# MICROSAR Classic TCPIP

Technical Reference

Version 17.03.00

| | |
|---|---|
| Authors | visal, visfdn, viswnk, visjsb, vismjv, visabn, vispcn, vistwe, visbdk, vistbk, sschyja, visjer, tschubert |
| Status | Released |

# Document Information

## History

| Author | Date | Version | Remarks |
|---|---|---|---|
| visal<br>visfdn | 2015-05-18 | 1.0.0 | Creation of document.<br>This document combines the Technical References of the former modules TcpIp, IpV4 and IpV6. |
| visfdn | 2015-10-19 | 1.0.1 | Added Known Issues (IETF RFC Conformance) |
| visfdn<br>visal | 2016-01-13 | 2.0.0 | Added IPv4 Fragmentation<br>Added API to read the local physical address<br>Added sending the FQDN option via DHCP to identify the ECU<br>Added triggering of IP address assignment |
| visal | 2016-03-15 | 2.1.0 | TLS as TcpIp plug-in |
| visfdn | 2016-06-29 | 3.0.0 | Added description of unicast address assignment methods Added description of configuration of static on-link prefixes for IPv6 |
| visfdn | 2016-09-22 | 3.1.0 | Added description of optional callout for link-local address assignment. |
| visfdn | 2016-10-20 | 4.0.0 | Updated description of MainFunction, <Up>_CopyTxData() Callback and Tls Heartbeat. |
| visfdn | 2017-01-18 | 4.1.0 | Release of DHCPv4 Server. |
| viswnk | 2017-03-17 | 5.0.0 | Added Production Error Reporting and Other Error Reporting and Diagnostic Options |
| visjsb<br>viswnk | 2017-04-28<br>2017-05-03 | 6.0.0 | Added description of TcpIp_ClearARCache()<br>Extended Production Error Reporting, Added 5.5.1.7 TcpIpDuplicateAddressDetectionConfig and 3.6.3.4 IPv6 Duplicate Address Detection Conflict |
| visjsb | 2017-05-23 | 6.1.0 | Added description of TcpIp_GetNdpCacheEntries() according ASR 4.3 |
| vismjv | 2017-05-30 | 6.1.0 | Added description of TcpIp_GetArpCacheEntries() according ASR 4.3 |
| visal | 2017-08-10 | 6.2.0 | Add new socket owner callback for TLS validation result |
| vismjv | 2018-03-15 | 7.0.0 | Added Support of DHCPv4 Requested IP Address Callout |
| visfdn | 2018-05-29 | 7.0.1 | Updated list of “Dynamic Files”. |
| visfdn | 2018-07-04 | 8.0.0 | Updated API documentation.<br>Removed legacy APIs. |
| visabn | 2019-03-05 | 9.0.0 | Updated Component History.<br>Updated Scalability Class reference.<br>Added description for TcpIp_SendGratuitousArpReq () TcpIp_MainFunctionRx (), TcpIp_MainFunctionTx (), TcpIp_MainFunctionState () |
| vispcn | 2019-03-22 | 10.00.00 | Added DHCP user options and DHCP event callout function. |

| | | | |
|---|---|---|---|
| vistwe | 2019-05-02 | 10.00.01 | Added chapter “Security Information” |
| vispcn | 2019-07-16 | 10.00.02 | Improved feature description of DHCP user option handling. |
| vismjv<br>visabn | 2019-08-05<br>2019-08-12 | 12.00.00 | Added service to read and reset measurement data for diagnostic purpose (TcpIp_GetAndResetMeasurementData() API)<br>Added service to support IPsec Authentication Header for IpV4 |
| vismjv | 2019-09-09 | 13.00.00 | Added support of Transmission of Gratuitous ARP reply on startup |
| visabn | 2020-01-28 | 13.00.01 | Extended TcpIp_GetAndResetMeasurementData() to support IPsec SA diagnostics |
| vismjv | 2020-02-25 | 13.00.02 | Extended TcpIp_GetAndResetMeasurementData() to support for insufficient TCP Tx/Rx buffer and IpV4 reassembly buffer |
| vismjv | 2020-02-25 | 13.00.02 | Updated configuration parameter of <Up_PhysAddrTableEntryDiscarded> callout |
| visbdk | 2020-04-29 | 13.00.03 | Update description of TcpIp_RequestComMode() API |
| vismjv | 2020-05-06 | 14.00.00 | Added Support of AUTOSAR parameter TcpIpArpRequestTimeout |
| vismjv | 2020-05-28 | 14.00.01 | Added Support of Icmpv4 Destination Unreachable Message |
| visbdk | 2020-10-12 | 14.00.02 | Names of authors removed.<br>Chapter ‘Component History’ removed.<br>Extend description of TcpIp_Bind API.<br>Add chapter ‘Timing Considerations’. |
| vispcn | 2020-10-27 | 14.01.00 | Updated description of configuration variants and initialization of the module. |
| vispcn,<br>vismjv | 2020-11-13 | 14.01.01 | Added chapter about generation customizations of configuration dependent data.<br>Updated parameter name of API TcpIp_GetDhcpTimeoutInfo(). |
| vistbk | 2021-01-18 | 14.02.00 | Added chapter about Static Default Router Configuration.<br>Clarified description of TCPIP_MEAS_DROP_UDP/ TCPIP_MEAS_DROP_TCP. |
| vistbk | 2021-02-01 | 14.02.01 | Updated chapter about Static Default Router Configuration. |
| vispcn | 2021-03-08 | 15.00.00 | Added support for AUTOSAR TLS. |
| vismjv | 2021-04-21 | 15.01.00 | Updated <Up_IpSecSpdCalloutFunction> and <Up_IpSecAuditEventCalloutFunction>. |
| visabn | 2021-04-22 | 15.01.00 | Added Multicore Support. |

| | | | |
|---|---|---|---|
| vistbk | 2021-04-27 | 15.01.01 | Updated chapter for IPsec singlecall changes. |
| vistbk | 2021-06-23 | 15.02.00 | Updated TcpIp_IpSecSaEntryPairAdd.<br>Added new IPsec parameter TcpIpIpSecSaEntryTxActivationTimeout. |
| vistbk | 2021-06-26 | 15.02.00 | Added prepare deletion of IPsec SA entry pairs.<br>Added chapter about IPsec. |
| visal | 2021-06-30 | 15.03.00 | Added API TcpIp_GetAvailableTxBufferSize to read TCP/TLS available tx buffer size. |
| visabn | 2021-07-01 | 15.03.00 | Added support for CSM Indirect Access for TcpIp |
| visabn | 2021-07-14 | 15.03.00 | Added support for TLS RL Fragmentation |
| vispcn | 2021-07-19 | 15.03.00 | Removed parameter TcpIpTcpFinWait2Timeout from the list of not supported features. |
| visbdk | 2021-07-22 | 15.04.00 | Added TcpIp Event TCPIP_TLS_HANDSHAKE_SUCCEEDED |
| vistbk | 2021-09-15 | 15.05.00 | Add restriction for multicore instances for some API calls |
| vistbk | 2021-09-17 | 15.06.00 | Add requirement to API TcpIp_IpSecSaEntryPairAdd |
| vismjv | 2021-09-20 | 15.06.00 | Listed Simultaneous Open Attempts as not supported feature. |
| vistbk | 2021-10.01 | 15.06.00 | Added “Disable UDP Checksum Calculation in SW” feature. |
| vismjv | 2021-10-05 | 15.06.00 | Added parameter TCPIP_PARAMID_TLS_CONNECTION_ASSIGNME NT |
| vistbk | 2021-10-10 | 15.06.00 | Add Multicore Mainfunctions to API list |
| vismjv | 2021-10-15 | 15.07.00 | Updated TechRef for TC8 testes with marker ‘OPEN_ALLIANCE_TC8_TEST_RELEVANT’. |
| vistbk | 2021-10-29 | 15.08.00 | Allowed mapping of big data structs in different sections. |
| vistbk | 2021-11-12 | 15.09.00 | Update chapter ‘Security Information’ with new feature to make port scanning more difficult. |
| vistwe | 2021-11-16 | 15.09.00 | Changed parameter TlsConIdx to SocketId in function TcpIp_Tls_GetUserError. |
| visbdk | 2021-12-13 | 15.10.00 | Update TlsBuffer configuration,<br>Update TLS Extension configuration,<br>Add TLS Validation Result callout. |

| | | | |
|---|---|---|---|
| vistwe | 2021-12-13 | 15.10.00 | Added support for real-time clock. Using it in the client and server random number. |
| vistbk | 2021-12-20 | 15.11.00 | Change parameter MSL for TCP sockets with TcpIp_ChangeParameter API |
| vistwe | 2021-12-23 | 15.11.00 | Added API to read the identifier of the used root certificate TcpIp_Tls_GetRootCertificateId. |
| vistbk | 2022-01-12 | 15.11.00 | Update chapter “Critical Section” |
| visbdk | 2022-01-12 | 15.11.00 | Update chapter “Pseudo Random Function (PRF)” |
| vispcn | 2022-01-26 | 15.12.00 | Changed memory section names and scope of delivery.<br>Updated links to RFC documents.<br>Clarified not supported AUTOSAR R4.2.1 standard conform feature. |
| visbdk | 2022-02-15 | 16.00.00 | Add ECC TLS configuration. |
| vispcn | 2022-03-07 | 16.00.00 | Product name updated to MICROSAR Classic. |
| vispcn | 2022-03-07 | 16.00.00 | Support communication state ONHOLD. |
| vistbk | 2022-03-16 | 16.01.00 | Support multiple TCP connection with equal local endpoint. |
| vismjv | 2022-03-22 | 16.01.00 | Updated TechRef for TCP retransmission timeout calculation. |
| vispcn | 2022-03-24 | 16.01.00 | Added IdsM security event reporting. |
| visbdk | 2022-03-31 | 16.02.00 | Add supported functionality for TLS-Server and TLS-Client. |
| visbdk | 2022-04-01 | 16.02.00 | Add additional recommendation for TLS time stamp |
| vistbk | 2022-04-01 | 16.02.00 | Added handling of not processable IPv6 extension header. |
| vispcn | 2022-04-14 | 16.02.00 | Updated references in IETF RFC conformance chapter. |
| visbdk | 2022-04-21 | 16.02.00 | Add TLS extension Record Size Limit (RFC8449) |
| vispcn | 2022-05-17 | 16.03.00 | Removed description of not supported parameter TcpIpTlsMaxConnections. |
| vistbk | 2022-06-20 | 16.04.00 | Extend limitation of the number of supported sockets |
| visal | 2022-06-22 | 16.04.00 | Add limitation to not update certs for open sockets. |
| visbdk | 2022-06-25 | 16.05.00 | Add support for TLS-Server configuration. |
| vismjv | 2022-07-08 | 16.05.00 | Adapted description for ARP discarded entry handling. |
| vistbk | 2022-07-14 | 16.05.00 | Add support for more than 250 sockets |

