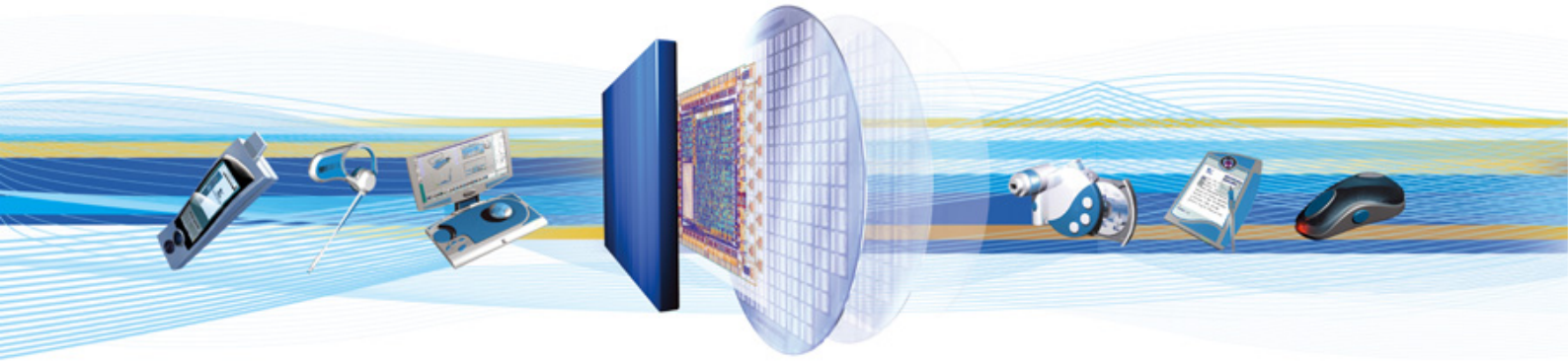
CSR Synergy Framework 3.1.1

Transport Layer Security (TLS)

API Description

November 2011



CSR

Cambridge Science Park
Milton Road
Cambridge CB4 0WH
United Kingdom

Registered in England 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the API between the CSR TLS and the application running in the CSR scheduler. The document is intended for developers utilising the API for their own application. The underlying CSR TLS task exists only in a generic Linux port using OpenSSL. Presently, no platform independent (GSP) or Windows (pcwin) port is available.

In some cases, TLS functionality can already be available on host platform and porting the API to use these libraries is therefore reasonable. Therefore, the document also includes some porting notes which developers should pay close attention to when porting the library.

1.2 Assumptions and Dependencies

TLS is an acronym for Transport Layer Security. It is used for communicating securely between two parties meaning a third party is unable to eavesdrop on the communication. While TLS conceptually is not limited to traffic using a specific underlying transport protocol, the CSR TLS API is designed solely for use with TCP carried over IPv4 or IPv6 over some unspecified medium.

The following assumptions and preconditions are made:

- The lower layers are implemented
- Only one instance of CSR TLS is active at any time
- Configuration of the interfaces is handled by CSR TLS API

The following dependencies are present:

- OpenSSL v. 1.0.0 in terms of libSSL and libCrypto

1.3 Features

The CSR TLS API has the following features:

- Support for TLS v 1.2 Ciphers as defined in RFC 5246
- Supports IPv4 and IPv6
- Support the application to be client
- Facilitate a high level API, for rapid development

It is important to note that while the API supports both IPv4 and IPv6, it is optional for an implementation of CSR TLS to provide *functional* IPv6 support. If it does not implement support for IPv6, the API defines the semantics for signalling this to an application, and applications using CSR TLS must be able to handle lack of IPv6 support in CSR TLS gracefully.

1.4 Limitations

In the current port for Linux, the following limitations are present:

- The verification process does not take min and/or max key size specified in CSR_TLS_CONFIG_CIPHER signal into account
- Similarly not all cipher suits defined in RFC 5246 are supported in OpenSSL. These are marked in the primitive description

- Multiple connections per CSR TLS session as well as connection resumption are currently not supported
- Certificates of ASN.1 style are currently not supported
- Configuration of compression level is currently not available

1.5 Prerequisites

This document assumes that the reader has a fundamental and thorough understanding of IP communication as well as secure communication architecture. Moreover, it is assumed that the reader has a sound understanding of the CSR Synergy Framework and its scheduler. If not, we would like to point the reader to the CSR Synergy Framework User Guide and the CSR Synergy Framework Scheduler API.

2 Description

This section will briefly describe the purpose of the CSR TLS API, and where the CSR TLS API is located in the CSR Synergy Framework.

2.1 Introduction

The CSR TLS API is intended to make it easy for customers to port and write applications to use CSR TLS as their provider of a secure connection to a server. The CSR TLS API is wrapping the establishing of connection, negotiating cryptographically condition for the connection as well as the decrypting and encrypting all messages with a minimum of interaction with the application. The main part of additional interaction with the application compared to a regular CSR IP Socket connection is in the configuration part, which is optional.

If TLS functionality already exists on the end platform, the developer can also choose to port the CSR TLS API to use the platform library, similarly to this generic Linux port. Such a port could possibly take outset in a ported version of OpenSSL.

2.2 Architecture

Figure 1 illustrates the architecture in which the CSR TLS API is operating relative to the application and OpenSSL.

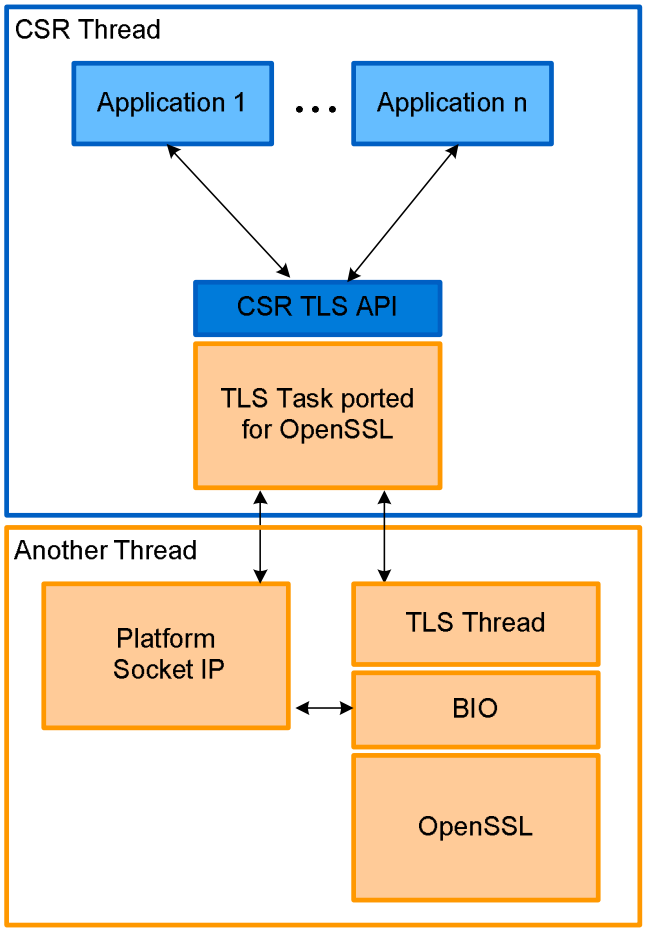


Figure 1: The CSR TLS API shown relative to OpenSSL and the platform Socket IP

2.3 Definitions

The following definitions are used in the document:

Definition	Meaning
Application	Application is referring to the task which the developer develops and which uses TLS for communication.
CSR TLS	CSR TLS is the task in the scheduler with which the application communicates.
CSR TLS Session	A CSR TLS session is an umbrella for the connections, containing information about the security configuration for the communication with a server and keeping track of the underlying connections. Although it is not a limitation in the CSR TLS API itself, it is not the intention that one session should be used for communication with different servers. When communicating with several servers, a session for each server is needed.
CSR TLS Connection	Inside a CSR TLS session, a connection to the server using the session configuration can be established. The CSR TLS connection is the connection using IP socket to connect to the server. Currently CSR TLS only supports *one* connection per session, but the API is prepared for several concurrent connections to the same server inside the same CSR TLS session reusing the session configuration.

2.4 Porting

The CSR TLS is like other CSR Synergy tasks designed to use message passing and to be non-blocking. However, the OpenSSL is by default a function library that has a blocking nature due to the crypto engine which is a computational intensive task. In order not to block the rest of the CSR Synergy tasks it is necessary to have a separate thread for OpenSSL.

