|  |  |
| --- | --- |
|  |  |

EPTF Core Library Transport, Function Description

Contents

[1 Introduction 3](#_Toc500163239)

[1.1 Revision history 3](#_Toc500163240)

[1.2 How to read this document 3](#_Toc500163241)

[1.3 Scope 3](#_Toc500163242)

[1.4 Recommended way of reading 3](#_Toc500163243)

[1.5 Typographical conventions 4](#_Toc500163244)

[1.6 Abbreviations 4](#_Toc500163245)

[1.7 Terminology 4](#_Toc500163246)

[2 General description 4](#_Toc500163247)

[2.1 Transport types 4](#_Toc500163248)

[2.1.1 EPTF\_Transport\_CT 5](#_Toc500163249)

[2.1.2 EPTF\_TransportIPL4\_CT 5](#_Toc500163250)

[2.1.3 EPTF\_TransportIPL2\_CT 5](#_Toc500163251)

[3 Functional interface 6](#_Toc500163252)

[3.1 Naming conventions 6](#_Toc500163253)

[3.2 The routing component 6](#_Toc500163254)

[3.2.1 Initialization 6](#_Toc500163255)

[3.2.2 Process Incoming Message 7](#_Toc500163256)

[3.2.3 Process Outgoing Message 7](#_Toc500163257)

[3.2.4 Summary table of all public functions for EPTF transport routing 7](#_Toc500163258)

[3.3 The MessageBufferManager component 7](#_Toc500163259)

[3.3.1 Initialization 7](#_Toc500163260)

[3.3.2 CleanUp 7](#_Toc500163261)

[3.3.3 Set message to buffer 7](#_Toc500163262)

[3.3.4 Get message from buffer 8](#_Toc500163263)

[3.3.5 Summary table of all public functions for EPTF transport routing 8](#_Toc500163264)

[3.4 Transport components 9](#_Toc500163265)

[3.4.1 Initialization 9](#_Toc500163266)

[3.4.2 Registering callback to calculate the message length 9](#_Toc500163267)

[3.4.3 Set the message and event handler functions 9](#_Toc500163268)

[3.4.4 Connect 10](#_Toc500163269)

[3.4.5 Listen 10](#_Toc500163270)

[3.4.6 Send a message 11](#_Toc500163271)

[3.4.7 Send a message to a remote peer 11](#_Toc500163272)

[3.4.8 Close connections 11](#_Toc500163273)

[3.4.9 Setup interfaces 12](#_Toc500163274)

[3.4.10 Append interfaces 12](#_Toc500163275)

[3.4.11 Remove interfaces 12](#_Toc500163276)

[3.4.12 Activate default buffering handler 12](#_Toc500163277)

[3.4.13 Summary table of all public functions for EPTF Transport IPL2 Communication Port, EPTF\_CLL\_Transport and the IPL4 Communication Port new API 13](#_Toc500163278)

[3.5 The IPL4 Communication Port component with the obsolete API 14](#_Toc500163279)

[3.5.1 Initialization 14](#_Toc500163280)

[3.5.2 Set getMessageLength function 14](#_Toc500163281)

[3.5.3 Set receive template and handler function 15](#_Toc500163282)

[3.5.4 The send function 15](#_Toc500163283)

[3.5.5 The activate handler function 15](#_Toc500163284)

[3.5.6 Default connection result event handler 15](#_Toc500163285)

[3.5.7 Summary table of all public functions for EPTF Transport CommPortIPL4 legacy API 16](#_Toc500163286)

[3.6 IPL2 support of multiple interfaces 16](#_Toc500163287)

[4 IPsec Support 17](#_Toc500163288)

[4.1 Initialization 17](#_Toc500163289)

[4.2 Checking initialization 18](#_Toc500163290)

[4.3 Allocating SPI 18](#_Toc500163291)

[4.4 Creating Security Policy 18](#_Toc500163292)

[4.5 Creating Security Association 19](#_Toc500163293)

