wants to maintain the Session handling manually there is an interface for getting and setting the Session which is used manually.

5.4.1. wip_SSLSession_t wip_SSLGetSession(wip_channel_t ssl_channel)

Get a handle on the session used by the SSL channel. The channel must be connected when the function is called. The resources associated with the session won't be released until a call to `wip_SSLReleaseSession()` is made. If there is an error or the channel is not open, function will return NULL.

5.4.2. wip_SSLSession_t wip_SSLGetSessionFromHTTPSDataChannel (wip_channel_t http_DataChannel)

This function is for getting the SSL Session when doing HTTPS.
Get a handle on the session used by the HTTP(S) channel. The channel must be connected when the function is called. The resources associated with the session won't be released until a call to `wip_SSLReleaseSession()` is made. If there is an error or if the channel is not open, it will return NULL.

5.4.3. void wip_SSLReleaseSession(wip_SSLSession_t ssl_session)

Release the handle on a SSL session. There must be as many calls to `wip_SSLReleaseSession() as to wip_SSLGetSession()` before the resource is actually freed.

5.5. Saving Sessions across Reboots

It is possible to encode a Session to a byte string, save this byte string across reboots and then re-encode it to a Session. Please consider your servers time out settings for session and also the security aspects.

5.5.1. wipssl_EncodeSession

5.5.1.1. Prototype

```
int wipssl_EncodeSession (wip_SSLSession_t in_session ,
                          unsigned char ** pp_out_buffer,
                          int * p_outbuff_len)
```

5.5.1.2. Parameters

in_session:

In: the session to be encoded.

pp_out_buffer:

Out: A pointer to a pointer that will contain the encoded Session.
Please note that the calling function is responsible to free this buffer with adl_memRelease()

p_outbuff_len:

Out: A pointer to a integer which will contain the size of the pp_out_buffer

5.5.1.3. Returned Values

The integer value that is returned will be 0 if success. A value separate from 0 for a failure.

5.5.2. wipssl_DecodeSession

5.5.2.1. Prototype

```
wip_SSLSession_t wipssl_DecodeSession ( const unsigned char * p_in_buffer,
                                        int in_buff_len);
```

5.5.2.2. Parameters

p_in_buffer:

In: the buffer containing the byte string of encoded Session.

in_buff_len:

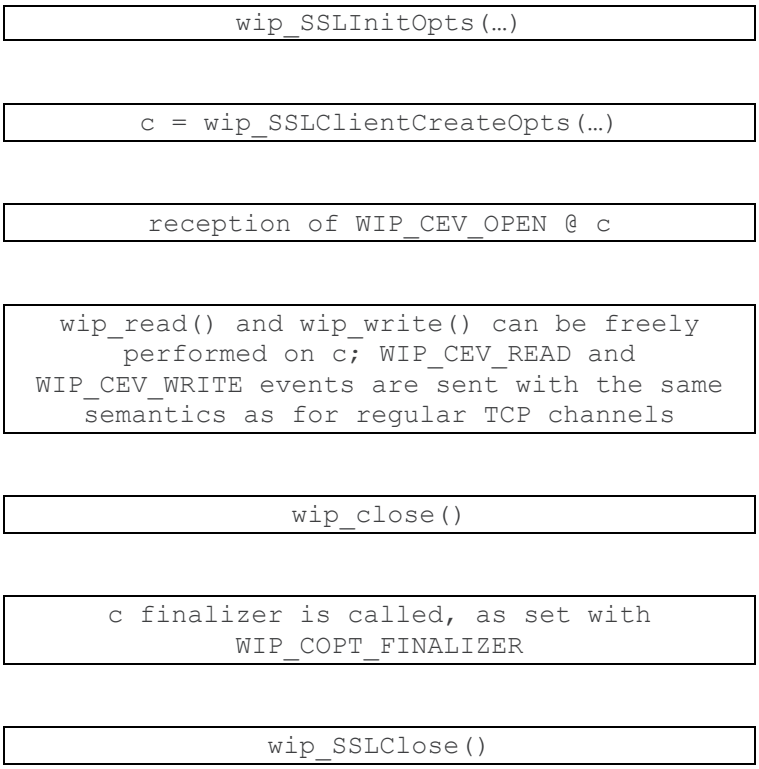
In: the size of the buffer p_in_buffer.

5.5.2.3. Returned Values

A pointer to the Session if success. If failure, then it will return NULL.
Important is that the calling function is responsible for freeing it using wip_SSLReleaseSession() when it is no longer needed.