If it is decided to port the CSR TLS API to use other TLS libraries which are blocking, it is important to maintain this disconnection between the CSR TLS task and at least the crypto part of the TLS library.

To enable flexibility of the TLS PSK implementation a standard call-back is present which uses the PSKs delivered in the `CSR_TSL_CONFIG_CIPHER_REQ` message, however by setting the compiler define `CSR_TLS_EXTERNAL_PSK_CLIENT_CALLBACK_FUNCTION` the application is able to create its own function to handle this.

Because of the similarities between the CSR TLS and CSR IP tasks, it is possible to configure CSR Synergy in a way such that the IP and TLS tasks share a common worker thread instead of having to threads that perform similar tasks. This is done by setting the CSR_IP_SUPPORT_TLS make variable when building or, alternatively if not using the CSR Synergy build system, defining the CSR_IP_SUPPORT_TLS symbolic constant when compiling. However, it is very important to note the following consequence of using this configuration because it has important consequences for the applications using the CSR IP and CSR TLS APIs.

First of all, running both TLS and IP in a single thread means that it is not possible to take advantage of multiple processors which would allow IP and TLS processing to be done in parallel. Since TLS involves crypto, which requires a lot of processing, it may be desirable to let TLS and IP run in parallel to avoid disrupting IP communications.

Secondly, in some situations TLS processing suspends the worker thread while it waits for a response from the application on whether or not it accepts a given certificate. This means that when TLS and IP share a worker

thread, for some period of time both TLS and IP will be suspended until the application responds with a certificate acceptance request. This means that existing IP connections will be unable to exchange data during this period but also that it is not possible to establish new connections. This means that an application must not rely on IP communication when verifying certificates or the system will deadlock. For instance, if it wishes to check if the certificate is not present on a CRL (certificate revocation list), it must fetch this list prior to establishing the TLS connection. This also means that an application should try to respond to certificate acceptance indications as quickly as possible.

3 Interface Description

In the following, we will present the signal flow for the different part (initialisation, data transfer and disconnection) of the interface in terms of message sequence charts. The descriptions only illustrate the standard case scenarios, and we point the readers to the primitive description in section 3.4, for more details about how use the different signals, when other conditions occur.

3.1 Initialisation

In Figure 2 the initialisation of the secure connection is illustrated. It is important to notice that TLS operates with two levels for connections: CSR TLS sessions and CSR TLS connections. Developers being used for working with the CSR IP API, please note that these cannot be directly compared with the socket IP. An important difference is that a session, as defined in section 2.3, can contain more than one secure connection, using the same negotiated properties. However, the application needs to wait for a `CSR_TLS_CONNECTION_CREATE_CFM` before trying to create the next connection inside a session.

A secure session is initiated by signalling a `CSR_TLS_SESSION_CREATE_REQ`. The signal must contain the schedule handle to the calling application. Before creating a connection to a specific server, the application can choose to change the default cipher configuration using `CSR_TLS_CONFIG_CIPHER_REQ`. Furthermore, the connection can be configured to use a specific client certificate and/or key-pair by `CSR_TLS_CONFIG_CERT_CLIENT_REQ`. Before creating the connection the server certificate pre-validation can also be configured by sending a `CSR_TLS_CONFIG_CERT_VERIFY_REQ`.

A connection can be created using `CSR_TLS_CONNECTION_CREATE_REQ` that creates a socket context in the CSR TLS task and creates a socket in the underlying Linux IP Socket interface. When the connection is created, a handle ID for the connection is returned. Having the connection established the application can choose to bind the connection to a local port and IP using `CSR_TLS_CONNECTION_BIND_REQ`. This command is just a pass though from the CSR TLS to the underlying Socket IP.

The application can now establish a connection using the IP address and port of the remote server together with `CSR_TLS_CONNECTION_CONNECT_REQ`. The CSR IP Socket can be used for resolving the IP address of a server using its DNS service, if the IP is not directly known by the application. Having an established connection, CSR TLS will negotiate the cryptograph condition for the connection. This has been hidden for the application except for the fact that the application needs to approve the certificate of the server. This is sent via a `CSR_TLS_CONNECTION_CERT_IND`, and includes the status of the CSR TLS task pre-verification of the certificate. Please see section 3.4.3 for more information. When the connection is established, CSR TLS will send a `CSR_TLS_CONNECTION_CONNECT_CFM`, and the application can now use the connection to transmit and receive data securely.

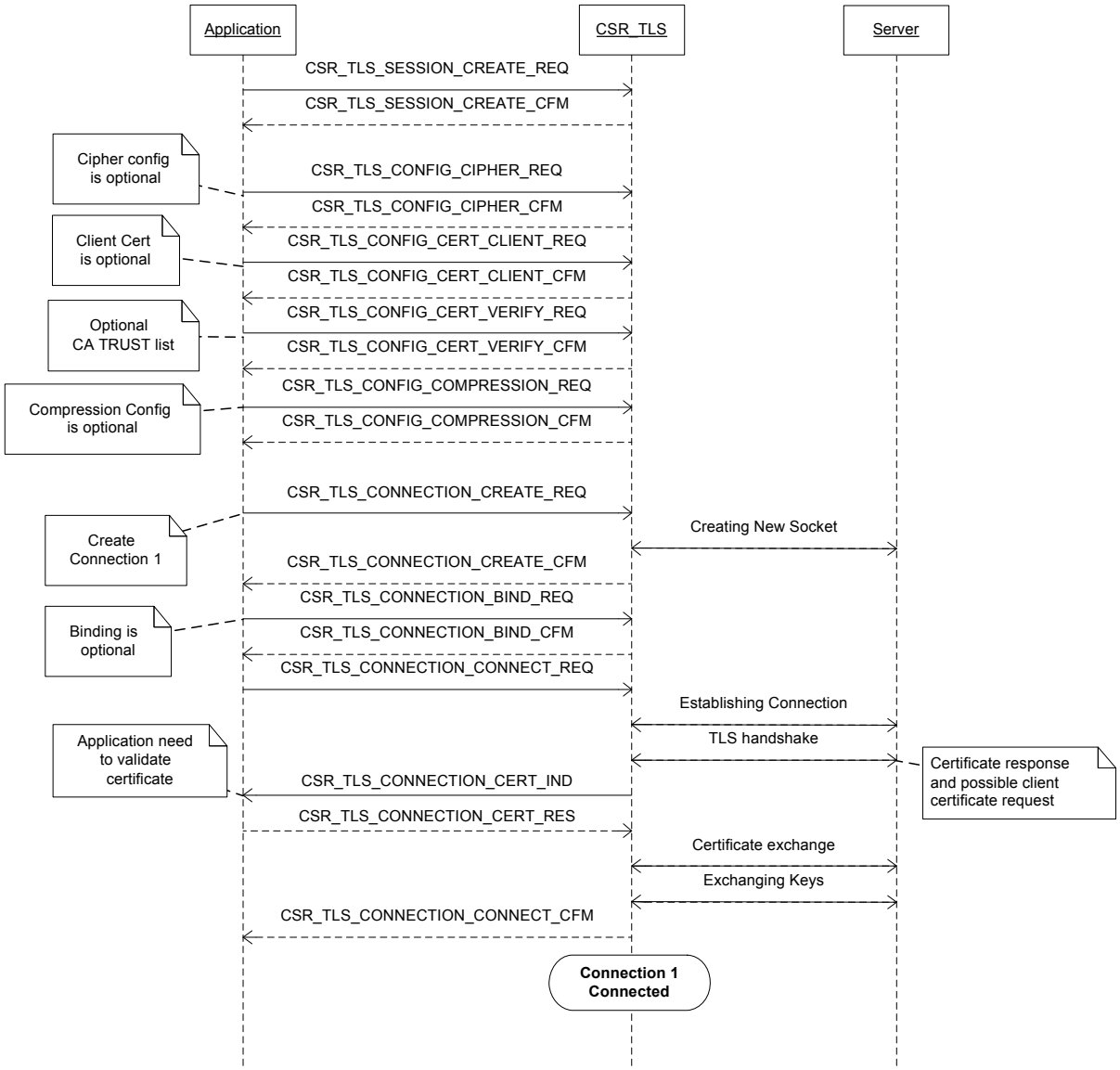


Figure 2: MSC for initialisation of the CSR TLS session

3.2 Data Transfer

Figure 3 illustrates the signals being sent when receiving and transmitting data. Data can be transmitted using `CSR_TLS_CONNECTION_DATA_REQ` which must contain handling of the connection with the receiver, length of and a pointer to the payload, which is about to be sent. CSR TLS will then take care of encrypting the data. When CSR TLS is ready to receive more data, a `CSR_TLS_CONNECTION_DATA_CFM` is sent.

