Contents

[1 Introduction 2](#_Toc506220643)

[1.1 How to Read this Document 2](#_Toc506220644)

[1.2 Scope 2](#_Toc506220645)

[1.3 References 2](#_Toc506220646)

[1.4 Abbreviations 2](#_Toc506220647)

[1.5 Terminology 2](#_Toc506220648)

[1.6 System Requirements 2](#_Toc506220649)

[2 Protocol Modules 3](#_Toc506220650)

[2.1 Overview 3](#_Toc506220651)

[2.2 Installation 3](#_Toc506220652)

[2.3 Configuration 3](#_Toc506220653)

[3 Functional specification 3](#_Toc506220654)

[3.1 Supported TLS versions 3](#_Toc506220655)

[3.2 TLS object handling function 3](#_Toc506220656)

[3.3 TLS descriptor 4](#_Toc506220657)

[3.4 Connection establishment 5](#_Toc506220658)

[3.5 Data exchange function 6](#_Toc506220659)

[3.5.1 Send data to the peer 6](#_Toc506220660)

[3.5.2 Receive data from the peer 7](#_Toc506220661)

# Introduction

## How to Read this Document

This is the Function Specification for the TLS library. The TLS library are developed for the TTCN-3 Toolset with TITAN. This document should be read together with Product Revision Information [3].

## Scope

The purpose of this document is to specify the content of the TLS library. The document is primarily addressed to the end users of the product. Basic knowledge of TTCN-3 [2] and TITAN TTCN-3 Test Executor [4] is valuable when reading this document.

## References

1. OpenSSL  
   https://www.openssl.org/
2. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3; Part 1: Core Language
3. 109 21-CNL 113 839-2 Uen  
   TLS library for TTCN-3 Toolset with TITAN, Product Revision Information

1. [1/198 17-CRL 113 200/6](http://ttcn.ericsson.se/download/doc/userguide.pdf) Uen  
   User Guide for the TITAN TTCN-3 Test Executor

## Abbreviations

DTLS Datagram Transport Layer Security

TLS Transport Layer Secuity

TTCN-3 Testing and Test Control Notation version 3

SSL Secure Socket Layer

## Terminology

No specific terminology is used.

## System Requirements

The TLS library is a set of TTCN-3 source code files that can be used as part of TTCN-3 test suites only. Hence, TLS library alone do not put specific requirements on the system used. However, in order to compile and execute a TTCN-3 test suite using the set of protocol modules the following system requirements must be satisfied:

* TITAN TTCN-3 Test Executor version CRL 113 200/6 R3A (6.3.pl0) or higher installed. Please note: This version of the protocol module is not compatible with TITAN releases earlier than CRL 113 200/6 R3A. For installation guide see [2].
* OpenSSL development library is installed. See [1]

# Protocol Modules

## Overview

The TLS library implement the control functions in a formalized way. This allows defining of test data (templates) in the TTCN-3 language [2] and correct encoding/decoding of messages when executing test suites using the TITAN TTCN-3 test environment.

## Installation

The set of protocol modules can be used in developing TTCN-3 test suites using any text editor. However, to make the work more efficient a TTCN-3-enabled text editor is recommended (e.g. nedit, xemacs). Since the TLS library is used as a part of a TTCN-3 test suite, this requires TTCN-3 Test Executor be installed before the module can be compiled and executed together with other parts of the test suite. For more details on the installation of TTCN-3 Test Executor see the relevant section of [4].

## Configuration

None.

# Functional specification

## Supported TLS versions

The TLS library uses the OpenSSL, hence the supported SSL/TLS/DTLS versions are depends on the used version of the OpenSSL. The TLS library provides interface for:

* SSLv3
* TLSv1, TLSv1.1, TLSv1.2
* DTLSv1.0, DTLSv1.2

## TLS object handling function

In order to establish TLS/DTLS connections a TLS object should be created.

external function TLS\_New\_object(

in TLS\_descriptor descr

out integer object\_id,

in integer user\_idx := -1 ) return TLS\_op\_result;

The function returns the result of the object creation. The object can be referenced later by the object\_id.

The optional user\_idx is not used by the TLS library directly, just stored as is. It can be used a reference to another object or connection.