6. Internet Library SSL Usage

6.1. Typical Call Sequence



6.2. Sample

The function apply_entry_point() is called after the IP stack has been fully initialized. See Internet Library samples to reach this state.

```
/* File entry_point.c */
#include "wip_ssl.h"
#define SERVER "192.168.1.5"
#define PORT   4433

// Keys and certificates, defined elsewhere
extern const char *CA_CERTIFICATE, *MY_CERTIFICATE, *MY_PRIVATE_KEY;

// event handler for the SSL channel
static void evh( wip_event_t *ev, void *ctx);
```

```
// Application entry point
void appli_entry_point() {
  wip_channel_t c;

  wip_debug( "[SAMPLE] Application starts\n");

  wip_SSLInitOpts( WIP_COPT_CERT_AUTHORITY, CA_CERTIFICATE,
                   WIP_COPT_CERT,           MY_CERTIFICATE,
                   WIP_COPT_PRIVATE_KEY,    MY_PRIVATE_KEY,
                   WIP_COPT_VERIFY,         WIP_SSL_ALWAYS,
                   WIP_COPT_END);

  c = wip_SSLClientCreate( SERVER, PORT, evh, NULL);
  if( ! c) { wip_debug( "[SAMPLE] error: Can't create channel\n"); }
}

// event handler for the SSL channel
void evh( wip_event_t *ev, void *ctx) {
  switch( ev->kind) {

  case WIP_CEV_OPEN:
    wip_debug("[SAMPLE] SSL connection successfully open\n");
    break;

  case  WIP_CEV_READ: {
    char buff[256]; int n, i;
    wip_debug("[SAMPLE] Server says:\n");
    do {
      n = wip_read( ev->channel, buff, sizeof( buff)-1);
      if( n<0) break;
      buff[n]='\0';
      wip_debug("[SAMPLE] received: \"%s\"", buff);
      for(i=0; i<n; i++)
        if( buff[i]>='a' && buff[i]<='z')
          buff[i] += 'A'-'a';
      i = wip_write( ev->channel, buff, n);
      wip_debug( "[SAMPLE] Wrote back %i bytes\n", i);
      if( i<n) { wip_debug("[SAMPLE] Couldn't write enough!\n"); }
    } while( sizeof( buff) - 1 == n);
    break;
```

```
  }

  case WIP_CEV_ERROR:
    wip_debug( "[SAMPLE] error %i\n", ev->content.error.errnum);
    break;


  case WIP_CEV_PEER_CLOSE:
    wip_debug( "[SAMPLE] connection closed by peer\n");
    wip_close( ev->channel);
    break;
  }
}
```