Data is received by a `CSR_TLS_CONNECTION_DATA_IND` that contains the connection id on which the data are received. The data will arrive decrypted to the application. When the application has read the data, it has to send `CSR_TLS_CONNECTION_DATA_RES` to signal to CSR TLS that it is ready to receive more data.

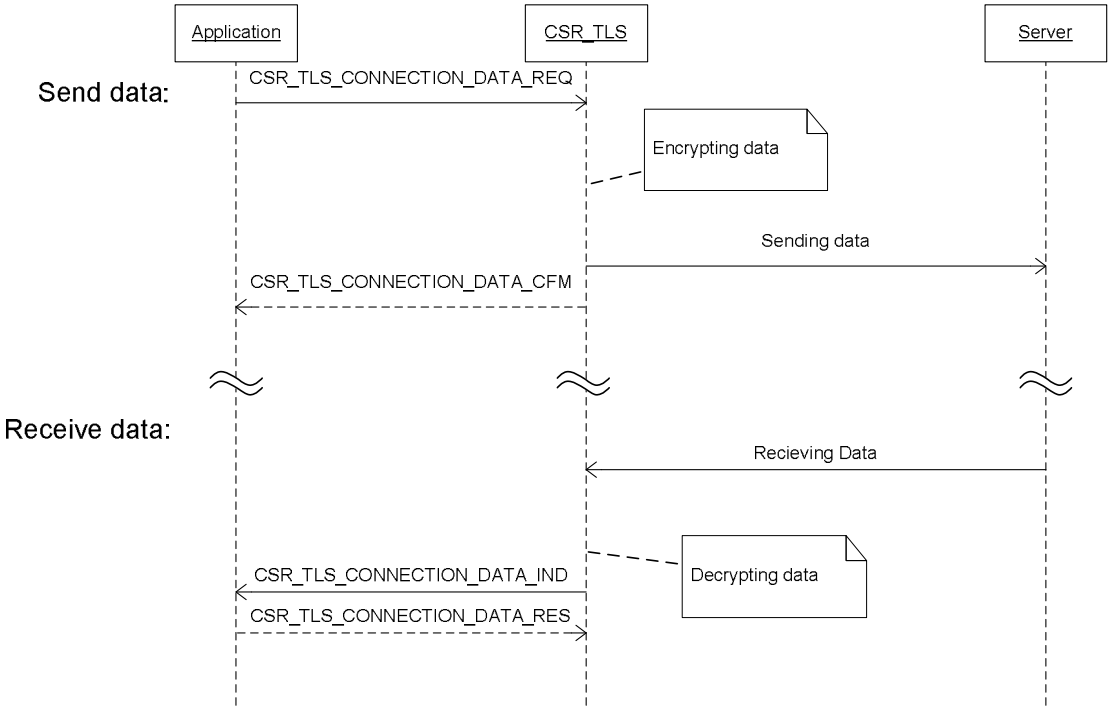


Figure 3: MSC for data transfer in a CSR TLS session

3.3 Disconnection

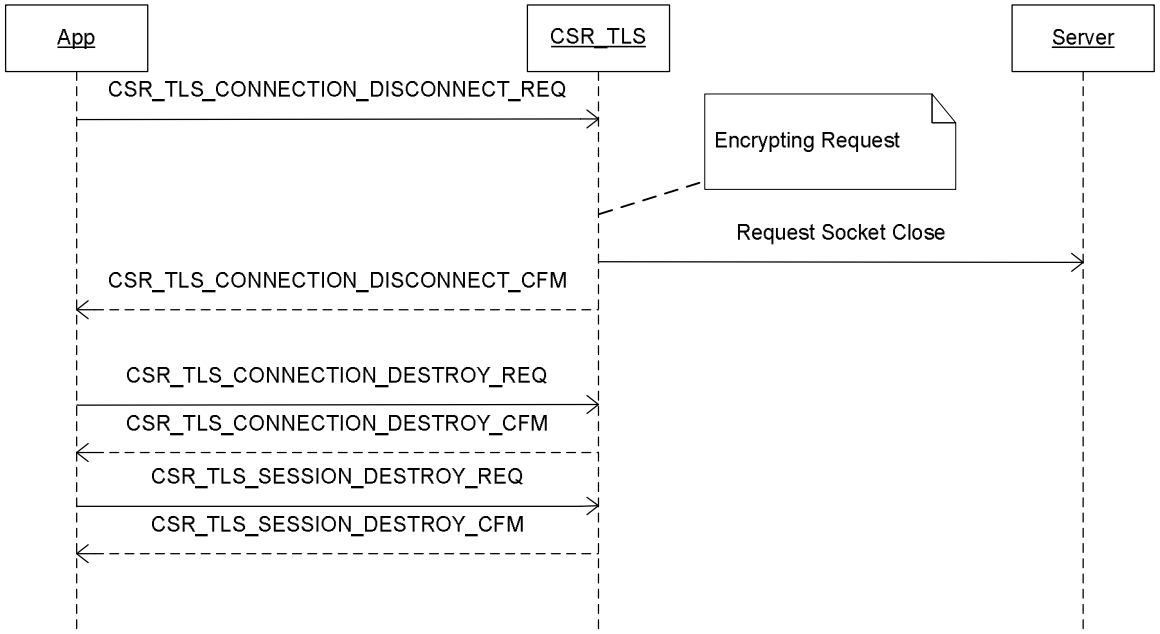


Figure 4 and

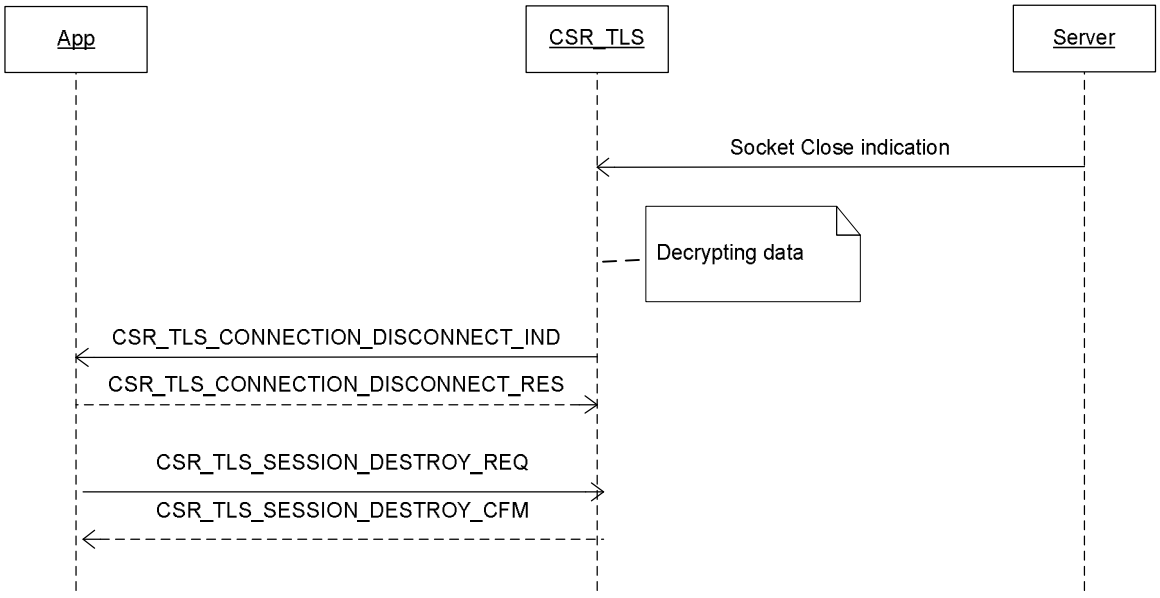


Figure 5 show disconnection request from client and server respectively. When the application wants to disconnect a connection, it has to signal `CSR_TLS_CONNECTION_DISCONNECT_REQ`. When the TLS task has closed down the connection and no more data will be received or transmitted, it will send `CSR_TLS_CONNECTION_DATA_CFM`. The application can now reopen the connection again by signalling a new connect or destroy it by sending a `CSR_TLS_CONNECTION_DESTROY_REQ`. When destroying the connection, it can no longer be reopened. When the session is no longer needed the application can destroy it by sending a `CSR_TLS_SESSION_DESTROY_REQ`.