| | | | |
|---|---|---|---|
| vistbk | 2022-07-18 | 16.05.00 | Added chapter IPv6 configuration |
| vistbk | 2022-08-01 | 16.06.00 | Added chapter IPv6 address assignment |
| sschyja | 2022-08-03 | 16.06.00 | Added chapter Server Name Indication |
| vistbk | 2022-08-11 | 16.06.00 | Added support for IPv6 manual address assignment |
| visbdk | 2022-08-12 | 16.06.00 | Added support for TLS Error Indication callout |
| vismjv | 2022-08-24 | 16.07.00 | Added chapter IPv4 configuration |
| sschyja | 2022-08-24 | 16.07.00 | Added support for PSK for the TLS Server |
| vispcn | 2022-09-02 | 16.07.00 | Updated list of static source files. |
| vistbk | 2022-09-09 | 16.07.00 | Update IPsec behavior |
| vistbk | 2022-09-14 | 16.07.00 | Correct ICMPv6 Destination Unreachable support |
| vistbk | 2022-09-14 | 16.07.00 | Add IPsec restriction for CSM synchronous jobs |
| visjer | 2022-09-21 | 16.07.00 | Interpret not set TcpIpTlsPortAssignment as wildcard |
| visbdk | 2022-09-22 | 16.07.00 | Add new TLS event callback for TCPIP_TLS_HANDSHAKE_SUCCEEDED |
| visbdk | 2022-10-05 | 16.07.00 | Add information about runtime TLS handshake selection. |
| sschyja | 2022-10-11 | 16.07.00 | Use TcpIpTlsLocalAddrsRef to bind a TLS connection to a dedicated local address |
| visjer | 2022-12-08 | 16.08.00 | Added support for SECP521r1 curve for TLS key exchange |
| visal | 2023-01-12 | 16.08.00 | Added support for ClientAuthentication with curves SECP256r1 and SECP521r1 for the TLS client. |
| visbdk | 2023-01-25 | 16.08.01 | Added configuration hints for TLS when using HSM crypto driver. |
| visbdk | 2023-02-07 | 16.08.02 | Add information for TLS buffer configuration. |
| visjer | 2023-04-11 | 16.09.00 | Added support for TLS Client single connection using certificates with different curves |
| vistbk | 2023-07-17 | 16.10.02 | Add limitation to TcpIp_RequestComMode |
| vistbk | 2023-08-08 | 16.10.03 | Add chapter UDP Address matching |
| visal | 2023-10-06 | 17.00.00 | Support of TLS 1.3 client |
| vistbk | 2023-10-07 | 17.00.00 | Add configuration for TLS 1.3 HKDF |
| tschubert | 2023-11-13 | 17.01.00 | Add new API to read TLS secrets and send NSS Key Log Frame |
| sschyja | 2023-11-16 | 17.01.00 | Added support for TLS 1.2 Server SECP521r1 curve for TLS key exchange |
| vistbk | 2023-11-22 | 17.01.00 | Add limitation to IPsec |

| | | | |
|---|---|---|---|
| vistbk | 2023-11-22 | 17.01.00 | Add new ARP APIs |
| tschubert | 2023-11-23 | 17.01.00 | Added support for curve x448 key exchange for TLS 1.3 client |
| visjer | 2023-12-01 | 17.01.00 | Added support for TLS Server connection with client authentication |
| vistwe | 2024-15-01 | 17.01.01 | Added support for ed448 and ed25519 for TLS 1.3 client authentication |
| tschubert | 2024-01-31 | 17.01.01 | Added info about parameters CSM job/key for signature verification |
| vistbk | 2024-02-02 | 17.01.01 | Modify description of TcpIp_ChangeParameter |
| vistbk | 2024-02-05 | 17.01.01 | Add supported ICMPv6 packets types |
| vistbk | 2024-02-16 | 17.02.00 | Change parameter RTT for TCP sockets with TcpIp_ChangeParameter API |
| vistbk | 2024-02-29 | 17.02.00 | Configurable drop of ICMPv6 Router Advertisement and Redirect |
| tschubert | 2024-03-15 | 17.03.00 | Improved description for KeyUpdateCallout / Tls_GetTlsSecrets-API / TLS 1.3 close events |
| visjer | 2024-03-18 | 17.03.00 | Added support for TLS 1.3 Client Session Resumption |
| tschubert | 2024-03-22 | 17.03.00 | Added information for behavior in case of failing CSM API-call during encryption/HMAC generation in TLS |

## Reference Documents

| No. | Source | Title | Version |
|---|---|---|---|
| [1] | AUTOSAR | Specification of SCP/IP Stack | R4.2.1 |
| [2] | AUTOSAR | Specification of Default Error Tracer | R4.2.1 |
| [3] | AUTOSAR | Specification of Diagnostic Event Manager | R4.2.1 |
| [4] | AUTOSAR | List of Basic Software Modules | R4.2.1 |
| [5] | AUTOSAR | Specification of Ethernet Interface | R4.2.1 |
| [6] | AUTOSAR | Specification of Socket Adaptor | R4.2.1 |
| [7] | Vector | TechnicalReference CommonAsr_ComStackLib | see delivery |
| [8] | IETF RFC768 | User Datagram Protocol | Aug 1980 |
| [9] | IETF RFC791 | Internet Protocol | Sep 1981 |
| [10] | IETF RFC792 | Internet Control Message Protocol (ICMP) | Sep 1981 |
| [11] | IETF RFC793 | Transmission Control Protocol | Sep 1981 |
| [12] | IETF RFC813 | Window and Acknowledgement Strategy in TCP | Jul 1982 |
| [13] | IETF RFC826 | An Ethernet Address Resolution Protocol (ARP) | Nov 1982 |
| [14] | IETF RFC894 | A Standard for the Transmission of IP Datagrams over Ethernet Networks | Apr 1984 |
| [15] | IETF RFC1112 | Host Extensions for IP Multicasting | Aug 1989 |
| [16] | IETF RFC1122 | Requirements for Internet Hosts -- Communication Layers | Oct 1989 |
| [17] | IETF RFC1323 | TCP Extensions for High Performance | May 1992 |
| [18] | IETF RFC1981 | Path MTU Discovery for IP version 6 | Aug 1996 |
| [19] | IETF RFC2018 | TCP Selective Acknowledgment Options | Oct 1996 |
| [20] | IETF RFC2131 | Dynamic Host Configuration Protocol (DHCP) | Mar 1997 |
| [21] | IETF RFC2460 | Internet Protocol, Version 6 (IPv6) Specification | Dec 1998 |
| [22] | IETF RFC2464 | Transmission of IPv6 Packets over Ethernet Networks | Dec 1998 |
| [23] | IETF RFC2711 | IPv6 Router Alert Option | Oct 1999 |
| [24] | IETF RFC3122 | Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification | Jun 2001 |
| [25] | IETF RFC3315 | Dynamic Host Configuration Protocol for IPv6 (DHCPv6) | Jul 2003 |
| [26] | IETF RFC3810 | Multicast Listener Discovery Version 2 (MLDv2) for IPv6 | Jun 2004 |
| [27] | IETF RFC3927 | Dynamic Configuration of IPv4 Link-Local Addresses | May 2005 |
| [28] | IETF RFC4291 | IP Version 6 Addressing Architecture | Feb 2006 |
| [29] | IETF RFC4429 | Optimistic Duplicate Address Detection (DAD) for IPv6 | Apr 2006 |
| [30] | IETF RFC4443 | Internet Control Message Protocol (ICMPv6) for the IPv6 Specification | Mar 2006 |
| [31] | IETF RFC4702 | The DHCP Client FQDN option | Oct 2006 |

| | | | |
|---|---|---|---|
| [32] | IETF RFC4704 | The DHCPv6 Client FQDN option | Oct 2006 |
| [33] | IETF RFC4861 | Neighbor Discovery for IP version 6 (IPv6) | Sep 2007 |
| [34] | IETF RFC4862 | IPv6 Stateless Address Autoconfiguration | Sep 2007 |
| [35] | IETF RFC4884 | Extended ICMP to Support Multi-Part Messages | Apr 2007 |
| [36] | IETF RFC4941 | Privacy Extensions for Stateless Address Autoconfiguration in IPv6 | Sep 2007 |
| [37] | IETF RFC5095 | Deprecation of Type 0 Routing Headers in IPv6 | Dec 2007 |
| [38] | IETF RFC5482 | TCP User Timeout Option | Mar 2009 |
| [39] | IETF RFC5681 | TCP Congestion Control | Sep 2009 |
| [40] | IETF RFC5722 | Handling of Overlapping IPv6 Fragments | Dec 2009 |
| [41] | IETF RFC5942 | IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes | Jul 2010 |
| [42] | IETF RFC6085 | Address Mapping of IPv6 Multicast Packets on Ethernet | Jan 2011 |
| [43] | IETF RFC6106 | IPv6 Router Advertisement Options for DNS Configuration | Nov 2010 |
| [44] | IETF RFC6298 | Computing TCP's Retransmission Timer | Jun 2011 |
| [45] | IETF RFC6582 | The NewReno Modification to TCP's Fast Recovery Algorithm | Apr 2012 |
| [46] | IETF RFC6724 | Default Address Selection for Internet Protocol version 6 (IPv6) | Sep 2012 |
| [47] | IETF RFC4301 | Security Architecture for the Internet Protocol | Dec 2005 |
| [48] | IETF RFC4302 | IP Authentication Header | Dec 2005 |
| [49] | IETF RFC4543 | The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH | May 2006 |
| [50] | IANA-IKEv2 | Internet Key Exchange Version 2 (IKEv2) Parameters | May 2019 |
| [51] | IETF Internet Draft | Deprecating gmt_unix_time in TLS | Dec. 2013 |
| [52] | IETF RFC8449 | Record Size Limit for TLS | Aug. 2018 |
| [53] | IETF RFC5246 | The Transport Layer Security (TLS) Protocol Version 1.2 | Aug. 2008 |
| [54] | IETF RFC8446 | The Transport Layer Security (TLS) Protocol Version 1.3 | Aug. 2018 |

## Scope of the Document

This technical reference describes the general use of the TCPIP basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler, controller) your Vector Ethernet Bundle has been configured for.

**Caution**
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

## Contents

## Illustrations

## Tables

# 1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module TCPIP as specified in [1].

| | | |
|---|---|---|
| **Supported Configuration Variants:** | pre-compile | |
| **Vendor ID:** | TCPIP_VENDOR_ID | 30 decimal<br>(= Vector-Informatik, according to HIS) |
| **Module ID:** | TCPIP_MODULE_ID | 170 decimal<br>(according to ref. [4]) |

The TcpIp provides access to a socket interface based on the Internet Protocol (IPv4 and IPv6) over Ethernet.

The TcpIp module consists of 5 submodules:

> TcpIp (TCP, UDP, Socket interface)
> IpV4 (IPv4, ARP, ICMPv4, DHCPv4 client)
> IpV6 (IPv6, NDP, ICMPv6, DHCPv6 client)
> DhcpV4Server (DHCPv4 server)
> IpSec (Authentication Header for IpV4)
> Tls (Transport Layer Security)

**Note**
The submodules **IpV4**, **IpV6, IpSec, Tls** and **DhcpV4Server** must be licensed separately for use in production code. Use of all the submodules is allowed in development code.

Scalability Classes (SC*) based on different combinations of IP submodules are possible to be configured in the TCPIP:

| Scalability Class | Submodules |
|---|---|
| SC 1 | **IpV4** only |
| SC 2 | **IpV6** only |
| SC 3 | **IpV4/IpV6** Dual stack |

**TcpIp** submodule is configured by default in all SC.
**DhcpV4Server** submodule may be configured only in SC1 and SC3.
**IpSec** submodule may be configured only in SC1 and SC3.
**Tls** submodule may be configured in all SC.

* SC: Scalability Class according to [1] Specification of SCP/IP Stack

## 1.1 Architecture Overview

The following figure shows where the TCPIP is located in the AUTOSAR architecture.