[4.6 Deleting Security Policy 19](#_Toc500163294)

[4.7 Deleting Security Association 19](#_Toc500163295)

[4.8 Deleting all SA/SP 20](#_Toc500163296)

[4.9 IPsec Logging 20](#_Toc500163297)

[4.10 Summary table of all public functions for EPTF transport IPsec Handler 21](#_Toc500163298)

[5 DTE handling 22](#_Toc500163299)

[6 References 22](#_Toc500163300)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Rev** | **Characteristics** | **Prepared** |
| 2008-01-28 | PA1 | First draft version | ENORPIN |
| 2010-05-31 | PB1 | IPL4 common API, IPL2 API added | EJNOSVN |
| 2010-06-24 | PC1 | Warning message if Buffer is full | EJNOSVN |
| 2010-09-03 | PC1 | Multiple interface support for IPL2 | EGBOTAT |
| 2011-11-25 | PD1 | Passing protocol info to IPL4 TP is complemented | EBALLUG |
| 2012-02-27 | PE1 | Multi-homing support | EBALLUG |
| 2014-10-08 | PF1 | Added section DTE handling | EDORNAG |
| 2014-11-21 | PF2 | Added comment on DTE handling. | ESZILSZ |
| 2014-12-02 | F | Updated for release | ESZILSZ |
| 2016-10-14 | PG1 | Added new section | EVAURBA |
| 2016-10-15 | PG2 | Updated revision history | EVAURBA |
| 2016-10-17 | PG3 | Editorial changes. | EIMRENA |
| 2016-10-17 | PG4 | Corrected revision | EIMRENA |
| 2016-10-17 | PG5 | Editorial changes. | EIMRENA |
| 2016-12-02 | G | Updated for release | EIMRENA |
| 2017-01-26 | PH1 | Added IPsec support | EIMRENA |
| 2017-06-09 | H | Updated for release | EIMRENA |
| 2017-11-09 | PJ1 | Changes in IPsec handling | ZNAGIMR |
| 2017-11-09 | PJ2 | Editorial fixes | ZNAGIMR |
| 2017-12-01 | J | Updated for release | ZLENZOL |

## How to read this document

This is the Function Description for the EPTF Transport of the Ericsson Performance Test Framework (EPTF), Core Library (CLL). EPTF Core Library is developed for the TTCN-3 [1] Toolset with TITAN [2]. For more information on the EPTF Core Library, consult the Product Revision Information [3].

## Scope

This document is to specify the content and functionality of the EPTF Transport feature of the EPTF Core Library.

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of EPTF Core Library [4]. They should get familiar with the list of acronyms and the glossary in Section 1.6 and 1.7, respectively.

## Typographical conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context. Moreover, whenever a concept is mentioned that has a special meaning as described in the Glossary (Section 1.7) of this document, then these occurrences are marked with an initial arrow, for example 🡪 *TitanSim Statistics*.

## Abbreviations

CLL Core Library

EPTF Ericsson Performance Test Framework

TitanSim New synonym for the EPTF Framework

TTCN-3 Testing and Test Control Notation version 3 [1]

## Terminology

*Core Library (CLL)* is that part of the TitanSim software that is totally project independent. (i.e., which is not protocol-, or application-dependent). The Core Library is to be supplied and supported by the TCC organization. Any Core Library development is to be funded centrally by Ericsson.

# General description

This document specifies the Transport control features of the EPTF Core Library.

The EPTF Transport feature makes it possible to:

* Route the incoming and outgoing messages between the components
* Store messages and information in a FreeBusyQueue
* Provides unified interface for different testports

## Transport types

There are component types which implement the unified interface for the different testports.

To be able to use these features, the user component should extend the right EPTF Transport component(s).

### EPTF\_Transport\_CT

This component type is testport-independent. The used testport can be specified runtime by socket.

### EPTF\_TransportIPL4\_CT

This component type uses the IPL4asp testport. It is kernel-based to make the solution simple.