The keys and certificates are to be stored as character strings. Here are examples, which can be used for tests, but must not be used in production (as they are published in Sierra Wireless documentation they will not provide adequate security standards):

```
/* File keys.c */
const char *CA_CERTIFICATE =
  "-----BEGIN CERTIFICATE-----\n"
  "MIIDIjCCAougAwIBAgIJAN9xGzmGzA4+MA0GCSqGSIb3DQEBBAUAMGoxCzAJBgNV\n"
  "BAYTAkZSMRcwFQYDVQQIEw5IYXV0cyBkZSBTZWluZTEcMBoGA1UEBxMTSXNzeS1s\n"
  "ZXMtTW91bGluZWF1eDETMBEGA1UEChMKV2F2ZWNvbSBTQTEPMA0GA1UEAxMGd2lw\n"
  "c3NsMB4XDTA2MTIxNDE2MzAwOFoXDTE2MTIxMTE2MzAwOFowajELMAkGA1UEBhMC\n"
  "RlIxFzAVBgNVBAgTDkhhdXRzIGRlIFNlaW5lMRwwGgYDVQQHExNJc3N5LWxlcy1N\n"
  "b3VsaW5lYXV4MRMwEQYDVQQKEwpXYXZlY29tIFNBMQ8wDQYDVQQDEwZ3aXBzc2ww\n"
  "gZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALrLrMLZTchpAM9oMzCXCf//ivW7\n"
  "BoZ9bXoF8eISkf+ddDf2dpUmR5lbqwVAbHDm0i71PcVX7TZOWgkO0A0nN00dHy4J\n"
  "0D6w6Ge7H2te2KBH7XWodPOwMhR00j1e9E7XU7n5mFjotbsk3fQ4fqYZH9M/UJPE\n"
  "4eMz+odgNobMtqOnAgMBAAGjgc8wgcwwHQYDVR0OBBYEFPL9xIwppGPnZ4yV4BNS\n"
  "W4Lb4LtzMIGcBgNVHSMEgZQwgZGAFPL9xIwppGPnZ4yV4BNSW4Lb4LtzoW6kbDBq\n"
  "MQswCQYDVQQGEwJGUjEXMBUGA1UECBMOSGF1dHMgZGUgU2VpbmUxHDAaBgNVBAcT\n"
  "E0lzc3ktbGVzLU1vdWxpbmVhdXgxEzARBgNVBAoTCldhdmVjb20gU0ExDzANBgNV\n"
  "BAMTBndpcHNzbIIJAN9xGzmGzA4+MAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEE\n"
  "BQADgYEAbm5j3kXs25Le+20SsPbhw7bX5lc8cNDMCRal9YcVCLG71rUxXdmcxZb4\n"
  "yEDMHRDZ+JcA5WUUTjG3W+jgAdVP7ppExiuRwgFUpUIa1uexA1WY/a/Mv0f91GXA\n"
  "+u6o+tMIFr6hsngw2qeDzuwfnsqvfTRdEWxqXXNSMsk/K+Nf4uk=\n"
  "-----END CERTIFICATE-----\n";

const char *MY_CERTIFICATE =
```

```
  "-----BEGIN CERTIFICATE-----\n"
  "MIIDJzCCApCgAwIBAgIBATANBgkqhkiG9w0BAQQFADBqMQswCQYDVQQGEwJGUjEX\n"
  "MBUGA1UECBMOSGF1dHMgZGUgU2VpbmUxHDAaBgNVBAcTE0lzc3ktbGVzLU1vdWxp\n"
  "bmVhdXgxEzARBgNVBAoTCldhdmVjb20gU0ExDzANBgNVBAMTBndpcHNzbDAeFw0w\n"
  "NjEyMTQxNjMyMTdaFw0xNjEyMTExNjMyMTdaMEwxCzAJBgNVBAYTAkZSMRcwFQYD\n"
  "VQQIEw5IYXV0cyBkZSBTZWluZTETMBEGA1UEChMKV2F2ZWNvbSBTQTEPMA0GA1UE\n"
  "AxMGd2lwc3NsMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDX7FDBP5jK7yOW\n"
  "63Ewo+EDQPv/7qPM9CyJk5m1n2cj0gyWSLjP3w9tjqO1/G8j7sAJDY4TP6fI7bWF\n"
  "5JPTFJVzuoZeW0Gs5JN2sxto085eBPRF/XxPMgDr6lNzGULUGLykResTUo6yaQLp\n"
  "CedF0EHo+p8ycDiADVmxca8T39mm8wIDAQABo4H6MIH3MAkGA1UdEwQCMAAwLAYJ\n"
  "YIZIAYb4QgENBB8WHU9wZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1Ud\n"
  "DgQWBBTxb9jtHCwg6Tg93wwbNeO3YW6GGzCBnAYDVR0jBIGUMIGRgBTy/cSMKaRj\n"
  "52eMleATUluC2+C7c6FupGwwajELMAkGA1UEBhMCRlIxFzAVBgNVBAgTDkhhdXRz\n"
  "IGRlIFNlaW5lMRwwGgYDVQQHExNJc3N5LWxlcy1Nb3VsaW5lYXV4MRMwEQYDVQQK\n"
  "EwpXYXZlY29tIFNBMQ8wDQYDVQQDEwZ3aXBzc2yCCQDfcRs5hswOPjANBgkqhkiG\n"
  "9w0BAQQFAAOBgQAMBdle6nZssUjCC1HP1j1WjEXHJXbSCs7UftqMemRJJjgQpa38\n"
  "HPHd+42BSUCiSH1Sn22eVpD1+YWIWn4TdBgwypt/sJey+c1SYB/f1Eqd3xkejb9S\n"
  "RqJVd62L9ux9F5kB0HwHF0L5ZpSvx4WRSxb3iZ217ysh3GmbDhl6QLSgGw==\n"
  "-----END CERTIFICATE-----\n";

const char *MY_PRIVATE_KEY =
  "-----BEGIN RSA PRIVATE KEY-----\n"
  "MIICXQIBAAKBgQDX7FDBP5jK7yOW63Ewo+EDQPv/7qPM9CyJk5m1n2cj0gyWSLjP\n"
  "3w9tjqO1/G8j7sAJDY4TP6fI7bWF5JPTFJVzuoZeW0Gs5JN2sxto085eBPRF/XxP\n"
  "MgDr6lNzGULUGLykResTUo6yaQLpCedF0EHo+p8ycDiADVmxca8T39mm8wIDAQAB\n"
  "AoGAen1KynYDfYvvypvB2G//I9NnoaaFMa2K3njnB8tnvUBZd5/Fh9bob6QtZv3P\n"
  "Jrk4I2qXIIBJ9Ig1I8GpwmK47KQ6Ky2zBHTBcbGzhaDnECJOwTlqZe2Iv2IO96zo\n"
  "sKy7kCFVooQeOeeS21E98ko0aBsVPZ/ZFLANbXQ9qQv2V4ECQQD//ZbWoYln0kNv\n"
  "g5+nwWW9Q214hPOPYdz7f/P+bYGDrrQD+fua1GrH0AW2SMYmnDdSoIJFbpNl5IdN\n"
  "fJZtwSBhAkEA1+5ZUZm4UYM5mtdftNNGBmTC0G/nuGO5WcmjTCIPi8/kPhnilq88\n"
  "HjZFCQ0S71qtkdA3y8YUFpDna5vnE4tX0wJARSAUODb8pLVpklZHqYQW1gm8KNw1\n"
  "7NTvWFaP63dkjsuBPsWlRITxpK0ura9vGoP6iGxhYSBf2xbf1nO7Jz4MYQJBAMxn\n"
  "5geYA+Kt3V8V2JSdl2FACyczd+CWDoTPmxTb/Wl1n/Oln1jTg456AzoBNVZ9uWcZ\n"
  "+2ecF7IQ8/FrAQEAXF8CQQDhhz0SB3zWEQASOoUJ3y3cVrISm4FAW9KOGy/0NNGq\n"
  "grgMzLppMep4drFbLoC/VUBPPuIH/FO/+R/JDfuadGmR\n"
  "-----END RSA PRIVATE KEY-----\n";
```