The application also needs to be ready to accept that the server side closes a session. The flow is as in

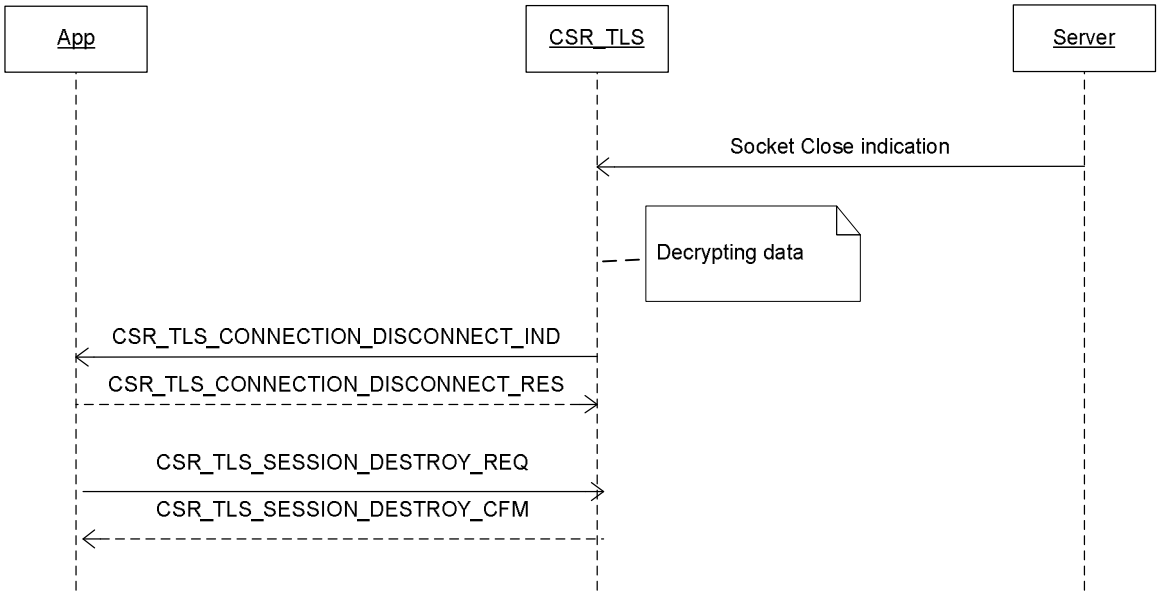


Figure 5 which is similar to the close down from client side although the requests come from the server. One major difference is that when the server closes a connection, the signal will implicitly also contain a destruction of the connection.

Finally, it is also possible to make a “dirty” disconnection by just destroying the session via `CSR_TLS_SESSION_DESTROY_REQ`. The TLS task will then take care of closing all connections inside this session. This is illustrated in

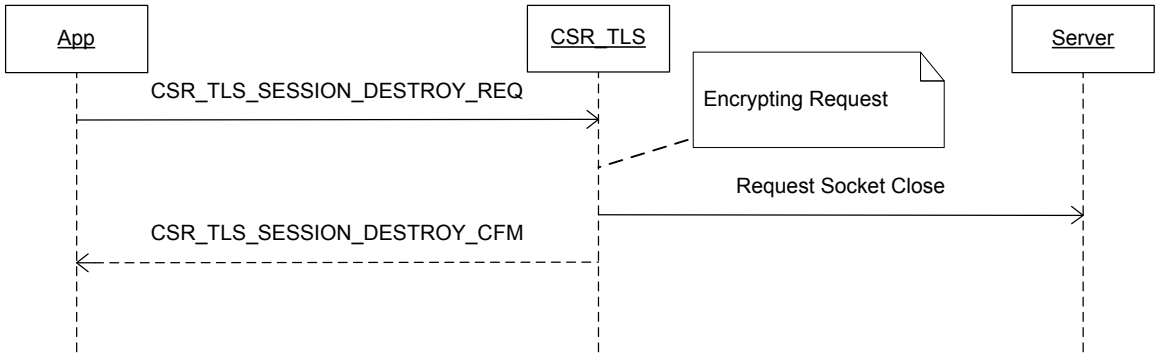


Figure 6.

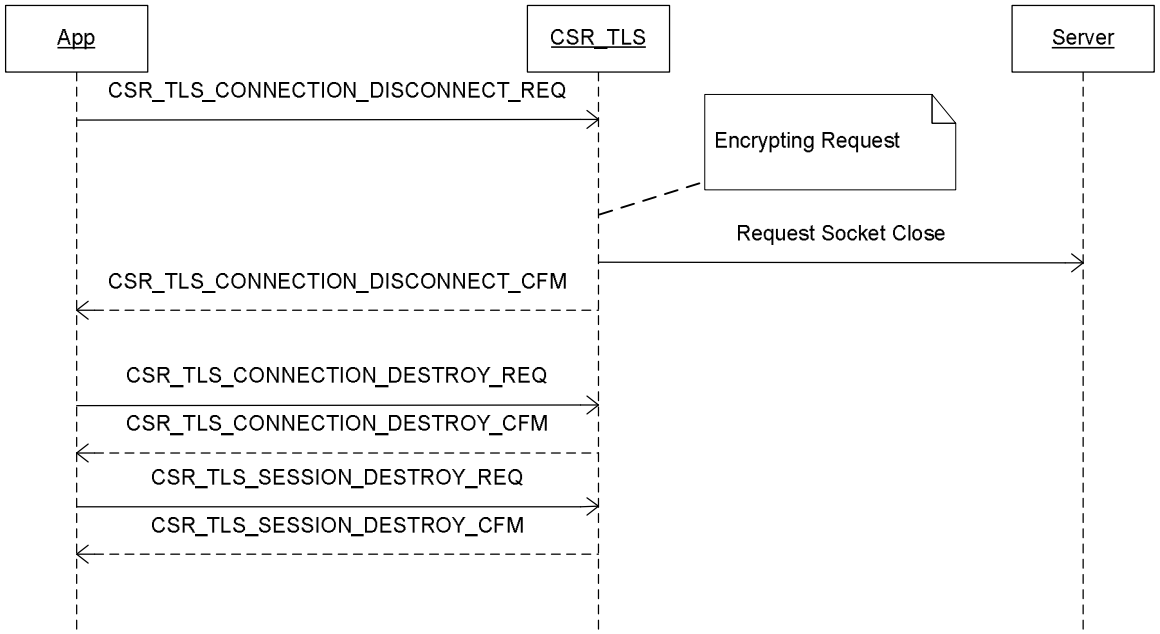


Figure 4: MSC for disconnecting and destroying a CSR TLS session from client side

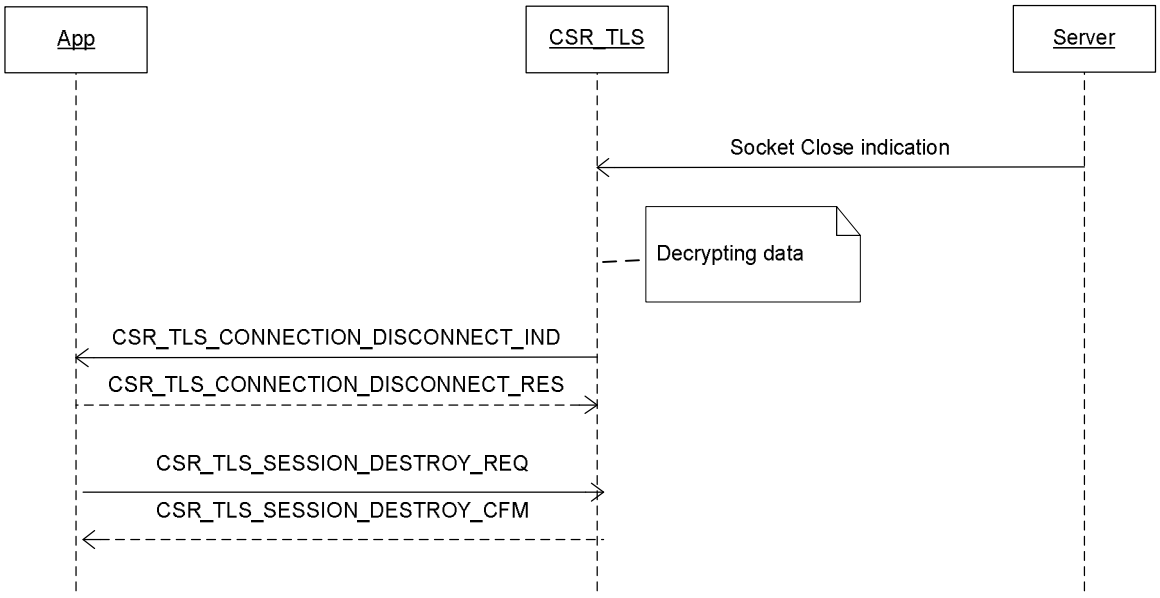


Figure 5: MSC for disconnecting and destroying a CSR TLS session from server side