Figure 1 shows how the functionalities are distributed using the IPL4 transport type.

IPL4 common transport

TCP

UDP

IP

SUT

Socket database

Application / AppLib

SCTP

SSL

IPL4 test port

Kernel

Figure 1 Distribution of functionalities using IPL4 common transport

### EPTF\_TransportIPL2\_CT

The IPL2 transport type moves the socket handling into the user space in order to make it possible to optimize the socket handling independently from the kernel.

Figure 2 shows how the functionalities are distributed using the IPL2 transport type.

TCP

SCTP

SSL

LANL2 test port

SUT

Application / AppLib

UDP

IP

Socket database

IPL2 common transport

Kernel Packet Socket

Figure 2 Distribution of functionalities using IPL2 common transport

# Functional interface

Apart from this description a cross-linked reference guide for the EPTF Core Library Functions can be reached for on-line reading [5].

## Naming conventions

All functions have the prefix f\_EPTF\_<<Feature Name>>\_.

## The routing component

### Initialization

The user has to only initialize the component with the following function:

f\_EPTF\_Transport\_Routing\_init\_CT (  
in f\_EPTF\_Routing\_processMessage\_FT pl\_processOutgoingMsg   
in f\_EPTF\_Routing\_processMessage\_FT pl\_processIncomingMsg )

The parameters of the routing init function are the call-back functions, which routing the incoming and outgoing messages.

### Process Incoming Message

The user can use with this function the registered Process Incoming Message function.

### Process Outgoing Message

The user can use with this function the registered Process Outgoing Message function.

### Summary table of all public functions for EPTF transport routing

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_Routing\_init\_CT | Initializes the Routing component |
| f\_EPTF\_Routing\_processIncomingMessage | Calls the registered incoming message routing call-back function |
| f\_EPTF\_Routing\_processOutgoingMessage | Calls the registered outgoing message routing call-back function |

## The MessageBufferManager component

### Initialization

The user has to only initialize the component with the following function:

f\_EPTF\_MessageBufferManager\_init\_CT ()

### CleanUp

The user has to use the following function to cleanup this component:

f\_EPTF\_MessageBufferManager\_cleanup\_CT function

### Set message to buffer

The user can set with the following function a message to the buffer with the given key:

f\_EPTF\_MessageBufferManager\_setMessageToBuffer(  
in octetstring pl\_message,  
in charstring pl\_information,  
in integer pl\_key)

The first parameter of the functions is the message, the second parameter is a charstring, which can contains some other information to the message (for example routing information) and the last parameter is the key value. If the buffer reaches the limit, it clears itself – deletes all previous messages but stores the new message – and warns the user about this.

### Get message from buffer

The user can get a message from the buffer with the following function:

f\_EPTF\_MessageBufferManager\_getMessageToBuffer(  
in integer pl\_key,  
out octetstring pl\_message,  
out charstring pl\_information)

The first parameter of the functions is the key value, the second parameter is the founded message and the last parameter is the stored information. The message is deleted from the buffer while reading it out.

### Summary table of all public functions for EPTF transport routing

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_MessageBufferManager\_init\_CT () | Initializes the MessageBufferManager component |
| f\_EPTF\_MessageBufferManager\_cleanup\_CT | Cleanups the MessageBufferManager component |
| f\_EPTF\_MessageBufferManager\_setMessageToBuffer | Sets the given message into the buffer |
| f\_EPTF\_MessageBufferManager\_getMessageFromBuffer | Gets the message from the buffer |

## Transport components

There are several Transport component types, which have the same API, as it is described in 2.1. Since their API is the same, the import sections determine which communication port is used. Using the EPTF\_Transport\_CT component type the pl\_transportType argument determines the transport type to be used.