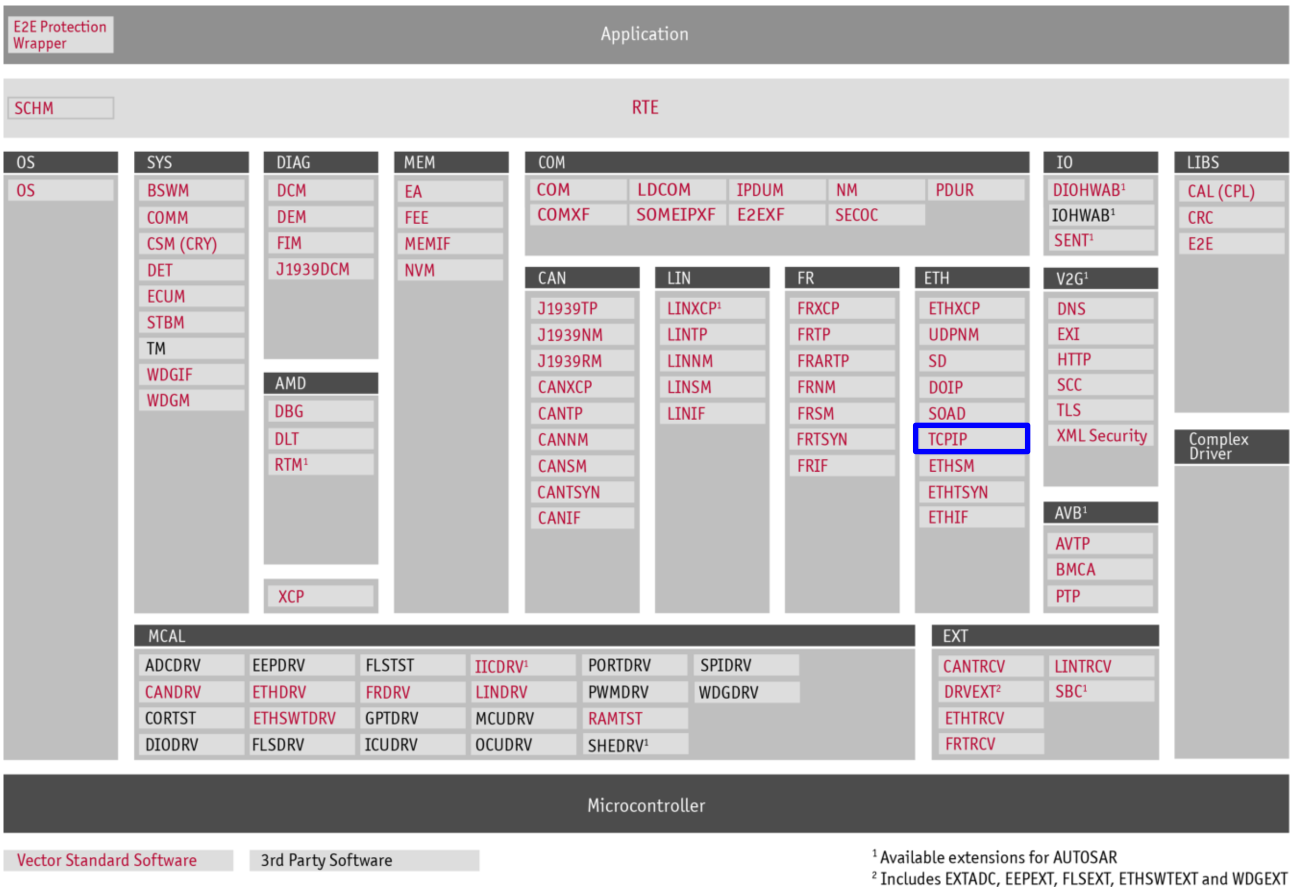


Figure 1-1 AUTOSAR 4.2 Architecture Overview

## 1.2 Transport Layer Security

The Transport Layer Security (TLS) is a protocol, located at layer 5 of the ISO/OSI model, which provides mechanisms for an encrypted and authenticated transmission of data. It is a hybrid encryption protocol which uses a combination of asymmetric and symmetric encryption.

The communication via TLS can be divided into two phases. First, a connection is established in which client and server prove their identity to each other, so called the TLS handshake. Once a trustworthy connection has been established, the data is transferred using a negotiated encryption algorithm.

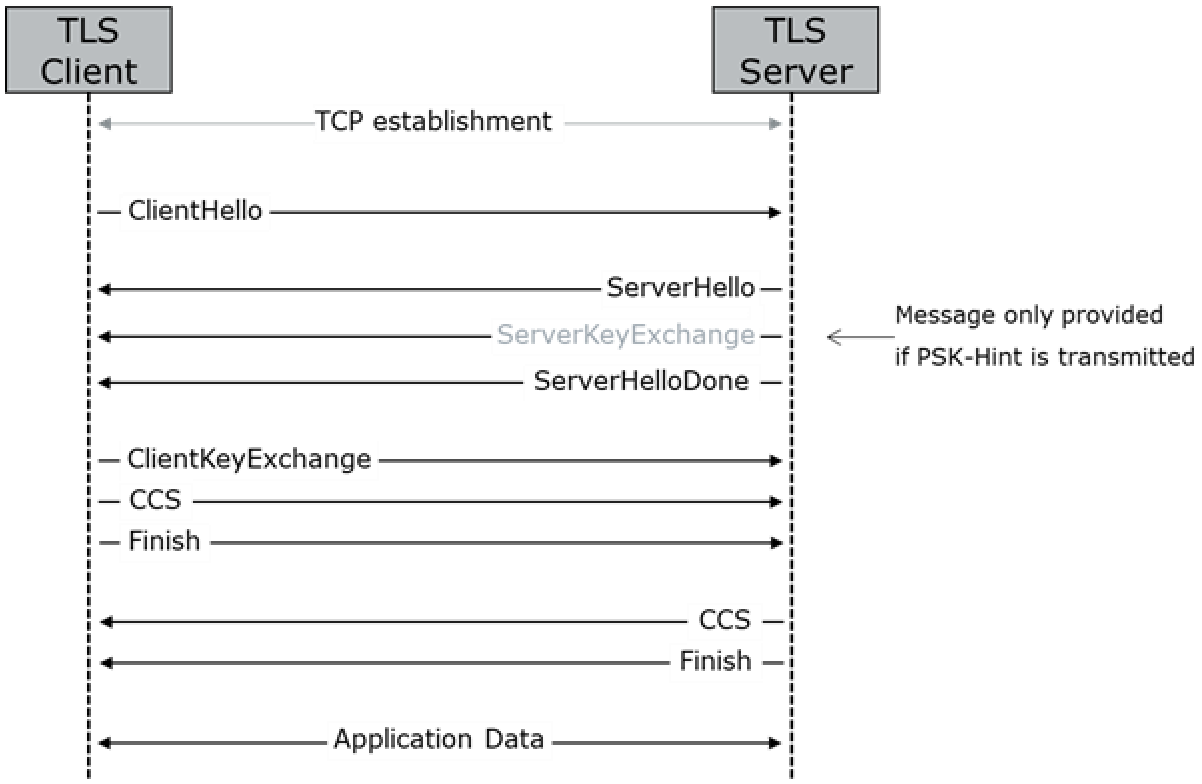


Figure 1-2 Overview of the TLS handshake (with PSK)

## 1.3 Support for Ethernet Multicore

The MICROSAR Classic Ethernet stack can be configured to run on multicore systems. In this mode of operation, a separate instance of TCPIP shall run on each configured core. The TCPIP on each core can handle traffic of one or more VLANs. The following figure shows how the TCPIP instances are mapped on the multicore Ethernet stack.

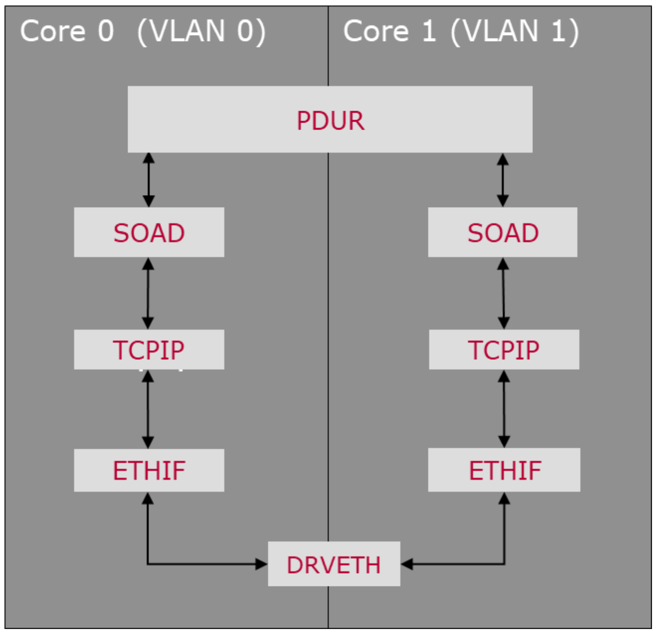


Figure 1-3 Ethernet Multicore Support

# 2 Functional Description

## 2.1 Features

The features listed in the following tables cover the complete functionality specified for the TCPIP.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 2-1 Supported AUTOSAR R4.2.1 standard conform features

> Table 2-2 Supported AUTOSAR R4.3.0 standard conform features

> Table 2-4 Not supported AUTOSAR R4.2.1 standard conform features

Vector Informatik provides further TCPIP functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 2-9 Features provided beyond the AUTOSAR standard

| Supported AUTOSAR R4.2.1 Standard Conform Features |
|---|
| Configuration according to AUTOSTAR R4.2.1 |
| Implementation of TCP and UDP |
| Implementation of IPv4 (incl. ARP, ICMPv4, DHCPv4 client/ server) with multi controller support |
| Implementation of IPv6 (incl. NDP, ICMPv6, DHCPv6 client) with multi controller support |

Table 2-1 Supported AUTOSAR R4.2.1 standard conform features

| Supported AUTOSAR R4.3.0 Standard Conform Features |
|---|
| Support for read and reset of diagnostic measurement data |

Table 2-2 Supported AUTOSAR R4.3.0 standard conform features

| Supported AUTOSAR R20-11 Standard Conform Features |
|---|
| Support for reporting security events to the intrusion detection system manager (IdsM). |

Table 2-3 Supported AUTOSAR R20-11 standard conform features

### 2.1.1 Deviations

The following features specified in [1] are not supported:

| Not supported AUTOSAR R4.2.1 standard conform features | |
|---|---|
| TcpIp | Production error reporting. |
| TcpIp | API TcpIp_ResetIpAssignment |
| TcpIp / TCP | Delayed ACKs. |
| TcpIp / TCP | Simultaneous Open Attempts (IETF RFC 793) (OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| TcpIp / TCP | The following configuration parameters are currently not used: |

| Not supported AUTOSAR R4.2.1 standard conform features | |
|---|---|
| | TcpIpTcpReceiveWindowMax, TcpIpTcpMaxRtx, TcpIpTcpSynMaxRtx, TcpIpTcpSynReceivedTimeout. |
| TcpIp / TCP | Computing of RTT/RTO is according to RFC 6298 instead of RFC1122 and only if time stamp option is enabled on Server as well as on Client. |
| TcpIp / UDP | Forward received multicasts to all matching sockets. |
| IpV4 | Path MTU Discovery for IPv4 (IETF RFC1191). |
| IpV4 / ARP | Extract physical address information from each received IP packet. (SWS_TCPIP_00092) |
| IpV4 / ARP | Remove unused ARP table entries after a specified timeout. (SWS_TCPIP_00091) If no new received entry overwrites the outdated one, the outdated entry is kept in the ARP table and may be used if a new transmission is requested. |
| IpV4 / ICMP | Transmit ICMP messages other than Echo Request / Echo Reply and Destination Unreachable message. Receive ICMP messages other than Echo Request and Echo Reply. Configuration of ICMP message handler. |
| IpV4 / ICMP | API TcpIp_IcmpTransmit |

Table 2-4 Not supported AUTOSAR R4.2.1 standard conform features

### 2.1.2 IETF RFC Conformance

The TcpIp module conforms to the following RFCs published by the Internet Engineering Task Force (IETF) with the noted exceptions.