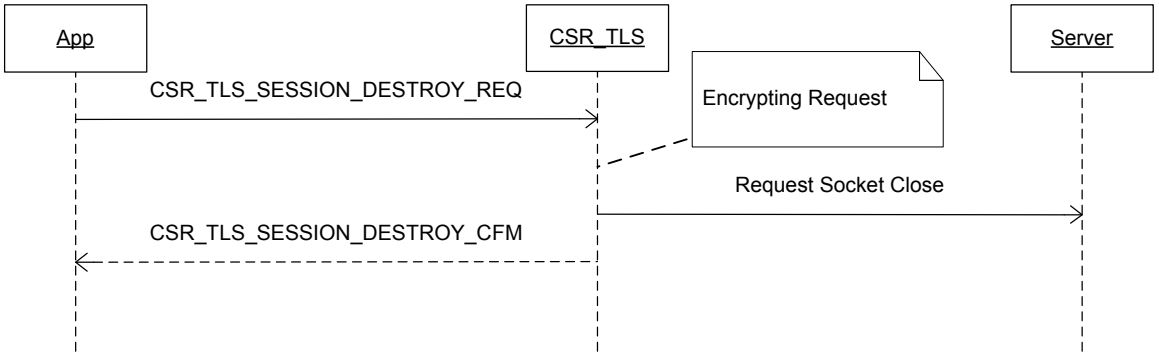


Figure 6: MSC for dirty disconnecting and destroying a CSR TLS session from client side

3.4 Certificate Format and Validation Basics

In the following, some details about the certificates and its validation will be presented:

3.4.1 Certificate and Keys

When establishing a secure connection using public key cryptography an asymmetric key algorithm is used. Here both a public and private key is needed. The private key is private to the client or server side, but the public key will be sent to the other party. Together with the public key of at least the server, typically a certificate containing information about the sender is sent. Besides the public key, the certificate contains information about which algorithm is used for key generation and hashing, the name and address (post address) of the sender, the period of validity of the certificate, the purpose of the certificate, a hash of the certificate to verify that it has not been changed and more. This information is stored in a X509v3 format.

3.4.2 PEM Format

The PEM (Privacy Enhanced Mail) format is an encoding scheme that encodes the X509v3 format into base64 plus adding "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----" to it. Like this:

```
-----BEGIN CERTIFICATE-----
MIICdzCCAeCgAwIBAgIBBDANBgkqhkiG9w0BAQUFADBNMQswCQYDVQQGEwJESzEQ
MA4GA1UEBwwHQWFsYm9yZzEMMAoGA1UECgwDQ1NSMR4wHAYDVQQDDBVDZXJ0aWZp
Y2F0ZSBBdXRob3JpdHkwHhcNMTAwNzI4MDAwMDAxWhcNNDkwNzI4MDAwMDAxWjA5
MQswCQYDVQQGEwJESzEMMAoGA1UEChMDQ1NSMRwwGgYDVQQDExNDU1IgVExTIFRl
c3QgQ2xpZW50MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDk/Pf7cgJ794fn
siGVCqrwVgQFeKcMRGs3f6zqZ6LbKC3rk4UqDE88BIPYSf/HokgGXTvrmWloWsuu
43wjFgbZgWFkR080wZS39KBNUzI/oNfPLiqi7y6RRHUtAAuJPHMLUwWHBWTnJlvA
i4p6eI24gP8+f3laynbulP7MEtHCywIDAQABo3sweTAJBgNVHRMEAjAAMCwGCWCG
SAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4E
FgQUMG+wUYMqe1fdVlFpTK8Ovi51cnUwHwYDVR0jBBgwFoAUMtAal9Q50bedoLEI
gI1d9MwGTGAwDQYJKoZIhvcNAQEFBQADgYEAUWC3oJ0EMDGQJx4x1JGbVcEEPl7F
lZSqV37q1OUufWZG/QAH/Q7f2mdhBTruKLKGMbPq3wV/88fhTevTWw8mWJSbgBtZ
t0uDtgRqA8y3u+RIdQYNEIuA2cXxGUZYMfwtFbAv34sOCHB7Pq51NvNub9GD5yjb
7AwlJHFrQoS/O7A=
-----END CERTIFICATE-----
```

3.4.3 Validity and Certificate Authority (CA)

Every server can issue its own certificate. Therefore, a server having a certificate does not make it a trusted server. To improve the validity of a certificate, a certificate is usually not issued by the server itself but by a certificate authority (CA). This CA will then sign the certificate by calculating the certificate hash with its own key. If the client then trusts the CA, and has its certificate, it can verify the validity of the server's certificate. However, usually the client will not have the CA's certificate, maybe from a root CA having authorised the CA. `CSR_TLS_CONFIG_VERIFY` can be used for loading the chain of CA certificates to verify the validity of the server certificate.

If the server has signed its own certificate, a so called self-signed certificate, the server certificate has to be used as CA when verifying the certificate.

When the connection sends a `CSR_TLS_CONNECTION_CERT_IND` it contains the status of the pre-verification and the certificate's validity. The CSR TLS task performs a pre-verification checking things as if the current date is within the validity period, and that the certificate is signed by a trusted CA certificate chain. If a CA certificate is not loaded via `CSR_TLS_CONFIG_VERIFY` it will always return not valid. It is then the task of the application to do the final verification, however preferably the pre-verification can be used as a guideline. Please note that the pre-verification should not be used blindly. E.g. is it not possible for the CSR TLS task to know if the server certificate really belongs to the server to which the connection is established. Only that the certificate is valid.

4 CSR TLS Primitives

This section introduces all the primitives and parameters used in the CSR TLS API. Detailed information can be found in the `csr_tls_prim.h` file.

Primitives	Reference
CSR_TLS_SESSION_CREATE_REQ	See Section 4.1
CSR_TLS_SESSION_CREATE_CFM	See Section 4.1
CSR_TLS_SESSION_DESTROY_REQ	See Section 4.2
CSR_TLS_SESSION_DESTROY_CFM	See Section 4.2
CSR_TLS_CONFIG_CIPHER_REQ	See Section 4.3
CSR_TLS_CONFIG_CIPHER_CFM	See Section 4.3
CSR_TLS_CONFIG_CERT_CLIENT_REQ	See Section 4.4
CSR_TLS_CONFIG_CERT_CLIENT_CFM	See Section 4.4
CSR_TLS_CONFIG_CERT_VERIFY_REQ	See Section 4.5
CSR_TLS_CONFIG_CERT_VERIFY_CFM	See Section 4.5
CSR_TLS_CONFIG_COMPRESSION_REQ	See Section 4.6
CSR_TLS_CONFIG_COMPRESSION_CFM	See Section 4.6
CSR_TLS_CONNECTION_CREATE_REQ	See Section 4.7
CSR_TLS_CONNECTION_CREATE_CFM	See Section 4.7
CSR_TLS_CONNECTION_DESTROY_REQ	See Section 4.8
CSR_TLS_CONNECTION_DESTROY_CFM	See Section 4.8
CSR_TLS_CONNECTION_BIND_REQ	See Section 4.9
CSR_TLS_CONNECTION_BIND_CFM	See Section 4.9
CSR_TLS_CONNECTION_CONNECT_REQ	See Section 4.10
CSR_TLS_CONNECTION_CONNECT_CFM	See Section 4.10
CSR_TLS_CONNECTION_DISCONNECT_REQ	See Section 4.11
CSR_TLS_CONNECTION_DISCONNECT_CFM	See Section 4.11
CSR_TLS_CONNECTION_DISCONNECT_RES	See Section 4.11
CSR_TLS_CONNECTION_DISCONNECT_IND	See Section 4.11
CSR_TLS_CONNECTION_CERT_IND	See Section 4.12
CSR_TLS_CONNECTION_CERT_RES	See Section 4.12
CSR_TLS_CONNECTION_DATA_REQ	See Section 4.13
CSR_TLS_CONNECTION_DATA_CFM	See Section 4.13
CSR_TLS_CONNECTION_DATA_IND	See Section 4.13
CSR_TLS_CONNECTION_DATA_RES	See Section 4.13

4.1 CSR_TLS_SESSION_CREATE

Primitives \ Parameters	type	qid	session	result
CSR_TLS_SESSION_CREATE_REQ	✓	✓		
CSR_TLS_SESSION_CREATE_CFM	✓		✓	✓

Table 1: CSR_TLS_SESSION_CREATE Primitives

Description

CSR_TLS_SESSION_CREATE_REQ creates a new TLS session and receives a handle for it. The session handles obtained using this primitive are required for establishing a secured connection.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_SESSION_CREATE_REQ/CFM
`CsrSchedQid`	`qid`	The identity of the calling task.
`CsrTlsSession`	`session`	A unique session handle that is used subsequent session communication
`CsrResult`	`results`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 2: Arguments to CSR_TLS_SESSION_CREATE Primitives

Porting Note

No note.