### Initialization

The user has to only initialize the component with the following function:

f\_EPTF\_Transport\_init(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in charstring pl\_selfName,  
in EPTF\_Transport\_InterfaceInformationList pl\_interfaceList := {},  
in boolean pl\_enableBufferManager := true)

The parameters are the transport type, the name of the component, the interface information list and the enableBufferManager. The first two parameters are mandatory. The last two are only optional and currently not used by IPL2. In IPL4 it is the same as in the old API.

If the users want to use all transport types simultaneously, or they do not want to decide at initialization, the value "ALL" should be used.

### Registering callback to calculate the message length

The user can set a callbackfunction to calculate the message with the following function:

f\_EPTF\_Transport\_registerMsgLenCallback4LGen (  
in EPTF\_Transport\_TransportType pl\_transportType,  
in EPTF\_Transport\_getMsgLen\_FT pl\_function,  
in EPTF\_IntegerList pl\_msgLenArgs,  
in charstring pl\_LGenType )

The second parameter of the function is a function reference to the callback function. The third parameter is an IntegerList, which contain the arguments of the getMsgLen function. The last parameter of the function is the LGenType identifier. The function will be set to this LGenType.

### Set the message and event handler functions

The users can set the message and event handler functions with the following function:

f\_EPTF\_Transport\_registerMsgCallback (  
in EPTF\_Transport\_TransportType pl\_transportType,  
in charstring pl\_LGenType,  
in EPTF\_Transport\_MsgCallback\_FT pl\_msghandler,  
in EPTF\_Transport\_EventCallback\_FT pl\_eventhandler := null  
)

The message handler in the third parameter and the event handler in the last parameter will be registered to the LGenType defined in the second parameter.

### Connect

The user can connect to an already opened socket on a host with the following function:

f\_EPTF\_Transport\_connect(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in ProtoTuple pl\_proto,  
in HostName pl\_localHost,  
in\_PortNumber pl\_localPort,  
in HostName pl\_remoteHost,  
in PortNumber pl\_remotePort,  
in charstring pl\_LGenType,  
out Result pl\_result,  
in boolean pl\_automaticBuffering := false,  
in OptionList pl\_options)

The second parameter defines the protocol to be used in the opening socket between pl\_localHost : pl\_localPort and pl\_remoteHost : pl\_remotePort to an LGenType (defined in parameters 3-8). The result of the function is given back in the pl\_result parameter. The last two parameters are not used in IPL2, only in IPL4. pl\_options can be used to enable SCTP multihoming.

### Listen

The user can open a socket on a host with the following function:

f\_EPTF\_Transport\_listen(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in ProtoTuple pl\_proto,  
in HostName pl\_hostName,  
in PortNumber pl\_portNumber,  
in charstring pl\_LGenType,  
out Result pl\_result,  
in boolean pl\_automaticBuffering := false,  
in OptionList pl\_options)

With pl\_proto protocol a socket is opened on pl\_hostName : pl\_PortNumber to an LGenType. The last two parameters are not used in IPL2, only in IPL4.

### Send a message

The user can send a message on the socket with the following function:

f\_EPTF\_Transport\_send(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in ConnectionId pl\_connId,  
in octetstring pl\_msg,  
out Result pl\_result,  
in boolean pl\_needBuffering := false,  
in ProtoTuple pl\_proto:={uspecified:={}})

The message pl\_msg is sent through the connection identified by the pl\_connId. The result of the function is given back in the pl\_result parameter. The pl\_needBuffering parameter is not used in IPL2, only in IPL4.

### Send a message to a remote peer

The user can send a message to a remote peer with the following function:

f\_EPTF\_Transport\_sendTo(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in ConnectionId pl\_connId,  
in HostName pl\_remoteHost,  
in PortNumber pl\_remotePort,  
in octetstring pl\_msg,  
out Result pl\_result,  
in boolean pl\_needBuffering := false,  
in ProtoTuple pl\_proto:={uspecified:={}})

The message pl\_msg is sent through the connection identified by the pl\_connId to the host to pl\_remoteHost : pl\_remotePort. The result of the function is given back in the pl\_result parameter. The pl\_needBuffering parameter is not used in IPL2, only in IPL4.