After the TLS/DTLS connection is not used any more the object should be destroyed.

external function TLS\_Delete\_object(in integer object\_id) return TLS\_op\_result;

## TLS descriptor

The TLS descriptor has the following fields:

* tls\_method: type: TLS\_method, mandatory  
  Default value: TLS\_method\_TLS  
  Allowed values: TLS\_method\_TLS, TLS\_method\_DTLS  
  It specifies which method to use: TLS/SSL or DTLS.
* min\_supported\_version, max\_supported\_version  
  Type: TLS\_Supported\_proto\_versions, optional  
  Default value: omit  
  Allowed values: TLS\_SSL3\_VERSION, TLS\_TLS1\_VERSION, TLS\_TLS1\_1\_VERSION, TLS\_TLS1\_2\_VERSION, TLS\_DTLS1\_VERSION, TLS\_DTLS1\_2\_VERSION  
  The minimum and the maximum version of the supported protocol.
* ssl\_key\_file: type: charstring, optional  
  Default value: omit  
  It specifies a PEM encoded file’s path on the file system containing the server’s RSA private key.
* ssl\_certificate\_file: type: charstring, optional  
  Default value: omit  
  It specifies a PEM encoded file’s path on the file system containing the certificate chain.
* ssl\_trustedCAlist\_file: type: charstring, optional  
  Default value: omit  
  It specifies a PEM encoded file’s path on the file system containing the certificates of the trusted CA authorities to use.
* ssl\_cipher\_list: type: charstring, optional  
  Default value: omit  
  It can be used to specify the allowed cipher list.
* ssl\_password: type: charstring, optional  
  Default value: omit  
  It specifies the password protecting the private key file.
* ssl\_verify\_certificate: type: boolean, optional  
  Default value: true  
  It can be used to specify whether to check the certificate of the other side.
* psk\_hint: type: charstring, optional  
  Default value: omit  
  The server can provide a "PSK identity hint" in the ServerKeyExchange message. In the case where PSK identity hint is omit, the server does not send the ServerKeyExchange message to the client.
* psk\_identity: type: charstring, optional  
  Default value: omit  
  The “PSK identity” is included in the ClientKeyExchange message and transmitted to the server. After the negotiation for “PSK identity” is done, the client and the server can generate their pre-master secrets with the pre-shared key.
* psk\_key: type: charstring, optional  
  Default value: omit  
  It is the psk key in hexadecimal representation form.
* psk\_for\_server: type: boolean, optional  
  Default value: omit  
  It can be used to specify the used psk callback function, the psk server callback or the psk client callback function will be called.

## Connection establishment

The function to use to establish connection:

external function TLS\_Handshake(in integer object\_id,

in boolean is\_server

in octetstring input\_stream,

out octetstring output\_stream ) return TLS\_op\_result;

The function should be called repeatedly until the handshake is finished.

The return values:

* TLS\_OK: handshake finished
* TLS\_NEED\_MORE\_DATA: send the output stream to the peer if len >0, call the function again with more data from peer
* TLS\_DATA\_TO\_SEND: send the output stream to the peer, and call the function again
* TLS\_ERROR: something went wrong.

Parameters:

* object\_id: The TLS object, created by the TLS\_New\_object
* is\_server: true: server side, false: client side
* input\_stream: The data received from the peer, or empty string
* output\_stream: The function places the data should be sent to the peer.

## Data exchange function

### Send data to the peer

The following function can be used to send data to the peer:

external function TLS\_Write(in integer object\_id,

in octetstring user\_data

in octetstring input\_stream,

out octetstring output\_stream ) return TLS\_op\_result;

Should be called repeatedly until returns TLS\_OK.

The return values:

* TLS\_OK: all data ready to be sent.
* TLS\_NEED\_MORE\_DATA: send the output stream to the peer if len >0, call the function again with more data from peer
* TLS\_DATA\_TO\_SEND: send the output stream to the peer, and call the function again
* TLS\_ERROR: something went wrong.

Parameters:

* object\_id: The TLS object, created by the TLS\_New\_object
* user\_data: the data to be sent
* input\_stream: The data received from the peer, or empty string
* output\_stream: The function places the data should be sent to the peer.

### Receive data from the peer

The following function can be used to decrypt the data received from the peer:

external function TLS\_Read(in integer object\_id,

out octetstring user\_data,

in octetstring input\_stream,

out octetstring output\_stream

) return TLS\_op\_result;

Should be called repeatedly until returns TLS\_OK.

The return values:

* TLS\_OK: all data received.
* TLS\_NEED\_MORE\_DATA: send the output stream to the peer if len >0, call the function again with more data from peer
* TLS\_DATA\_TO\_SEND: send the output stream to the peer, and call the function again
* TLS\_ERROR: something went wrong.

Parameters:

* object\_id: The TLS object, created by the TLS\_New\_object
* user\_data: the received data
* input\_stream: The data received from the peer, or empty string
* output\_stream: The function places the data should be sent to the peer.