These keys are direct copies of the ones put in files by OpenSSL. For details about key generation and handling, please refer to OpenSSL document.

7. Troubleshooting

7.1. Internet Library Trace Service

Internet Library contains a trace service that can be of assistance when troubleshooting issues with this API. Be sure to refer to Internet Library trace logs when troubleshooting problems related to Internet Library.

7.2. Low Security Errors

Note that Security Library issues an error to users who are testing their system with very basic security keys indicating that their system could easily be hacked or the like. Be sure that your security key is designed in a random fashion based on high security principles to ensure avoiding these errors (and increasing your security as well).

Appendix A. Interface Update

The current version represents a major update of the underlying OpenSSL solution and capabilities, but in most aspects the interface stays the same.

Some things minor things have been changed. This appendix tries to clarify the major changes of the interface from the previous WIPSSL solution.

A.1 Enums Removed

Old Name	Description
WIP_SSL_DH_RSA	Removed. Insecure and no longer supported.
WIP_SSL_DH_DSS	Removed. Insecure and no longer supported.
WIP_SSL_V2	Removed support for SSL version 2. Very Insecure.

A.2 Enums Renaming

For some enums were used for multiple things and have not only been renamed, but split into one specific for each usage.

Old Name	New Name	Description
WIP_SSL_NULL	WIP_SSL_AUTH_NULL	A split for each usage.
	WIP_SSL_ENC_NULL	
WIP_SSL_ALL	WIP_SSL_KEY_ALL	A split for each usage.
	WIP_SSL_AUTH_ALL	
	WIP_SSL_ENC_ALL	
	WIP_SSL_MAC_ALL	
	WIP_SSL_VER_ALL	
WIP_SSL_RSA	WIP_SSL_KEY_RSA	The enum has been split in two. This way the case for KEY and AUTH is separated.
	WIP_SSL_AUTH_RSA	
WIP_SSL_DH_EPH	WIP_SSL_KEY_DH_EPH	
WIP_SSL_DSS	WIP_SSL_AUTH_DSS	
WIP_SSL_DES	WIP_SSL_ENC_DES	
WIP_SSL_3DES	WIP_SSL_ENC_3DES	
WIP_SSL_RC2	WIP_SSL_ENC_RC2	
WIP_SSL_RC4	WIP_SSL_ENC_RC4	
WIP_SSL_MD5	WIP_SSL_MAC_MD5	
WIP_SSL_SHA1	WIP_SSL_MAC_SHA1	

A.3 Default Values

Before the default values was set to support all possible values.
That has now in some cases changed, and the default values are now set to values that are more secure.

Value	Default values	Comment
WIP_COPT_VERIFY	Always	
WIP_COPT_KEY	All	
WIP_COPT_AUTHENTICATION	All except NULL	
WIP_COPT_ENCRYPTION	All except NULL	
WIP_COPT_VERSION	All	

A.4 Changed Accepted Parameters wip_setOpts()

The function wip_SetOpts() no longer supports the following two parameter.

Option	Definition
WIP_COPT_SESSION	SSL session to resume (variable type-wip_SSLSession_t)
WIP_COPT_CHANNEL	TCP channel on which we want create SSL channel (variable type - wip_channel_t)

Index

setOpts, 24, 25, 26
wip_SSLClientCreate, 17
wip_SSLClientCreateOpts, 18
wip_SSLClose, 15
wip_SSLInit, 12
wip_SSLInitOpts, 13, 16
writeOpts, 20

SIERRA
WIRELESS