### Close connections

The user can close a connection with the following function:

f\_EPTF\_Transport\_Close(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in ConnectionId pl\_connId,  
out Result pl\_result,  
in ProtoTuple pl\_proto:={uspecified:={}})

The connection to close is identified by the pl\_connId.

### Setup interfaces

The user can set up interfaces with the following function:

f\_EPTF\_Transport\_setUpInterfaces (  
in EPTF\_Transport\_TransportType pl\_transportType,  
out Result pl\_result)

This function sets up the interfaces that were passed to the transport in the init function.

### Append interfaces

The following function can be used to set up additional interfaces (that is, after init):

f\_EPTF\_Transport\_appendInterfaces(  
in EPTF\_Transport\_TransportType pl\_transportType,  
in EPTF\_Transport\_InterfaceInformationList pl\_interfaceList,  
out Result pl\_result)

### Remove interfaces

The user can set down interfaces with the following function:

f\_EPTF\_Transport\_setDownInterfaces (  
in EPTF\_Transport\_TransportType pl\_transportType,  
out Result pl\_result)

This function also sets down interfaces set up through f\_EPTF\_Transport\_appendInterfaces.

### Activate default buffering handler

The user can activate the default buffering handler with the function:

f\_EPTF\_Transport\_activateDefaultBufferingHandler(

  in EPTF\_Transport\_TransportType pl\_transportType,

  in charstring pl\_LGenType := "")

For the IPL4 transport type the function activates the default buffer handler in IPL4 transport. However, for the IPL2 transport type the function does nothing.

### Summary table of all public functions for EPTF Transport IPL2 Communication Port, EPTF\_CLL\_Transport and the IPL4 Communication Port new API

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_Transport\_init | Initializes the component |
| f\_EPTF\_Transport\_cleanup\_CT | Cleanups component |
| f\_EPTF\_Transport\_registerMsgLenCallback4LGenType | Sets a getMessageLength function |
| f\_EPTF\_Transport\_registerMsgCallback | Sets a message handler and an event handler function |
| f\_EPTF\_Transport\_connect | Function to connect to a socket |
| f\_EPTF\_Transport\_listen | Function to listen on a socket |
| f\_EPTF\_Transport\_send | Function to send a message over a socket |
| f\_EPTF\_Transport\_sendTo | Function to send a message to a remote host |
| f\_EPTF\_Transport\_close | Function to close a socket |
| f\_EPTF\_Transport\_setUpInterfaces | Sets up the interfaces given at init |
| f\_EPTF\_Transport\_appendInterfaces | Sets up additional interfaces |
| f\_EPTF\_Transport\_setDownInterfaces | Sets down the interfaces |
| f\_EPTF\_Transport\_activateDefaultBufferingHandler | Activates the default buffering handler |

## The IPL4 Communication Port component with the obsolete API

This set of functions was made before the raise of the idea of the common transport layer. The names of its functions have IPL4-specific identifiers, and do not correspond to the function signatures of the common transport. There are functions which do not exist in the common API because they do not fit the new transport handling concept. By using this function set, it is impossible to write a user code where the IPL4 transport can be replaced easily with other transport type, for example IPL2 transport. This function set is obsolete. It is used only to be compatible with legacy code.

The following sections describe the set of these functions.

### Initialization

The user has to only initialize the component with the following function:

f\_EPTF\_CommPort\_IPL4\_init(  
in EPTF\_CommPort\_IPL4\_InterfaceInformationList pl\_interfaceList,  
in Boolean pl\_enableBufferManager)

The parameters of the IPL4 communication port are the interface information list and the enableBufferManager. With the first parameter the user can define interface list, and these interfaces will be set up. With the second parameter the user enables the buffering mechanism in the IPL4 communication port component.