> RFC referenced by the AUTOSAR SWS TcpIp R4.2.1
> RFC supported beyond the AUTOSAR standard

**Note**
Unless otherwise noted the following parts of the IETF RFCs are not implemented:
> Sections that specify server/router specific functionality
> Sections that are excluded by the AUTOSAR specification

| IETF RFC | Title |
|---|---|
| 768 | User Datagram Protocol |
| 793 | Transmission Control Protocol |
| 813 | Window and Acknowledgement Strategy in TCP |
| 896 | Congestion Control in IP/TCP Internetworks (Nagle) |
| 1122 | Requirements for Internet Hosts -- Communication Layers |
| 1323 | TCP Extensions for High Performance |
| 2018 | TCP Selective Acknowledgment Options |
| 3390 | Increasing TCP's Initial Window |

| IETF RFC | Title |
|---|---|
| 5482 | TCP User Timeout Option |
| 5681 | TCP Congestion Control |
| 6298 | Computing TCP's Retransmission Timer |
| 6582 | The NewReno Modification to TCP's Fast Recovery Algorithm |
| 4301 | Security Architecture for the Internet Protocol |
| 4302 | IP Authentication Header |
| 4543 | The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH |

Table 2-5 List of supported TcpIp IETF RFCs

| IETF RFC | Title |
|---|---|
| 791 | Internet Protocol |
| 826 | An Ethernet Address Resolution Protocol (ARP) |
| 792 | Internet Control Message Protocol (ICMP) |
| 815 | IP Datagram Reassembly Algorithms |
| 894 | A Standard for the Transmission of IP Datagrams over Ethernet Networks |
| 1112 | Host Extensions for IP Multicasting |
| 2131 | Dynamic Host Configuration Protocol (DHCP) / client and server |
| 3925 | Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4) |
| 3927 | Dynamic Configuration of IPv4 Link-Local Addresses |

Table 2-6 List of supported IPv4 IETF RFCs

| IETF RFC | Title |
|---|---|
| 1981 | Path MTU Discovery for IP version 6 |
| 2460 | Internet Protocol, Version 6 (IPv6) Specification |
| 2464 | Transmission of IPv6 Packets over Ethernet Networks |
| 2711 | IPv6 Router Alert Option |
| 3122 | Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification |
| 3315 | Dynamic Host Configuration Protocol for IPv6 (DHCPv6) / client |
| 3810 | Multicast Listener Discovery Version 2 (MLDv2) for IPv6 |
| 4291 | IP Version 6 Addressing Architecture |
| 4429 | Optimistic Duplicate Address Detection (DAD) for IPv6 |
| 4443 | Internet Control Message Protocol (ICMPv6) for the IPv6 Specification |
| 4861 | Neighbor Discovery for IP version 6 (IPv6) |
| 4862 | IPv6 Stateless Address Autoconfiguration |
| 4884 | Extended ICMP to Support Multi-Part Messages |
| 4941 | Privacy Extensions for Stateless Address Autoconfiguration in IPv6 |

| IETF RFC | Title |
|---|---|
| 5095 | Deprecation of Type 0 Routing Headers in IPv6 |
| 5722 | Handling of Overlapping IPv6 Fragments |
| 5942 | IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes |
| 6085 | Address Mapping of IPv6 Multicast Packets on Ethernet |
| 6106 | IPv6 Router Advertisement Options for DNS Configuration |
| 6724 | Default Address Selection for Internet Protocol version 6 (IPv6) |

Table 2-7 List of supported IPv6 IETF RFCs

| IETF RFC | Title |
|---|---|
| 5246 | **RFC5246 – The Transport Layer Security (TLS) Protocol Version 1.2**<br>The MICROSAR Classic TCPIP is compliant to the following sections of the RFC5246 to perform a TLS handshake. This also includes both encrypting and decrypting application data as well as generating and validating the message authentication code (MAC) under the negotiated cryptographic parameters of the TLS handshake.<br>Appendix A.1 (Record Layer Protocol Version) - TLS v1.2<br>Section 6.2 (Record Layer)<br>  Section 6.2.3 (Record Payload Protection)<br>Section 6.3 (Key Calculation)<br>Section 7 (TLS Handshake Protocols)<br>Section 8 (Cryptographic Computations)<br>Section 10 (Application Data Protocol)<br>Appendix D.2 Certificates and Authentication |
| 8422 | **RFC8422 - Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier** |
| 5289 | **RFC5289 - TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)**<br>The MICROSAR Classic TCPIP is compliant to the following necessary sections and extensions of the RFC8422 and RFC5289 to perform a TLS handshake using ECC cipher suites:<br>Section 5.1 (Client Hello Extensions)<br>  Section 5.1.1. (Supported Elliptic Curves Extension)<br>    - secp256r1 {23}<br>    - secp521r1 {25}<br>    - x25519 {29}<br>    - x448 {30}<br>Section 5.2. (Server Hello Extension)<br>Section 5.3. (Server Certificate)<br>Section 5.4. (Server Key Exchange)<br>  Section 5.4.1. (Uncompressed Point Format for NIST Curves)<br>Section 5.7. (Client Key Exchange)<br>Section 5.9. (Elliptic Curve Certificates)<br>Section 6 (Cipher Suites) |

| | |
|---|---|
| 4279 | **RFC4279 - Pre-Shared Key Cipher suites for Transport Layer Security (TLS)**<br>The MICROSAR Classic TCPIP is compliant to the following necessary sections and extensions of the RFC4279 to perform a TLS handshake using PSK cipher suites:<br>Section 2 (PSK Key Exchange Algorithm)<br>Section 2 (PSK, DHE_PSK, and RSA_PSK Key Exchange Algorithms with AES-GCM)<br>Section 3 (Cipher Usage)<br>Section 3.1 (PSK Key Exchange Algorithm with SHA-256/384)<br>Section 5.1 (PSK Identity Encoding)<br>Section 5.2 (Identity Hint)<br>Section 5.3 (Requirements for TLS Implementations) |
| 8449 | **RFC8449 -Record Size Limit Extension for TLS** |
| 8446 | **RFC8446 – The Transport Layer Security (TLS) Protocol Version 1.3**<br>The MICROSAR Classic TCPIP is compliant to the following sections of the RFC8446 to perform a TLS handshake. This also includes both encrypting and decrypting application data under the negotiated cryptographic parameters of the TLS handshake.<br>Section 4.1 (Key Exchange Messages)<br>Section 4.2 (Extensions)<br>Section 4.3 (Server Parameters)<br>Section 4.4 (Authentication Messages)<br>Section 5.1 (Record Layer)<br>Section 5.2 (Record Payload Protection)<br>Section 5.3 (Per-Record Nonce)<br>Section 6 (Alert Protocol)<br>Section 7.1 (Key Schedule)<br>Section 7.3 (Traffic Key Calculation)<br>Section 7.4 ((EC)DHE Shared Secret Calculation)<br>Appendix C.2 Certificates and Authentication |

Table 2-8 List of supported TLS RFCs

### 2.1.3 Additions / Extensions

The following features are provided beyond the AUTOSAR standard:

| Features Provided Beyond the AUTOSAR Standard | |
|---|---|
| API TcpIp_GetLocNetAddr: Get the local IPv4 address | 4.2.52 |
| API TcpIp_GetLocNetMask: Get the local IPv4 network mask | 4.2.53 |
| API TcpIp_GetIpAddrCfgSrc: Get the address configuration source of a local address | 4.2.30 |
| API TcpIp_DiagDataReadAccess: Read access to some internal variables | 4.2.32 |
| API TcpIp_GetDhcpTimeoutInfo: Get the information if a DHCP timeout has occurred | 4.2.29 |
| API TcpIp_GetLocSockAddr: Get the current local socket address for one socket | 4.2.22 |

| Features Provided Beyond the AUTOSAR Standard | |
|---|---|
| API TcpIp_GetRemNetAddr: Get the current remote socket address for one socket | 4.2.20 |
| IPv4 Manual override of any IPv4 local unicast address during runtime | 5.3.1 |
| IpV4 Configuration of static ARP Tables (disable dynamic address resolution) | 5.4.2 |
| IpV4: Static Link Layer Address Resolution | 5.6.6 |
| IpV4/IpV6: Persistence of static IPv4/IPv6 local unicast address | 5.3.1 |
| IpV6: Transmit and reassemble fragmented packets | 5.3.1 |
| IpV6: Inverse Neighbor Discovery | |
| TcpIp / TCP Reception of out-of-order segments | 2.1.3.2.1 |
| TcpIp / TCP Selective Acknowledgement | 2.1.3.2.2 |
| TcpIp: Separate MainFunctions for reception/transmission of user data and state handling. | 2.6 |
| TcpIp: Extended <Up>_CopyTxData-Callback for transmission of data with dynamic length. | 5.3.1 |
| API TcpIp_ClearARCache: Clear the address resolution cache. | 4.2.33 |
| DHCP event callout which is invoked for each received and prior transmission of client DHCP message. | 4.5.1.10 |
| API TcpIp_IpSecSaEntryPairAdd: Insert a pair of SA entries in the Security Association Database | 4.2.55 |
| API TcpIp_IpSecSaEntryPairDelete: Delete a pair of SA entries from the Security Association Database | 4.2.56 |
| API TcpIp_IpSecSaEntryPairPrepareDelete: Prepare deletion of SA entry pair. Afterwards SA entries will only be used for reception of messages. | 4.2.57 |

Table 2-9 Features provided beyond the AUTOSAR standard

#### 2.1.3.1 IPsec

The MICROSAR Classic TCP/IP stack supports IPsec with the features listed in chapters 2.1.5.3.2 and 2.1.5.3.3. It can be used to secure the traffic on the IP layer. Therefore, IPsec contains a configurable list called SPD (Security Policy Database) that describes if and how a connection should be secured. This list could be configured using “DaVinci Configurator 5” under `/TcpIp/TcpIpConfig/TcpIpIpSecConfigSet/TcpIpSpdEntry`.

IPsec can be used to prove authenticity and integrity in the AH (Authentication Header) mode and additional confidentiality with ESP (Encapsulating Security Payload). For AH and ESP, the crypto algorithms need key material that could be provided by the IKE (Internet Key Exchange) protocol. Currently, only AH in transport mode is supported by MICROSAR Classic.

There is a second list called SAD (Security Association Database) that contains the negotiated attributes like algorithm type, key material and other properties. SA entries can be managed using the following TcpIp service functions:

> `TcpIp_IpSecSaEntryPairAdd`

> `TcpIp_IpSecSaEntryPairPrepareDelete`

> `TcpIp_IpSecSaEntryPairDelete`

Additional to the RFC functionalities of IPsec, MICROSAR Classic supports the following functions:

##### 2.1.3.1.1 “Optional” mode for Security Policies

Security policies can set to “Optional” mode. IPsec will secure the message related to this security policy if a fitting SA entry exist. If there is no SA entry in the SAD, then the message will be sent without IPsec protection.

##### 2.1.3.1.2 SAs prepared for deletion

An SA can be set to prepared for deletion by using the service function `TcpIp_IpSecSaEntryPairPrepareDelete`. After that call, this SA can only be used for reception of messages.