4.2 CSR_TLS_SESSION_DESTROY

Parameters / Primitives	type	session	result
CSR_TLS_SESSION_DESTROY_REQ	✓	✓	
CSR_TLS_SESSION_DESTROY_CFM	✓	✓	✓

Table 3: CSR_TLS_SESSION_DESTROY Primitives

Description

CSR_TLS_SESSION_DESTROY_REQ destroys the session. Any connection existing inside the session will be lost .

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_SESSION_DESTROY_REQ/CFM
`CsrTlsSession`	`session`	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
`CsrResult`	`results`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 4: Arguments to CSR_TLS_SESSION_DESTROY Primitives

Porting Note

No note.

4.3 CSR_TLS_CONFIG_CIPHER

Primitives \ Parameters	type	session	*cipherSuite	cipherSuiteLength	keybitsMin	keybitsMax	result
CSR_TLS_CONFIG_CIPHER_REQ	✓	✓	✓	✓	✓	✓	
CSR_TLS_CONFIG_CIPHER_CFM	✓	✓					✓

Table 5: CSR_TLS_CONFIG_CIPHER Primitives

Description

An optional primitive used for changing the default cipher suite configuration. To change the default configuration, a list of supported cipherSuites should be send. If the list contain a non-supported cipherSuite, the confirm result would be a failure. By default all supported cipher suites is supported except CSR_TLS_DH_anon_WITH_XXX which is very vulnerable for man-in-the-middle attacks and should only be used on explicit request from the application, and CSR_TLS_XXX_PSK_WITH_XXX which need a pre-shared key.

Parameters

Type	Argument	Description
CsrTlsPrim	type	CSR_TLS_CONFIG_CIPHER_REQ/CFM
CsrTlsSession	session	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
CsrUint16	*cipherSuite	CSR_TLS_NULL_WITH_NULL_NULL CSR_TLS_RSA_WITH_NULL_MD5 (*) CSR_TLS_RSA_WITH_NULL_SHA (*) CSR_TLS_RSA_WITH_NULL_SHA256 CSR_TLS_RSA_WITH_RC4_128_MD5 (*) CSR_TLS_RSA_WITH_RC4_128_SHA (*) CSR_TLS_RSA_WITH_3DES_EDE_CBC_SHA CSR_TLS_RSA_WITH_AES_128_CBC_SHA (*) CSR_TLS_RSA_WITH_AES_128_CBC_SHA256 CSR_TLS_RSA_WITH_AES_256_CBC_SHA (*) CSR_TLS_RSA_WITH_AES_256_CBC_SHA256 Continuing on next page:

		Continued from last page: CSR_TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA CSR_TLS_DH_DSS_WITH_AES_128_CBC_SHA CSR_TLS_DH_DSS_WITH_AES_128_CBC_SHA256 CSR_TLS_DH_DSS_WITH_AES_256_CBC_SHA CSR_TLS_DH_DSS_WITH_AES_256_CBC_SHA256 CSR_TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA CSR_TLS_DH_RSA_WITH_AES_128_CBC_SHA CSR_TLS_DH_RSA_WITH_AES_128_CBC_SHA256 CSR_TLS_DH_RSA_WITH_AES_256_CBC_SHA CSR_TLS_DH_RSA_WITH_AES_256_CBC_SHA256 CSR_TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (*) CSR_TLS_DHE_DSS_WITH_AES_128_CBC_SHA (*) CSR_TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 CSR_TLS_DHE_DSS_WITH_AES_256_CBC_SHA (*) CSR_TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 CSR_TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (*) CSR_TLS_DHE_RSA_WITH_AES_128_CBC_SHA (*) CSR_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 CSR_TLS_DHE_RSA_WITH_AES_256_CBC_SHA (*) CSR_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 CSR_TLS_DH_anon_WITH_3DES_EDE_CBC_SHA (*) CSR_TLS_DH_anon_WITH_AES_128_CBC_SHA (*) CSR_TLS_DH_anon_WITH_AES_128_CBC_SHA256 CSR_TLS_DH_anon_WITH_AES_256_CBC_SHA (*) CSR_TLS_DH_anon_WITH_AES_256_CBC_SHA256 CSR_TLS_DH_anon_WITH_RC4_128_MD5 (*) CSR_TLS_PSK_WITH_RC4_128_SHA (*) CSR_TLS_PSK_WITH_3DES_EDE_CBC_SHA (*) CSR_TLS_PSK_WITH_AES_128_CBC_SHA (*) CSR_TLS_PSK_WITH_AES_256_CBC_SHA (*) CSR_TLS_DHE_PSK_WITH_RC4_128_SHA CSR_TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA CSR_TLS_DHE_PSK_WITH_AES_128_CBC_SHA CSR_TLS_DHE_PSK_WITH_AES_256_CBC_SHA CSR_TLS_RSA_PSK_WITH_RC4_128_SHA CSR_TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA CSR_TLS_RSA_PSK_WITH_AES_128_CBC_SHA CSR_TLS_RSA_PSK_WITH_AES_256_CBC_SHA
CsrUint16	cipherSuiteLength	Length of buffer
CsrUint16	keybitsMin	Reserved for future update
CsrUint16	keybitsMax	Reserved for future update
CsrResult	result	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 6: Arguments to CSR_TLS_CIPHER_CONFIG Primitives

Porting Note

Only the cipherSuite marked with (*) is available in the Linux port due to the limited support of ciphers in OpenSSL.

Keybits Min/Max are currently not considered in the Linux port.

4.4 CSR_TLS_CONFIG_CERT_CLIENT

Primitives \ Parameters	type	session	certificateType	*certificate	certificateLength	*password	passwordLength	result
CSR_TLS_CONFIG_CERT_CLIENT_REQ	✓	✓	✓	✓	✓	✓	✓	
CSR_TLS_CONFIG_CERT_CLIENT_CFM	✓	✓						✓

Table 7: CSR_TLS_CONFIG_CERT_CLIENT Primitives

Description

An optional primitive used for setting the client certificate. It can be used for both certificates, pre-shared keys (PSK) and eventual a private key if not included in certificate. If this primitive is not sent, the client will terminate the connection if the server requests a certificate/PSK. Currently, only one certificate or PSK per session is supported. If certificate does not include the private key, the CSR_TLS_PRIVATE_KEY has to be sent after CSR_TLS_CERTIFICATE if it is used. Certificate needs to be formatted in the X.509 v3 format. The password protecting the private key needs to be sent with the certificate containing the private key. If the private key is not protected, the password needs to be set to NULL and the passwordLength to 0.

Parameters

Type	Argument	Description
CsrTlsPrim	type	CSR_TLS_CERT_CLIENT_REQ/CFM
CsrTlsSession	session	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
CsrUint8	certificateType	CSR_TLS_CERTIFICATE CSR_TLS_PRIVATE_KEY CSR_TLS_PSK
CsrUint8	*certificate	Containing either a certificate or a PSK. The certificate in X.509 v3 format
CsrSize	certificateLength	Length of buffer
CsrUint8	*password	Password protecting the private key. NULL if the private key is not protected
CsrSize	passwordLength	Length of the password. Set to 0 if no private key or if not password protected.
CsrResult	result	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 8: Arguments to CSR_TLS_CONFIG_CERT_CLIENT Primitives

Porting Note

If a PSK is used it will be stored and used by the function handling the openSSL call back, however, if it is chosen by the application not to use the standard implemented call back function by defining the CSR_TLS_EXTERNAL_PSK_CLIENT_CALLBACK_FUNCTION, then there is no need for the application to provide the PSK to the CSR-TSL.