### Set getMessageLength function

The user can set the getMessageLength function to the IPL4 testport with the following function:

f\_EPTF\_CommPort\_IPL4\_setMsgLen (  
in integer pl\_connId,  
in f\_IPL4\_getMsgLen pl\_function,  
in ro\_integer pl\_msgLenArgs  
)

The first parameter of the function is the connection Id. The function will be set to this connection Id. The second parameter of the function is a function reference to the getMessageLength function. The last parameter is an IntegerList which contains the arguments of the getMsgLen function.

### Set receive template and handler function

The user can set receive templates to the IPL4 tetstport and handler functions with the following function:

f\_EPTF\_CommPort\_IPL4\_setReceive (  
in template EPTF\_CommPort\_IPL4\_IncomingMessage pl\_receiveTemplate,  
in f\_EPTF\_CommPort\_IPL4\_messageProcess\_FT pl\_msghandler  
)

The first parameter of the function is the receive template and the second parameter is the reference of the handler function. If the received message matches with the template, the registered handler function will be called.

### The send function

The user can send general messages with the following function:

f\_EPTF\_CommPort\_IPL4\_send(  
in EPTF\_CommPort\_IPL4\_ASP\_OutgoingMessage pl\_data,  
in f\_EPTF\_CommPort\_IPL4\_messageProcess\_FT pl\_msghandler,  
in Boolean pl\_needBuffering  
)

The first parameter of the function is the general message. This can be Connection, Connection close, Send, SendTo, or Listen messages. The second parameter is a function reference. This function will be called after the operation. The last parameter can be used to enable the Buffering mechanism to Send and SendTo operation. The buffering mechanism can be used only if one message or connection ID must wait in the buffer.

### The activate handler function

The user can set the default event handler with the following function:

f\_EPTF\_CommPort\_IPL4\_activateDefaultBufferingHandler

### Default connection result event handler

The IPL4 test port component provides a default connection result handler function f\_EPTF\_CommPort\_IPL4\_defaultConnResultEventHandler ().

The user can use this function to receive the Connection result event (“IPL4\_ERROR\_AVAILABLE”) and send the buffered message after the connection result received.

### Summary table of all public functions for EPTF Transport CommPortIPL4 legacy API

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_CommPort\_IPL4\_init | Initializes the IPL4 Communication Port component |
| f\_EPTF\_CommPort\_IPL4\_setUpInterfaces | Sets up the given interfaces |
| f\_EPTF\_CommPort\_IPL4\_setDownInterfaces | Sets down the given interfaces |
| f\_EPTF\_CommPort\_IPL4\_cleanup\_CT | Cleanups the IPL4 Communication Port component |
| f\_EPTF\_CommPort\_IPL4\_setMsgLen | Sets a getMessageLength function to the IPL4 testport |
| f\_EPTF\_CommPort\_IPL4\_setReceive | Sets a receive template and a handler function to the IPL4 testport |
| f\_EPTF\_CommPort\_IPL4\_send | Function to send information over the IPL4 testport |
| f\_EPTF\_CommPort\_IPL4\_activateDefaultBufferingHandler | Function to activate the default event handler |
| f\_EPTF\_CommPort\_IPL4\_defaultConnResultEventHandler | Function to handle the Connection Result Event |

## IPL2 support of multiple interfaces

The IPL2 transport can be set to use either a single interface set through test port parameters in the configuration file, or multiple interfaces using the module parameter tsp\_EPTF\_TransportIPL2\_multipleInterfacesMode.

If this module parameter is true, the interfaces are configured using the interface list of the init function and, optionally, the interface list passed in to f\_EPTF\_Tranport\_appendInterfaces after initialization. The pcap packet filters are generated automatically.