##### 2.1.3.1.3 SA Tx Activation Timeout

During (re)negotiation of the SA entries it may happen that the SA entries become active on both ECUs with a small delay between each other. IPsec messages transmitted during this period of time may be discarded. To avoid such a scenario, IPsec SA entries created as IKE responder will initially be used for reception only. The created SA entry pair will also be used for transmission if a corresponding IPsec protected message is received or if a configurable timeout is exceeded. (See 5.10.4 IPsec SA Entry Tx Activation Timeout)

#### 2.1.3.2 TCP Extensions

##### 2.1.3.2.1 TCP Out Of Order Support

For TCP a support for messages received out of order (OOO) can be enabled. If this feature is enabled TCP stores incoming segments even if previous messages are missing. As soon as the missing segments are received the TCP forwards all stored data to the socket owner.

##### 2.1.3.2.2 TCP Selective Acknowledgement Support

In the standard configuration TCP won’t notify the communication partner that it has a gap in its received data sequence. This feature called ‘selective acknowledgement’ (SACK) can be enabled separately (in addition to OOO). Before SACK can be used it has to be agreed during TCP connection setup (using TCP header options).

#### 2.1.3.3 Parameter Checking

##### 2.1.3.3.1 Parameter checking for submodule TcpIp

The following development errors and error codes are reported additionally to the errors defined in [1].

Development errors:

> TCPIP_API_ID_SHUTDOWN

> TCPIP_API_ID_PROVIDE_TX_BUFFER

> TCPIP_API_ID_TRANSMIT_TO

> TCPIP_API_ID_GET_LOC_NET_ADDR

> TCPIP_API_ID_GET_LOC_NET_MASK

> TCPIP_API_ID_SET_DHCP_HOST_NAME
> TCPIP_API_ID_SET_DHCP_V4_TX_OPT
> TCPIP_API_ID_SET_DHCP_V6_TX_OPT
> TCPIP_API_ID_CBK_RX_INDICATION
> TCPIP_API_ID_CBK_TX_CONFIRMATION
> TCPIP_API_ID_CBK_TCP_ACCEPTED
> TCPIP_API_ID_CBK_TCP_CONNECTED
> TCPIP_API_ID_CBK_TCP_IP_EVENT
> TCPIP_API_ID_RX_SOCK_IDENT
> TCPIP_API_ID_TCP_STATE_CHG
> TCPIP_API_ID_LOC_IP_ASS_CHG
> TCPIP_API_ID_PREPARE_SHUTDOWN
> TCPIP_API_ID_TRCV_LINK_STATE_CHG
> TCPIP_API_ID_ADDR_RES_TIMEOUT
> TCPIP_API_ID_SOCK_ADDR_IS_EQ
> TCPIP_API_ID_SOCK_ADDR_IS_EQ_OR_UNSPEC
> TCPIP_API_ID_PATH_MTU_CHG
> TCPIP_API_ID_GET_IP_ADDR_CFG_SRC
> TCPIP_API_ID_GET_IP_ADDR
> TCPIP_API_ID_GET_REM_NET_ADDR
> TCPIP_API_ID_CLEAR_AR_CACHE
> TCPIP_API_ID_SEND_GRATUITOUS_ARP_REQ

Error codes:

> TCPIP_E_INV_SOCK_ID
> TCPIP_API_ID_GET_AND_RESET_MEAS_DATA
> TCPIP_API_ID_IPSEC_ADD_SAENTRY_PAIR
> TCPIP_API_ID_IPSEC_DEL_SAENTRY_PAIR
> TCPIP_API_ID_IPSEC_SPD_CALLOUT

#### 2.1.3.4 TcpIp Support in Multicore Environment

TcpIp supports operation on multicore systems. The feature allows TcpIp controllers to be handled on separate cores, enabling load balancing and other benefits. This feature can be enabled by the integrator during configuration. Please refer to relevant chapters for correct configuration of TcpIp in a multicore environment.

**Caution**
The TcpIp multicore feature must be enabled only in combination with other MICROSAR Classic modules that support multicore operation (SoAd, EthIf, PduR).

For multicore instances, the API TcpIp_<Up>GetSocket must be called on the same partition as the following calls of TcpIp_<Up>Transmit or TcpIp_RxIndication for messages on this socket.

#### 2.1.3.4.1 Unsupported Features in Multicore Environment

The following features are not supported while running on multicore systems, and must not be enabled:

- TLS
- IPsec
- IPv6
- DhcpV4 Server

### 2.1.3.5 IPv6 extension header

IPv6 can only handle a limited set of IPv6 extension headers.

**Not supported are:**

- Authentication Header
- Encapsulation Security Payload
- Mobility
- Host Identity Protocol
- Shim6 Protocol

**Skippable extension headers:**

- Hop-by-Hop Options
- Routing
- Destination Options

**Supported extension headers:**

- Fragmentation

Received packets with not supported extension headers will be discarded.
For the skippable extension headers this can be configurated with `TcpIp/TcpIpConfig/TcpIpCtrl/TcpIpIpVXCtrl/TcpIpIpV6Ctrl/TcpIpDiscardUnknownExtHdr` by default those extension headers will be skipped.
The fragmentation can be enabled and disabled by configuration of the TcpIpIpV6FragmentationConfig of an IPv6 controller.

Transmitted packets did not contain any extension headers, beside the fragmentation extension header if configured.

#### 2.1.3.6 UDP Address matching

TcpIp forwards received packets to the socket which is bound to the best matching IP address if more than one socket matches. This means that sockets which are bound to a IPv4-broadcast (255.255.255.255) or an IPv6-“All nodes” (FF02::1) address are preferred for packets that are sent to a broadcast or an “All nodes” destination. If no exact match exists a socket with another address of the same EthIf-interface can be used.

### 2.1.4 Supported TLS Features

#### 2.1.4.1 Supported TLS-Client Features

##### 2.1.4.1.1 Supported TLS 1.2 Client cipher suites

The following Table 2-10 defines the supported TLS 1.2 Client cipher suites which can be configured accordingly.

| IANA Name | Hex Code |
|---|---|
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 | 0xC025 |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 | 0xC02D |
| TLS_ECDH_ECDSA_WITH_NULL_SHA | 0xC001 |
| TLS_ECDHE_ECDSA_AES_128_CBC_SHA256 | 0xC023 |
| TLS_ECDHE_ECDSA_AES_128_GCM_SHA256 | 0xC02B |
| TLS_ECDHE_ECDSA_WITH_NULL_SHA | 0xC006 |
| TLS_PSK_WITH_AES_128_CBC_SHA | 0x008C |
| TLS_PSK_WITH_AES_128_CBC_SHA256 | 0x00AE |
| TLS_PSK_WITH_AES_128_GCM_SHA256 | 0x00A8 |
| TLS_PSK_WITH_NULL_SHA | 0x002C |
| TLS_PSK_WITH_NULL_SHA256 | 0x00B0 |

Table 2-10 Supported TLS-Client cipher suites

##### 2.1.4.1.2 Supported TLS 1.3 Client cipher suites

The following Table 2-11 defines the supported TLS 1.3 Client cipher suites which can be configured accordingly.

| IANA Name | Hex Code |
|---|---|
| TLS_AES_128_GCM_SHA256 | 0x1301 |
| TLS_AES_256_GCM_SHA384 | 0x1302 |
| TLS_CHACHA20_POLY1305_SHA256 | 0x1303 |

Table 2-11 Supported TLS 1.3 Client cipher suites

##### 2.1.4.1.3 Supported TLS-Client ECC Curves

The following named curves are supported by the TLS-Client for the ECDH(E) key exchange and the ECDSA signature validation:

- SECP256R1 for ECDH(E) and ECDSA
- SECP521R1 for ECDH(E) and ECDSA
- ED25519 for ECDSA
- X25519 for ECDH(E)
- X448 for ECDH(E)

#### 2.1.4.1.4 Supported TLS-Client Extensions

The following TLS-Client Extensions are supported and transmitted via the ClientHello message if configured accordingly. For further configuration notes see chapter 5.12.2.5 TLS Extension on page 174.

- RFC8422 Supported Point Formats Extension
- RFC5246/RFC8422 Signature Algorithms Extension
- RFC8422 Supported Elliptic Curves Extension
- RFC6066 Trusted CA Indication Extension (TLS 1.2 only, cert hash only)
- RFC6066 Certificate Status Request Extension
- RFC6066 Server Name Indication
- RFC8446 Certificate Authorities (TLS 1.3)
- RFC8446 Pre-Shared Key (TLS 1.3 Session Resumption)
- RFC8446 Pre-Shared Key Exchange Modes (TLS 1.3 Session Resumption)

#### 2.1.4.1.5 Support Client Authentication

The TLS client supports the client authentication using client certificates with the following signature and hash algorithms:

- SECP256R1 with SHA256 and ECDSA (TLS 1.2, TLS 1.3)
- SECP521R1 with SHA512 and ECDSA (TLS 1.2, TLS 1.3)
- Ed25519 (Only TLS 1.3)
- Ed448 (Only TLS 1.3)

The client authentication uses a configurable callout to the socket owner to determine the certificates that shall be used / sent by the client. The callout contains a struct with information about the received certificate request, and a struct that is filled by the socket owner with the information about the certificates that shall be sent.

The callout is issued during the TLS connection setup right before sending the client certificates. The callout works asynchronously and is repeated every MainFunction cycle until the socket owner sets the status to READY or NOT_AVAILABLE.

#### 2.1.4.1.6 Changeable order of elements in the ClientHello message

The ClientHello message contains a list of supported cipher suites, supported signature algorithms and supported TLS protocol versions (TLS 1.3 extension). Before establishing a TLS client connection, the socket owner can change the order of these elements in the

ClientHello message or even remove elements from the configured ones. The default element order depends on the configuration.

The change can be done in the context of a special user callback (TcpIpSocketOwner/TcpIpSocketOwnerTlsChangeableCipherName) to the socket owner in preparation of the ClientHello message.

The following elements shall be changeable:

- signature algorithms
- cipher suites

The first supported TLS version (in the supported versions extension) depends on the first cipher suite in the list.

#### 2.1.4.1.7 Compatibility Mode

The TLS 1.3 client supports the compatibility mode if enabled in the configuration. If enabled (TcpIpTlsConnection/TcpIpTlsUseMiddleboxCompatibilityMode), the client sends a session ID in the ClientHello filled with random data and sends a ChangeCipherSpec (CCS) message. This results in a TLS 1.3 connection that looks like a TLS 1.2 session resumption and therefore has less trouble with some old middleboxes on the internet.

The implementation differs slightly from [54]. It ignore CCS before the ServerHello message is received, and it ignore CCS in every state even if a non-empty session ID was received in the ClientHello message.

#### 2.1.4.1.8 Push Button Mode

The TLS client supports a special mode to fulfil a requirement from the ISO15118-20. Here a special handling for private environments is needed. For TLS 1.2 and ISO15118-2 this is handled using the validation callback (TcpIpSocketOwner/TcpIpSocketOwnerTlsValidationResultName) and allowing the establishment of TLS connection even with unknown root certificates.

In the ISO15118-20 this is the same, but additionally the “certificate authorities” extension (TLS 1.3) shall be suppressed in the ClientHello message. Therefore, an extra callback (TcpIpSocketOwner/TcpIpSocketOwnerTlsChangeableCipherName) is implemented to get the information if the push button mode is active before sending the ClientHello message.