4.5 CSR_TLS_CONFIG_CERT_VERIFY

Parameters / Primitives	type	session	*trustedCAcertificate	trustedCAcertificateLength	result
CSR_TLS_CONFIG_CERT_VERIFY _REQ	✓	✓	✓	✓	
CSR_TLS_CONFIG_CERT_VERIFY _CFM	✓	✓			✓

Table 9: CSR_TLS_CONFIG_CERT_VERIFY Primitives

Description

An optional primitive used for configuring under which criteria the TLS task should verify the validity of the certificate. This signal contains a certificate of a trusted certificate authority (CA), which can be used for verifying the correctness of the server certificate. If more CA's are in the verify chain, more certificates can be included in the same buffer. The certificate needs to be formatted in the X.509 v3 format encoded as PEM.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`Type`	CSR_TLS_CERT_CLIENT_REQ/CFM
`CsrTlsSession`	`Session`	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
`CsrUint8`	`*trustedCAcertificate`	A buffer containing a certificate of a trusted Certificate Authority (CA). The certificate is in X.509 v3 format encoded as a PEM string
`CsrSize`	`trustedCAcertificateLength`	Length of buffer in bytes
`CsrResult`	`Result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 10: Arguments to CSR_TLS_CONFIG_CERT_VERIFY Primitives

Porting Note

No note.

4.6 CSR_TLS_CONFIG_COMPRESSION

Primitives \ Parameters	type	session	compression	result
CSR_TLS_CONFIG_COMPRESSION_REQ	✓	✓	✓	
CSR_TLS_CONFIG_COMPRESSION_CFM	✓	✓		✓

Table 11: CSR_TLS_CONFIG_COMPRESSION Primitives

Description

An optional primitive used for setting the compression level. If not set, no compression will be supported. No compression is supported by the current generic version of CSR TLS, and the primitive is only reserved for porting and future extension.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONFIG_COMPRESSION_REQ/CFM
`CsrTlsSession`	`session`	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
`CsrUint8`	`compression`	Value of compression. 0 indicate no compression
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 12: Arguments to CSR_TLS_CONFIG_COMPRESSION Primitives

Porting Note

No note.

4.7 CSR_TLS_CONNECTION_CREATE

Primitives \ Parameters	type	session	socketFamily	conn	result
CSR_TLS_CONNECTION_CREATE_REQ	✓	✓	✓		
CSR_TLS_CONNECTION_CREATE_CFM	✓	✓		✓	✓

Table 13: CSR_TLS_CONNECTION_CREATE Primitives

Description

CSR_TLS_CONNECTION_CREATE_REQ tries to create a new connection in an existing session, having the session handle, `session`. A CSR_TLS_CONNECTION_CREATE_CFM is sent by CSR TLS with the result of the creation attempt and the connection handle, `conn`. `conn` is the identifier for the connection for all further operations on this connection. The `socketFamily` value is used to request either an IPv4 or an IPv6 based connection.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_CREATE_REQ/CFM
`CsrTlsSession`	`session`	A unique handle for the session obtained from CSR_TLS_SESSION_CREATE
`CsrIpSocketFamily`	`socketFamily`	CSR_IP_SOCKET_FAMILY_IP4 or CSR_IP_SOCKET_FAMILY_IP6
`CsrTlsConnection`	`conn`	A unique connection id handle
`CsrResult`	`result`	CSR_RESULT_SUCCESS, CSR_TLS_RESULT_IP6_NOT_SUPPORTED, or CSR_RESULT_FAILURE

Table 14: Arguments to CSR_TLS_CONNECTION_CREATE Primitives

Porting Note

No note.

4.8 CSR_TLS_CONNECTION_DESTROY

Primitives \ Parameters	type	conn	result
CSR_TLS_CONNECTION_DESTROY_REQ	✓	✓	
CSR_TLS_CONNECTION_DESTROY_CFM	✓	✓	✓

Table 15: CSR_TLS_CONNECTION_DESTROY Primitives

Description

CSR_TLS_CONNECTION_DESTROY_REQ requests the connection to terminate.
CSR_TLS_CONNECTION_DESTROY_CFM contains the result of the request. The connection cannot be reopened when it is destroyed.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_DESTROY_REQ/CFM
`CsrTlsConnection`	`conn`	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 16: Arguments to CSR_TLS_CONNECTION_DESTROY Primitives

Porting Note

No note.

4.9 CSR_TLS_CONNECTION_BIND

Parameters / Primitives	type	conn	ipAddress	port	result
CSR_TLS_CONNECTION_BIND_REQ	✓	✓	✓	✓	
CSR_TLS_CONNECTION_BIND_CFM	✓	✓			✓

Table 17: CSR_TLS_CONNECTION_BIND Primitives

Description

CSR_TLS_CONNECTION_BIND_REQ tries to bind an IP address and a port to a TLS connection. CSR_TLS_CONNECTION_BIND_CFM sends the result of the request. A bind needs to be performed before connecting on the connection, and is persistent even if the connection is disconnected and reopened. Please note that this is an optional command.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_BIND_REQ/CFM
`CsrTlsConnection`	`conn`	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
`CsrUint8`	`ipAddress[16]`	IP address to bind to. The address is stored in the same order as in the IP packet header. For IPv4 address, only the first four bytes are used.
`CsrUint16`	`port`	Port to bind to
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 18: Arguments to CSR_TLS_CONNECTION_BIND Primitives

Porting Note

No note.

4.10 CSR_TLS_CONNECTION_CONNECT

Primitives \ Parameters	type	conn	ipAddress	port	result
CSR_TLS_CONNECTION_CONNECT_REQ	✓	✓	✓	✓	
CSR_TLS_CONNECTION_CONNECT_CFM	✓	✓			✓

Table 19: CSR_TLS_CONNECTION_CONNECT Primitives

Description

CSR_TLS_CONNECTION_CONNECT_REQ can be sent by the application to try to connect the connection to a specified ip address, ipAddress, and port, port. CSR TLS will then negotiate the properties for the secure connection when the connection is ready for secure connection, or if the connection fails, CSR TLS will send a CSR_TLS_CONNECTION_CONNECT_CFM with the result of the request. A failure on the connect will invalidate the connection, and the application needs to destroy the connection.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_CONNECT_REQ/CFM
`CsrTlsConnection`	`conn`	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
`CsrUint8`	`ipAddress[16]`	Remote server IP address. The address is stored in the same order as in the IP packet header. For IPv4 address, only the first four bytes are used.
`CsrUint16`	`port`	Remote server port.
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 20: Arguments to CSR_TLS_CONNECTION_CONNECT Primitives

Porting Note

No note.

4.11 CSR_TLS_CONNECTION_DISCONNECT

Primitives \ Parameters	type	conn	result
CSR_TLS_CONNECTION_DISCONNECT_REQ	✓	✓	
CSR_TLS_CONNECTION_DISCONNECT_CFM	✓	✓	✓
CSR_TLS_CONNECTION_DISCONNECT_IND	✓	✓	
CSR_TLS_CONNECTION_DISCONNECT_RES	✓	✓	

Table 21: CSR_TLS_CONNECTION_DISCONNECT Primitives

Description

CSR_TLS_CONNECTION_DISCONNECT_REQ requests a disconnection of a connection, conn. When the TLS task has tried to disconnect the connection a CSR_TLS_CONNECTION_DISCONNECT_CFM is sent with the result. The disconnected connection can later be reconnected using a new CSR_TLS_CONNECTION_CONNECT. Since incoming traffic still can occur until a confirm is received, the application needs to be ready to handle indications after requesting a disconnect

Similarly a disconnect can be performed by server side. In this case a CSR_TLS_CONNECTION_DISCONNECT_IND is sent to the application to indicate that the connection is disconnected. The application needs to send a CSR_TLS_CONNECTION_DISCONNECT_RES response before the connection is effectively disconnected.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_DISCONNECT_REQ/CFM/IND/RES
`CsrTlsConnection`	`conn`	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 22: Arguments to CSR_TLS_CONNECTION_DISCONNECT Primitives

Porting Note

No note.