The following charstring module parameters are used to set up the loopback interface in multiple interfaces mode:

|  |  |
| --- | --- |
| **module parameter** | **default value** |
| tsp\_EPTF\_TransportIPL2\_loopbackInterface | “lo” |
| tsp\_EPTF\_TransportIPL2\_loopbackInterfaceAddress | “127.0.0.0” |
| tsp\_EPTF\_TransportIPL2\_loopbackInterfaceMask | “255.0.0.0” |
| tsp\_EPTF\_TransportIPL2\_openLoopbackInterface | false |

In multiple interfaces mode, interfaces are opened in one of the following cases:

* the loopback interface is implicitly opened at init if the module parameter tsp\_EPTF\_TransportIPL2\_openLoopbackInterface is set to true
* interfaces that are passed in the init function
* interfaces added through appendInterfaces after init

The loopback interface is opened regardless of tsp\_EPTF\_TransportIPL2\_openLoopbackInterface if it is specified for the init function or for the appendInterfaces function.

Outgoing messages are routed by the destination IP address using the route table of the transport. Incoming messages are accepted regardless of the iterface, that is if a socket is opened, it will accept messages from any of the open ethernet interfaces.

# IPsec Support

The Transport feature supports IPsec. Only IPL4 transportType can be used for IPsec. The Transport IPsec handler uses the XFRM API of TCCUsefulFunctions.

The XFRM functions use the kernel to implement the IPsec functionality. For this, root privilege is required. Without root privilege, the IPsec functionality of the Transport will not work or no encryption will be used.

**Note:** If the same machine is used for the source and the destination addresses, the kernel may not encrypt the messages even if IPsec is configured that way. Two different hosts are required to get encrypted messages on the network.

## Initialization

The Transport IPsec API can only be used after initializing the IPsec Handler of the Transport by calling the function:

f\_EPTF\_Transport\_IPsecHandler\_init\_CT(  
 in charstring pl\_selfName,   
 in EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_Server\_CT pl\_LoggingIPSecServer\_CT := null  
)

The parameter pl\_LoggingIPSecServer\_CT specifies the IPsec logging server component. If it is not specified or null, the IPsec data will not be sent to the IPsec logging server.

The IPsec Handler is not initialized automatically in the Transport.

## Checking initialization

The initialization of the Transport IPsec Handler can be checked by the function:

f\_EPTF\_Transport\_IPsecHandler\_initialized()

It returns true if f\_EPTF\_Transport\_IPsecHandler\_init\_CT was called.

## Allocating SPI

To request a new security parameter index (SPI) from the kernel the following function can be used:

f\_EPTF\_Transport\_IPsecHandler\_allocate\_SPI(  
 in AllocSPI\_Info pl\_AllocSPI\_info,   
 inout integer pl\_spi  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

It will return a free SPI in the pl\_spi argument. The return value shows if the function call is successful or an error occurred.

## Creating Security Policy

The security policy (SP) can be created by the function:

f\_EPTF\_Transport\_IPsecHandler\_createSP(  
 in SPAddInfo pl\_sp\_info  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

The pl\_sp\_info argument specifies the source and destination addresses, the SPI, the IPsec protocol to use and other parameters that are necessary for the creation of Security Policy.

The return value shows if the function call is successful or an error occurred.

## Creating Security Association

The security association (SA) can be created by the function:

f\_EPTF\_Transport\_IPsecHandler\_createSA(  
 in SAAddInfo pl\_sa\_info  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

The pl\_sa\_info argument specifies the source and destination addresses, the SPI, the IPsec protocol, the authentication and encryption algorithms and keys to use, and other parameters that are necessary for the creation of Security Association.

The return value shows if the function call is successful or an error occurred.

After the SA and SP is created for the connection (on both hosts) the messages will be sent encrypted in the configured direction.

## Deleting Security Policy

The security policy that has been added earlier can be deleted by the function:

f\_EPTF\_Transport\_IPsecHandler\_deleteSP(  
 in SPDelInfo pl\_sp\_del\_info  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

The pl\_sp\_del\_info argument specifies the parameters of the SP to be deleted.

If the f\_EPTF\_Transport\_IPsecHandler\_deleteSP function is called for TCP protocol, this function will delete the SP information.