### 2.1.4.2 Supported TLS-Server Features

#### 2.1.4.2.1 Supported TLS-Server cipher suites

The following Table 2-12 defines the supported TLS-Client cipher suites which can be configured accordingly.

| IANA Name | Hex Code |
|---|---|
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 | 0xC025 |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 | 0xC02D |
| TLS_ECDH_ECDSA_WITH_NULL_SHA | 0xC001 |
| TLS_ECDHE_ECDSA_AES_128_CBC_SHA256 | 0xC023 |
| TLS_ECDHE_ECDSA_AES_128_GCM_SHA256 | 0xC02B |
| TLS_ECDHE_ECDSA_WITH_NULL_SHA | 0xC006 |
| TLS_PSK_WITH_AES_128_CBC_SHA | 0x008C |
| TLS_PSK_WITH_AES_128_CBC_SHA256 | 0x00AE |
| TLS_PSK_WITH_AES_128_GCM_SHA256 | 0x00A8 |
| TLS_PSK_WITH_NULL_SHA | 0x002C |
| TLS_PSK_WITH_NULL_SHA256 | 0x00B0 |

Table 2-12 Supported TLS-Server cipher suites

#### 2.1.4.2.2 Supported TLS-Server ECC Curves

The following named curves are supported by the TLS-Server for the ECDH(E) key exchange and the ECDSA signature validation:

- SECP256R1 for ECDH(E) and ECDSA
- SECP521R1 for ECDH(E) and ECDSA
- ED25519 for ECDSA
- X25519 for ECDH(E)

#### 2.1.4.2.3 Supported TLS-Server Extensions

The following TLS-Server Extensions are supported and transmitted via the ServerHello message if configured accordingly and the corresponding extensions is offered by the TLS-Client in the ClientHello message. For further configuration notes see chapter 5.12.2.5 TLS Extension on page 174.

- RFC8422 Supported Point Formats Extension
- RFC8449 Record Size Limit Extension

#### 2.1.4.2.4 Support Client Authentication

The TLS server supports the client authentication as specified in RFC5246. In the CertificateRequest message it will add the signature and hash algorithms which are configured for the active cipher (sent in the ServerHello). The following signature and hash algorithms are supported:

- SECP256R1 with SHA256 and ECDSA
- SECP521R1 with SHA512 and ECDSA

According to RFC5246 7.4.6. Client Certificate a client may send an empty certificate message if no matching certificate is available. If the client does not send any certificates, the server MAY at its discretion either continue the handshake without client authentication or respond with a fatal handshake_failure alert.

Currently the TLS-Server has no callout configured to clarify with the upper layer if such a connection should be continued. Thus, an empty certificate is treated as an error case and a fatal handshake_failure alert will be send.

#### 2.1.4.3 TLS 1.3 features

##### 2.1.4.3.1 TLS 1.3 Close Behavior

In TLS 1.3 the close behavior differs compared to the one in TLS 1.2. In TLS 1.3 each side can decide to close its sending side separately. This means that receiving a close_notify alert from the peer does not mean that the ECU/TLS also has to close the connection immediately (as previously defined in TLS 1.2). If receiving the close_notify alert without a TCP FIN flag set, the user is informed about the received close request via the CLOSENOTIFY_RECEIVED event. If the TCP FIN flag is set, the user is informed via the FIN_RECEIVED event. In both cases, the user can decide to proceed with sending data (keep its send side open) or also close the connection from its side.

Note that RFC8446 does not specify any response for a received close_notify alert during the handshake. A close_notify alert should be used to close a connection. Therefore, it should never appear during the handshake. In case this happens, it will always lead to an unsuccessful handshake, since the peer won’t be sending any more TLS messages. There are two scenarios that need to be considered.

The first one is the reception of only a close_notify alert without a TCP FIN flag set. In this case the ECU/TLS responds with an unexpected_message alert and a TCP FIN. It also notifies the upper layer of the closed connection.

The second scenario is that the close_notify is received together with a TCP FIN flag. Since this clearly indicates, that the peer is not willing to continue the communication, the ECU/TLS responds with a TCP FIN flag and closes the connection. It also notifies the upper layer of the closed connection.

The deviation from the separate close behavior for TLS 1.3 (each peer decides, when to close its sending side) in these two scenarios is chosen on purpose. It ensures a specific closing behavior during the handshake since both scenarios would not lead to a successful handshake.

##### 2.1.4.3.2 TLS 1.3 Session Resumption

For specifically configured connections (resumption base connections, see TLS 1.3 Session Resumption Configuration) the session can be resumed on another connection, the so-called resumed connection. This is indicated by the pre_shared_key_exchange_modes extension in the ClientHello of the resumption base connection. After a successful handshake a resumption master secret is calculated. The Server may now send a NewSessionTicket message. A NewSessionTicket before the ClientFinished is send will always result in an unexpected_message alert from the ECU,

which is an intended protocol deviation, see RFC8446 chapter 4.6.1:
"Although the resumption master secret depends on the client's second flight, a server which does not request client authentication MAY compute the remainder of the transcript independently and then send a NewSessionTicket immediately upon sending its Finished rather than waiting for the client Finished."

RFC8446 states in chapter 4.2:
"If an implementation receives an extension which it recognizes and which is not specified for the message in which it appears, it MUST abort the handshake with an "illegal_parameter" alert."
Since a NewSessionTicket is only received if the handshake is already finalized it is not quite clear if this also applies to this message. In this implementation extensions in the NewSessionTicket are ignored. The only exception is when `TcpIpTlsRejectEarlyDataExtensionInTicket` is enabled in the `TcpIpTlsConfig` container a NewSessionTicket with an early_data extension is always rejected / ignored. If a NewSessionTicket with an invalid extension length or a duplicated extension is received the ECU will send a fatal alert and close the connection. However, it is not checked if the extension length is valid for the specific extension. For example, an early_data extension with zero length is not possible in a NewSessionTicket according to RFC8446 chapter 4.2.10. But this implementation will not send an alert in this case if the subsequent extension data is indeed empty.

Be aware that as a synonym for the parameter “ticket” of the NewSessionTicket (RFC8446 chapter 4.6.1) PSK identity is used.

If a valid NewSessionTicket is received the connection can be resumed. The user must request this via TcpIp_ChangeParameter (see 4.2.6 TcpIp_ChangeParameter).

In the resumed connection's ClientHello the TLS 1.3 client offers a PSK handshake via the pre_shared_key and pre_shared_key_exchange_modes extensions.

If the Server accepts this, a PSK handshake is performed without a certificate based authentication. The Certificate, CertificateVerify and CertificateRequest message must therefore be omitted by the Server. The PSK (derived from the resumption master secret and the information received in the NewSessionTicket) is input for the calculation of the early secret.

If the Server does not accept a PSK handshake in its ServerHello, as fallback a "normal" ECC handshake can be performed as well.

### 2.1.5 Limitations

#### 2.1.5.1 Limitations for AUTOSAR APIs

##### 2.1.5.1.1 DHCP write and read option

The APIs to write (TcpIp_DhcpWriteOption and TcpIp_DhcpV6WriteOption) as well as to read DHCP options (TcpIp_DhcpReadOption and TcpIp_DhcpV6ReadOption) can only be used to write and read statically configured options. It is not possible to read out or write “unknown” options during runtime.

The APIs are limited to:

> Fully Qualified Domain Name (FQDN) options.

> Statically configured DHCP user options.

#### 2.1.5.1.2 Triggering IP address request

The request of IP address assignment is configurable, but only a configuration subset is supported. All address assignments for one IpAddrId need to have the same configuration.

#### 2.1.5.1.3 Update of stored certificates

For certificate-based cipher suites, the TLS server sends a stored certificate chain to the client to prove its identity. These certificates and the corresponding private key of the server are stored in the CSM and may not be updated while a TLS listen socket (that uses these certificates) is open. This is to prevent inconsistencies between these elements when a TLS handshake is performed.

### 2.1.5.2 Limitations of AUTOSAR TLS implementation

#### 2.1.5.2.1 L002 – Record Layer message size limit negotiation

The Record Layer message size limitation (TLS Extension RFC8449) is only available for the TLS-Server implementation.

#### 2.1.5.2.2 L003 – PSK Identity Key sizes

The current implementation restricts the PSK Identity and PSK hint to the following maximum sizes:

| Parameter | Description | Size<br>min … max |
|---|---|---|
| TcpIpTlsPresharedKeyCsmKeyRef | CSM Key for the pre shared key | 1…64 Bytes |
| TcpIpTlsPresharedKeyIdentity | PSK Identity string | 1…128 Bytes |
| TcpIpTlsPresharedKeyIdentityHint | PSK Hint string | 1…128 Bytes |

Table 2-13 PSK Size limitations

#### 2.1.5.2.3 L004 – PSK Callout functions

The current implementation does not support the PSK callout function, in which the user can set either one of the following parameters during runtime. Only static configured values are supported.

| Parameter | Description |
|---|---|
| TcpIpTlsPskGetClientKeyIdentityFunc | Up_TlsClientGetPskIdentity() |
| TcpIpTlsPskGetKeyIdentityHintFunc | Up_TlsServerGetPskIdentityHint() |
| TcpIpTlsPskGetServerKeyIdentityFunc | Up_TlsServerGetPskIdentity() |

Table 2-14 PSK callout limitations

#### 2.1.5.2.4 L005 – Server Name Indication

The SNI (Server Name Indication) extension (RFC 6066) is only available for the TLS-Client implementation.

#### 2.1.5.2.5 L006 – Encrypt-then-MAC Extension

The current implementation does not support the Encrypt-then-MAC extension. The following parameter is not used: TcpIpTlsUseSecurityExtensionForceEncryptThenMac

#### 2.1.5.2.6 L007 – Runtime TLS handshake Selection

The current implementation does not support the runtime selection of the TLS handshake reference in an TLS-Client connection. See chapter Selection of handshake reference during runtime.

#### 2.1.5.2.7 L008 – KeyUpdate message

The current implementation does not support the KeyUpdate message. In case a KeyUpdate message is received, an unexpected_message alert is sent, and the connection is closed.

#### 2.1.5.2.8 L010 – User_Canceled Alert

The current implementation does not support the alert “user_canceled”. If this alert (with level warning) is received, a fatal error “internal_error” is sent. If the socket owner closes the socket while the TLS is in the handshake, the socket is closed on TCP level, without sending any TLS alert.

#### 2.1.5.2.9 L011 – No RSA Support

The current implementation does not include any RSA support, neither for key exchange nor for signature algorithms.

### 2.1.5.3 RFC specific deviations of submodule TcpIp

#### 2.1.5.3.1 RFC4301 Security Architecture for the Internet Protocol

| Section in Document | Limitation |
|---|---|
| 4.4.1.1. Selectors | ICMP is not supported as a selector |

Table 2-15 Deviations from IETF RFC4301

#### 2.1.5.3.2 RFC4302 IP Authentication Header

| Section in Document | Limitation |
|---|---|
| 3. Authentication Header Processing | Feature is only supported in combination with IPv4 controllers. |
| 3.1.2 Tunnel Mode | Tunnel mode is not supported. |
| 3.3.4 Fragmentation | Authentication of fragmentated packet is not supported. |

| Section in Document | Limitation |
|---|---|
| 4.4.1 The Security Policy Database (SPD) | An additional Policy OPTIONAL is introduced. |

Table 2-16 Deviations from IETF RFC4302

#### 2.1.5.3.3 RFC4543 The Use of GMAC in IPsec ESP and AH see [49]

| Section in Document | Limitation |
|---|---|
| All Sections related to ESP | ESP Header is not supported |

Table 2-17 Deviations from IETF RFC4301

### 2.1.5.4 RFC specific deviations of submodule IpV4

#### 2.1.5.4.1 RFC791 Internet Protocol [9]

| Section in Document | Limitation |
|---|---|
| Page 15 / IPv4 Header Options | IP options may appear in received packets but will be silently ignored. |
| Page 27 / Discussion (An Example Reassembly Procedure) | No support for adapting Reassembly-Timeout via TTL in received packet (OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| Page 29 / Discussion (An Example Reassembly Procedure) | IP Fragment overlapping is not supported. (OPEN_ALLIANCE_TC8_TEST_RELEVANT) |