4.12 CSR_TLS_CONNECTION_CERT

Primitives \ Parameters	type	conn	*cert	certLen	result	accepted
CSR_TLS_CONNECTION_CERT_IND	✓	✓	✓	✓	✓	
CSR_TLS_CONNECTION_CERT_RES	✓	✓				✓

Table 23: CSR_TLS_CONNECTION_CERT Primitives

Description

CSR_TLS_CONNECTION_CERT_IND indicates that the server has sent its certificate. The TLS task pre-verify the server certificate, and the result is stored in the results field. If the pre-verify could verify the server certificate it will return CSR_RESULT_SUCCESS. Else it will indicate what went wrong. For a last verification, the server certificate is sent to the application. The certificate is located in the buffer, cert. The application must now send a CSR_TLS_CONNECTION_CERT_RES with an accept/reject. If the certificate is rejected, the application has to disconnect and destroy the connection as well as destroy the session since they are all invalid.

Parameters

Type	Argument	Description
CsrTlsPrim	type	CSR_TLS_CONNECTION_CERT_IND/RES
CsrTlsConnection	conn	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
CsrUint8	*cert	The certificate in x509v3 format encoded as base64 PEM
CsrSize	certLen	Certificate length in bytes
CsrResult	result	Result of the pre-verification done by CSR TLS. Results can be: CSR_RESULT_SUCCESS CSR_TLS_CERT_NOT_YET_VALID CSR_TLS_CERT_HAS_EXPIRED CSR_TLS_CERT_SELFSIGNED CSR_TLS_CERT_MISSING_ISSUER_CERT CSR_TLS_CERT_UNABLE_TO_VERIFY_NO_SELFSIGNED_CERT CSR_TLS_CERT_CA_INVALID CSR_RESULT_FAILURE
CsrBool	accepted	Indicate of the certificate is accepted or not

Table 24: Arguments to CSR_TLS_CERT Primitives

Porting Note

No note.

4.13 CSR_TLS_CONNECTION_DATA

Parameters / Primitives	type	conn	bufLen	*buf	result
CSR_TLS_CONNECTION_DATA_REQ	✓	✓	✓	✓	
CSR_TLS_CONNECTION_DATA_CFM	✓	✓			✓
CSR_TLS_CONNECTION_DATA_IND	✓	✓	✓	✓	
CSR_TLS_CONNECTION_DATA_RES	✓	✓			

Table 25: CSR_TLS_CONNECTION_DATA Primitives

Description

CSR_TLS_CONNECTION_DATA_REQ encrypts and transmits data using the secure connection conn. CSR_TLS_CONNECTION_DATA_ CFM confirms that CSR TLS has encrypted the data and passed it to the local TCP/IP stack that will transmit the data. When a CSR_TLS_CONNECTION_DATA_ CFM is sent by the TLS task, it is ready to receive the next chunk of data for the connection conn.

CSR_TLS_CONNECTION_DATA_IND indicates that data has been received, decrypted, and is ready to be read by the application. The payload data is located in the buffer, buf, with the length, buffLen. When the application have read the data and is ready to receive new data it has to signal a CSR_TLS_CONNECTION_DATA_RES for confirming the reception of the data.

Parameters

Type	Argument	Description
`CsrTlsPrim`	`type`	CSR_TLS_CONNECTION_DATA_REQ/CFM/IND/RES
`CsrTlsConnection`	`conn`	A unique connection id handle obtained by CSR_TLS_CONNECTION_CREATE
`CsrUint8`	`*buf`	Pointer to the buffer which contains the payload to be sent
`CsrSize`	`bufLen`	Length of the payload
`CsrResult`	`result`	CSR_RESULT_SUCCESS or CSR_RESULT_FAILURE

Table 26: Arguments to CSR_TLS_CONNECTION_DATA Primitives

Porting Note

No note.

5 Document References

Document	Reference

Terms and Definitions

CSR	Cambridge Silicon Radio
DNS	Domain Name System
IP	Internet Protocol
MSC	Message Sequence Chart
TCP	Transmission Control Protocol

Table 27: Abbreviations and Definitions

Document History

Revision	Date	History
1	JUNE 1st 2010	Initial revision
2	AUGUST 20th 2010	Release version
3	OCT 2010	Ready for release 2.2.0
4	DEC 10	Ready for release 3.0.0
5	FEB 2011	Ready for release 3.0.1
6	Aug 2011	Ready for release 3.1.0

TradeMarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc or its affiliates. Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR. Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

No statements or representations in this document are to be construed as advertising, marketing, or offering for sale in the United States imported covered products subject to the Cease and Desist Order issued by the U.S. International Trade Commission in its Investigation No. 337-TA-602. Such products include SiRFstarIII™ chips that operate with SiRF software that supports SiRFInstantFix™, and/or SiRFLoc® servers, or contains SyncFreeNav functionality.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information

Appendix A TLS Handshake between Client and Server

The following illustrates the handshake protocol for establishing a TLS connection between client and server. This will be done internally in the CSR TLS task and the user does not need to consider this.

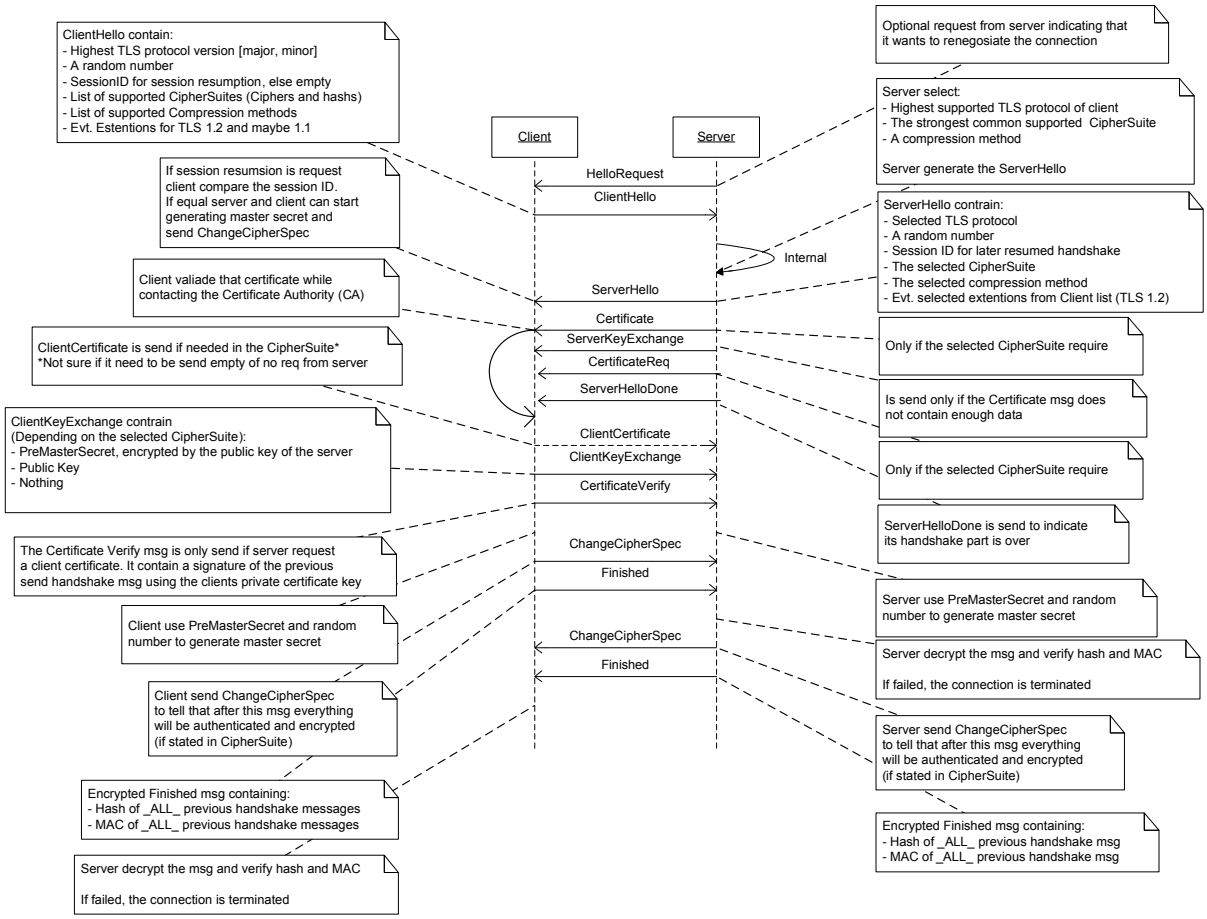


Figure 7: MSC for internal handshake when establishing a CSR TLS session