The closing TCP messages (FIN, ACK) will only be encrypted if the SA/SP information is not deleted in neither side of the connection. If they are deleted before the TCP connection is closed, they will be sent unencrypted.

## Deleting Security Association

The security association that has been added earlier can be deleted by the function:

f\_EPTF\_Transport\_IPsecHandler\_deleteSA(  
 in SADelInfo pl\_sa\_del\_info  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

The pl\_sa\_del\_info argument specifies the parameters of the SA to be deleted.

## Deleting all SA/SP

To delete all SA and SP data from the kernel, the following functions can be used respectively:

f\_EPTF\_Transport\_IPsecHandler\_flushSP(  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

f\_EPTF\_Transport\_IPsecHandler\_flushSA(  
) runs on EPTF\_Transport\_IPsecHandler\_CT return XFRM\_Result

## IPsec Logging

The IPsec Logging makes it possible to log IPsec data to a configurable log file. The following information is logged in CSV format:

* Protocol
* Source address
* Destination Address
* SPI
* Encryption algorithm
* CK (key for the encryption)
* Authentication algorithm
* IK (key for the authentication)

These are logged automatically whenever an SA is created by the f\_EPTF\_Transport\_IPsecHandler\_createSA function and IPsec logging is enabled. The new information is appended to the file. If the file does not exist, it will be created automatically.

The logging takes place on the IPsec Logging Server component. It can be started by its behavior function:

f\_EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_Server\_behaviour(  
 in charstring pl\_EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_IPSec\_File\_Path,   
 in charstring pl\_EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_IPSec\_File\_Name  
)   
runs on EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_Server\_CT

Its two parameters define the path and name of the log file that will be used to store the information.

To enable IPsec logging, the IPsec logging server component should be passed to the Transport IPsec initializer function.

## Summary table of all public functions for EPTF transport IPsec Handler

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_Transport\_IPsecHandler\_init\_CT () | Initializes the IPsecHandler component |
| f\_EPTF\_Transport\_IPsecHandler\_initialized | Returns true if IPsecHandler was initialized |
| f\_EPTF\_Transport\_IPsecHandler\_createSP | Creates Security Policy |
| f\_EPTF\_Transport\_IPsecHandler\_createSA | Creates Security Association |
| f\_EPTF\_Transport\_IPsecHandler\_deleteSP | Deletes Security Policy |
| f\_EPTF\_Transport\_IPsecHandler\_deleteSA | Deletes Security Association |
| f\_EPTF\_Transport\_IPsecHandler\_flushSP | Removes all SP from the kernel |
| f\_EPTF\_Transport\_IPsecHandler\_flushSA | Removes all SA from the kernel |
| f\_EPTF\_Transport\_IPsecHandler\_allocate\_SPI | Requests a new, unused SPI from the kernel |
| f\_EPTF\_CLL\_Transport\_IPsecHandler\_Logging\_Server\_behaviour | Behaviour function of the IPsec logging server |

# DTE handling

When the Transport executes the message handler functions registered by f\_EPTF\_CommPort\_IPL4\_setReceive, f\_EPTF\_CommPort\_IPL4\_send or EPTF\_CLL\_TransportIPL4\_Functions.f\_EPTF\_Transport\_registerMsgCallback and a dynamic test case error occurs, then the execution of the component will not stop due to the error. However, Transport will print out a warning message about the error.

By default DTE handling is disabled.

# References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. [Part 1: Core Language](http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.02.01_60/es_20187301v030201p.pdf)
2. 1/198 17-CRL 113 200/6 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 109 21-CNL 113 512-26 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Product Revision Information
4. 155 17-CNL 113 512 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
5. EPTF Core Library for TTCN-3 toolset with TITAN, [Reference Guide](http://ttcn.ericsson.se/TCC_Releases/Libraries/EPTF_Core_Library_CNL113512/doc/apidoc/html/index.html)
6. 25/198 17-CNL 113 512 Uen  
   EPTF Core Library Transport, User Guide