Table 2-18 Deviations from IETF RFC791

#### 2.1.5.4.2 RFC792 Internet Control Message Protocol (ICMP) [10]

| Section in Document | Limitation |
|---|---|
| ICMP Messages | ICMP message support is limited to Echo Request, Echo Reply and transmission of Destination Unreachable messages.<br>The length of the data mirrored in the Echo Reply message is limited (configurable). |
| Page 8 / Parameter Problem Message | No support for ICMPv4 Parameter Problem Message. (OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| Page 16 / Timestamp or Timestamp Reply Message | No support for ICMPv4 Timestamp or Timestamp Reply message. (OPEN_ALLIANCE_TC8_TEST_RELEVANT) |

Table 2-19 Deviations from IETF RFC792

#### 2.1.5.4.3 RFC1323 TCP Extensions for High Performance [17]

| Section in Document | Limitation |
|---|---|
| 2. “TCP Window Scale Option” | Not supported. |
| 4. “PAWS -- Protect Against Wrapped Sequence Numbers” | Not supported. |

Table 2-20 Deviations from IETF RFC1323

#### 2.1.5.4.4 RFC2131 Dynamic Host Configuration Protocol (DHCP) [20]

| Section in Document | Limitation |
|---|---|
| General | Basic DHCP client and server support implemented. |
| 3.2 Client-server interaction | "Client-server interaction - reusing a previously allocated network address" is not implemented.<br>DHCPv4 client does not skip sending of DISCOVER messages but it is possible to request a particular IP address by using the <User>_RequestedDhcpAddrCallout. |
| Probing/Announcing of Addresses | IP addresses assigned by DHCP are not probed by the IP-Module to be unique. Once the address is assigned, it will not be announced by sending broadcast ARP reply.<br>(OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| 4.3.5 DHCPINFORM message<br>4.3.4 DHCPRELEASE message<br>4.3.3 DHCPDECLINE message | DHCPINFORM, DHCPDECLINE and DHCPRELEASE messages are not sent by the client and are ignored by the DHCP server.<br>(OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| 3.5 Client parameters in DHCP | DHCPv4 client does not send DHCP message option 51 IP Address Lease Time.<br>(OPEN_ALLIANCE_TC8_TEST_RELEVANT) |
| 4.1 Constructing and sending DHCP messages | No support for DHCP option 52 Overload Option.<br>(OPEN_ALLIANCE_TC8_TEST_RELEVANT) |

Table 2-21 Deviations from IETF RFC2131

#### 2.1.5.4.5 RFC4702 The DHCP Client FQDN option [31]

| Section in Document | Limitation |
|---|---|
| General | The 'Flags' field is filled with fixed values. No DNS update handling is implemented in any way. |

Table 2-22 Deviations from IETF RFC4702

### 2.1.5.5 RFC specific deviations of submodule IpV6

#### 2.1.5.5.1 RFC2460 Internet Protocol, Version 6 (IPv6) Specification [21]

| Section in Document | Limitation |
|---|---|
| 4. "IPv6 Extension Headers" | IPsec / "Authentication" and "Encapsulating Security Payload" (ESP) Headers are not supported. |

Table 2-23 Deviations from IETF RFC2460

#### 2.1.5.5.2 RFC3315 Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [25]

| Section in Document | Limitation |
|---|---|
| 22.4. "Identity Association for Non-temporary Addresses Option" | At most one DHCPv6 address can be configured on each controller. |

Table 2-24 Deviations from IETF RFC3315

##### 2.1.5.5.3 RFC3484 Default Address Selection for Internet Protocol version 6 (IPv6) [46]

| Section in Document | Limitation |
|---|---|
| 5. “Source Address Selection” | Rule 4: Prefer home addresses. (no Mobile IP support)<br>Rule 6: Prefer matching label. |

Table 2-25 Deviations from IETF RFC3484

##### 2.1.5.5.4 RFC4291 IP Version 6 Addressing Architecture [28]

| Section in Document | Limitation |
|---|---|
| 2.5.1. “Interface Identifiers” | Only 64 bit interface identifiers are supported: “For all unicast addresses, except those that start with the binary value 000, Interface IDs are required to be 64 bits long and to be constructed in Modified EUI-64 format.” |

Table 2-26 Deviations from IETF RFC4291

##### 2.1.5.5.5 RFC4443 Internet Control Message Protocol (ICMPv6) for the IPv6 Specification [30]

| Section in Document | Limitation |
|---|---|
| 3.1. “Destination Unreachable Message” | The IpV6 will not send Destination Unreachable Messages. (ICMP Message Type 1) |

Table 2-27 Deviations from IETF RFC4443

##### 2.1.5.5.6 RFC4704 The DHCPv6 Client FQDN option [32]

| Section in Document | Limitation |
|---|---|
| General | The ‚Flags‘ field is filled with fixed values. No DNS update handling is implemented in any way. |

Table 2-28 Deviations from IETF RFC4443

##### 2.1.5.5.7 RFC4941 Privacy Extensions for Stateless Address Autoconfiguration in IPv6 [36]

RFC4941 “Privacy Extensions for Stateless Address Autoconfiguration” is currently not supported in MICROSAR Classic releases.

##### 2.1.5.5.8 RFC3810 Multicast Listener Discovery Version 2 (MLDv2) for IPv6 [26]

RFC3810 “Multicast Listener Discovery Version 2 (MLDv2) for IPv6” is currently not supported in MICROSAR Classic releases.

### 2.1.6 Known Issues (low priority)

This section lists known issues that are not reported by the Vector issue reporting because they have low priority.

**Compiler Warning “conditional expression is constant”**

Some compilers may report warnings about conditional expressions that always evaluate to TRUE. These warnings are configuration dependent. If a TCPIP feature is enabled but not used by all IP instances/controllers a runtime check is required in order to decide whether the feature is enabled on a specific controller. If the feature is enabled on all controllers the condition is replaced by the constant value TRUE, so the compiler can eliminate the runtime check.

If a feature is disabled on all controllers, the code will be completely excluded from compilation.

### 2.1.7 Known Issues (IETF RFC Conformance)

The following items describe behavior of the Vector TCP/IP stack that slightly deviates from the behavior defined by the considered IETF RFCs.

These deviations have been identified using IETF RFC conformance tests.

Although this behavior may reduce TCP/IP performance in a few cases, it should not cause operational issues in currently known use cases.

**IPv6 sends ICMP error messages in response to ICMP error messages**

According to RFC4443 the TCPIP must not send an ICMPv6 error message as a result of receiving an ICMPv6 error message. But the IpV6 will do this if the received ICMPv6 error message is corrupt before the ICMPv6 header of the received error message.

MSR4-33197

**IPv6 stops reassembly of packet if the same fragment is received twice**

According to IETF RFC 5722 IPv6 shall stop reassembly of a packet if two fragments overlap. But the RFC requirements are unclear about what shall be done if the same fragment is received twice. The IPv6 will also stop reassembly of a packet if a fragment is received twice.

MSR4-33196

**TCP backs off retransmission timer on fast retransmit**

When a “Fast Retransmit” according to IETF RFC5681 has been sent, TCP will back-off the timer for the next regular retransmission.

MSR4-33193

**TCP uses too big cwnd if SYN or SYNACK is lost during active open**

If a SYN/ACK segment gets lost during connection establishment, TCP assumes a bigger congestion window (CWND) than specified in IETF RFC5681. TCP will still respect the receive window of the remote host.

MSR4-33191

### TCP retransmits more than one segment after retransmit timeout expired

If the retransmit timer expires TCP may send more than one unacknowledged segment. The congestion window (CWND) is reduced as specified by IETF RFC6298.

MSR4-33190

### TCP does not reply timestamp of earliest unacknowledged segment

If multiple TCP segments with different timestamp values are ACKed by the TCP the most recent timestamp value instead of the oldest value will be replied.

MSR4-33194

### Limitations of TCP User Timeout Option (RFC5482)

The user timeout value for a TCP socket will only be used if it is set after connection establishment.

Retransmitted segments do not contain a user timeout option.

Received user timeout values will only be considered while the connection is in an established state.

MSR4-33192

### TCP sends window update with a value less than one MSS

TCP does not wait until the window can hold at least one MSS before sending a window update. This may affect TCP performance if very small chunks of data are received.

MSR4-33189

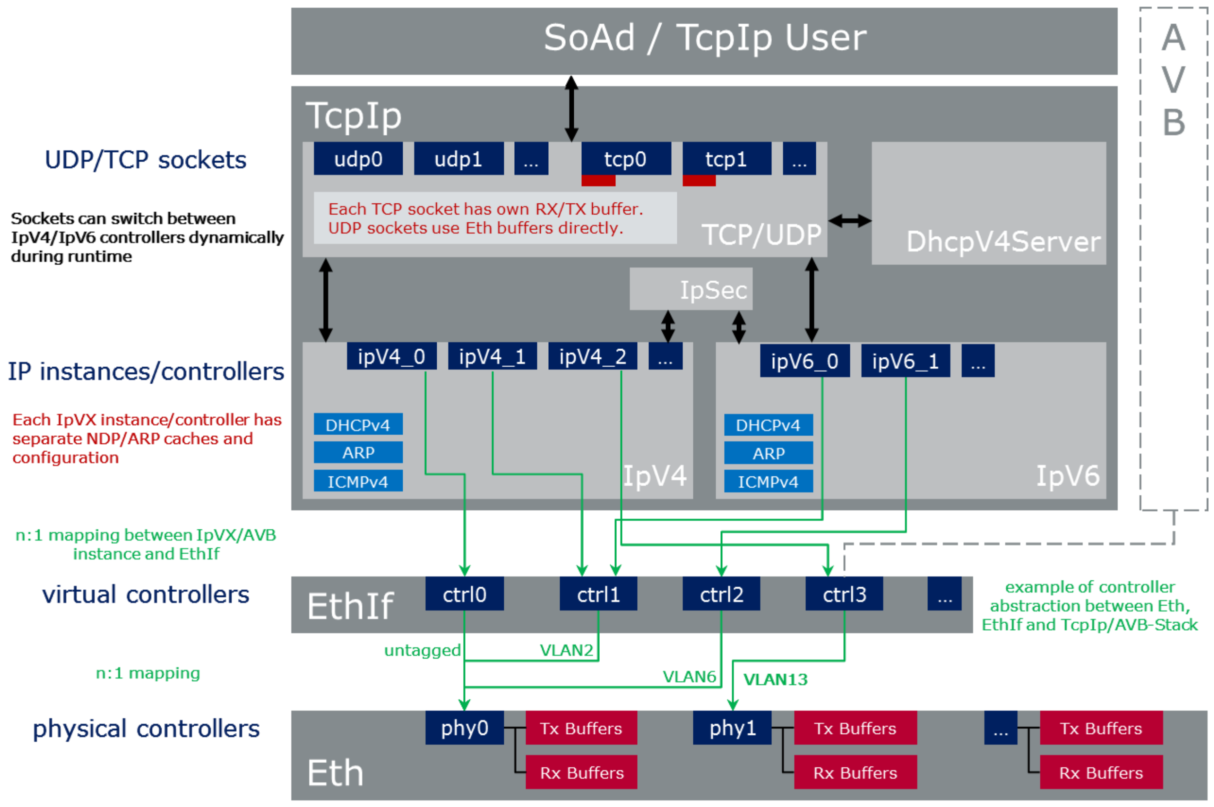
### TCP sends PSH flag in each packet and ignores URG flag

Urgent data is not supported by TcpIp. Therefore, any packet with the 'URG' flag set will be dropped. The 'PSH' flag is set in every outgoing TCP packet. (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

### TCP calculates RTO and RTT as specified in RFC 6298 but with deviation

RTT is only calculated and used for RTO calculation if Time stamp option is enabled both on client and server. Otherwise only configured amount of RTO is used for first retransmission of the Packet. (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

## 2.2 Structure of the TcpIp Stack



The TCPIP module provides UDP and TCP sockets to the upper layer (user).

A socket can be opened by the user during runtime via the API TcpIp_<Up>GetSocket by specifying the socket protocol (UDP/TCP) and the Internet Protocol version (IPv4/INET or IPv6/INET6).

After a socket has been opened it must be bound to a local IP address and port via the API TcpIp_Bind(). Local IP addresses are represented by identifiers (LocalAddrIds). A LocalAddrId represents an address of a specific IP instance/controller and is not shared between IP instances. IP instances operate independent from each other and use their own data structures like ARP/NDP caches and addresses.

Each IP instance/controller operates on exactly one virtual controller provided by the EthIf module.

**Expert Knowledge**
A virtual controller of the EthIf represents a physical Eth controller or a VLAN on a physical Eth controller.

### 2.2.1 IpV6

Each IPv6 instance has at least an auto configured link-local unicast address (see [34] 5.3. Creation of Link-Local Addresses) and an all-nodes multicast address. Additional unicast and multicast addresses may be configured.

### 2.2.2 IpV4

Each IPv4 instance has exactly one unicast and one broadcast address. Additionally, one or more multicast addresses may be configured.

### 2.2.3 TLS

The TCPIP module implements a TLS 1.2 server and client and a TLS 1.3 client. Not all features are implemented for both protocol version or for both roles.

#### 2.2.3.1 Behavior of MICROSAR Classic TCPIP with multiple PSK Identities (TLS 1.2)

The following chapter give some detailed information of the behavior of the MICROSAR Classic TCPIP stack in case multiple PSK identities are configured for one individual TLS connection.

**Generic Assumptions:**

The AUTOSAR model does not include any kind of priority for the PSK Identity container. This could lead to the issue, that in case one TLS connection refers to multiple PSK Identities, the TLS stack could not select a PSK Identity. Therefore, Vector introduces an additional Parameter `TcpIpTlsConnectionPskDefaultIdentityRef`, so called Default PSK Identity. The meaning of this parameter is described in the next section.

The `TcpIpTlsConnectionPskDefaultIdentityRef` parameter can also exists in the `TcpIpTlsConnectionPskIdentityRef`.

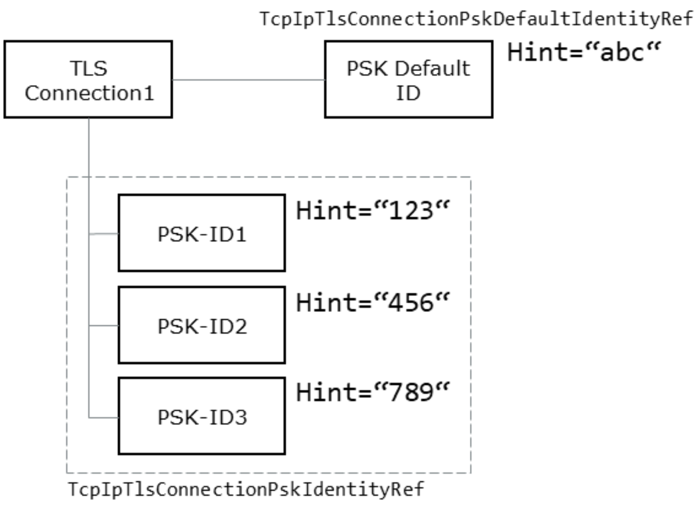


Figure 2-1 Example PSK configuration with multiple PSK Identities.

##### 2.2.3.1.1 TLS Client

The `TcpIpTlsConnectionPskIdentityRef` parameter is for the client only relevant if he must process a received PSK Hint.

The PSK Hint, which is sent by the TLS Server with the *ServerKeyExchange* message, will be compared to all configured PSK Identity Hints on this connection. If there is a match between the received PSK Hint and a configured PSK Hint, the TLS Client will select this match (case a). If none of the configured PSK Hints matches with the received one, the TLS Client uses the `TcpIpTlsConnectionPskDefaultIdentityRef` parameter (case b). If no *ServerKeyExchange* is received, then the TLS client will always choose the `TcpIpTlsConnectionPskDefaultIdentityRef` parameter.

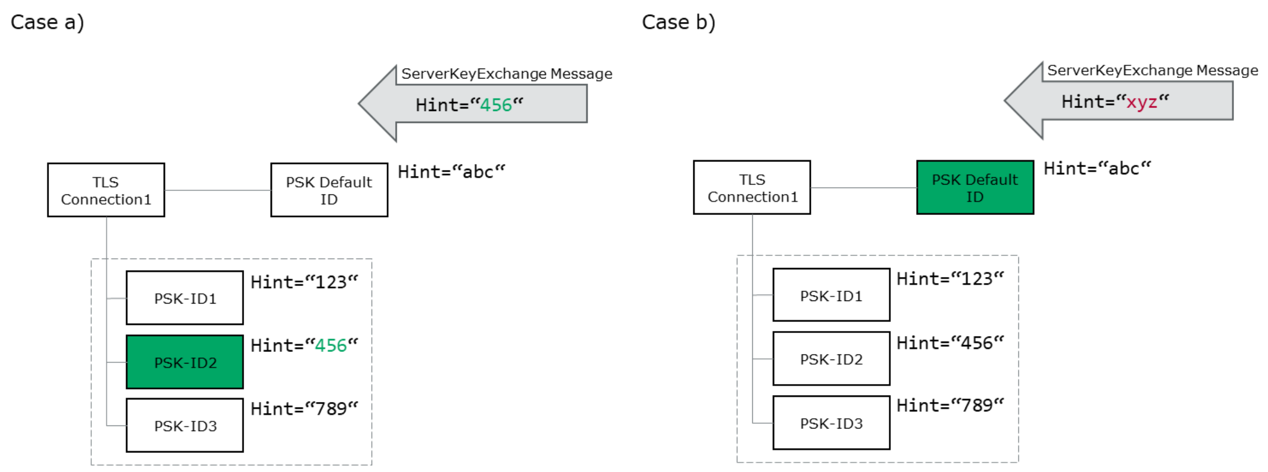


Figure 2-2 Selection of PSK Identity.

### 2.2.3.1.2 TLS Server

If the Parameter `TcpIpTlsConnectionPskDefaultIdentityRef` contains a configured PSK Identity Hint, the TLS server will send this hint in the *ServerKeyExchange* message. Otherwise, the *ServerKeyExchange* message is omitted.

The client selects a PSK Identity and sends this identity in the *ClientKeyExchange* message. The identity does not have to match with the PSK Identity Hint provided by the TLS Server. Thus, for the TLS server the `TcpIpTlsConnectionPskIdentityRef` parameter is always relevant. It is used to select a matching PSK Identity according to the received PSK Identity from the *ClientKeyExchange* message.

If no configured PSK Identity matches the received identity from the *ClientKeyExchange* message, the TLS Server sends a fatal alert of type '*decrypt_error*'.

### 2.2.3.2 Mapping between cipher suite and AUTOSAR parameter (TLS 1.2)

Figure 2-3 shows the mapping between the individual parts of a cipher suite and corresponding AUTOSAR parameter. For example, the key exchange mechanism during the TLS handshake and the de-, and encryption as well as authentication of the TLS stream (Application data).

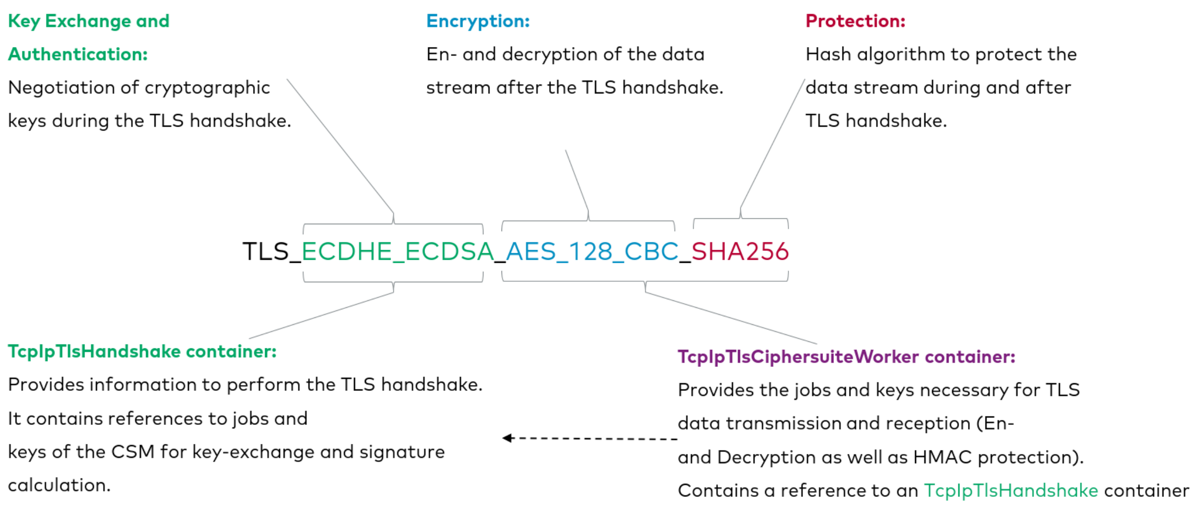

Key Exchange and Authentication:
Negotiation of cryptographic keys during the TLS handshake.
Encryption:
En- and decryption of the data stream after the TLS handshake.
Protection:
Hash algorithm to protect the data stream during and after TLS handshake.
TLS_ECDHE_ECDSA_AES_128_CBC_SHA256
TcpIpTlsHandshake container:
Provides information to perform the TLS handshake. It contains references to jobs and keys of the CSM for key-exchange and signature calculation.
TcpIpTlsCiphersuiteWorker container:
Provides the jobs and keys necessary for TLS data transmission and reception (En- and Decryption as well as HMAC protection). Contains a reference to an TcpIpTlsHandshake container

Figure 2-3 Assignment of Cipher suite to ECUC Container

### 2.2.3.3 User error reporting

Each TLS connection has the possibility to store occurred errors during the runtime in a so called '*UserErrorBuffer'*. These buffers are implemented as a stack, which means that the first detected error is stored on the lowest buffer index. Most of the time one detected issue leads to multiple other issues, which then are stored on the next logical index, as shown in Figure 2-4. The type of each '*UserError'* is defined by Table 2-29. The Values of the '*FunctionId'* and '*ErrorId'* can be found at '*TcpIp_Tls_Types.h'.*

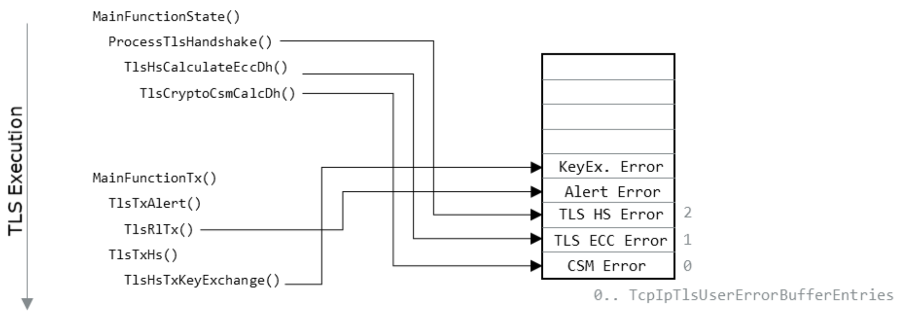


Figure 2-4 TLS User error reporting example

To inform the user (upper layer) over the detected and stored entries, a callback function must be configured with the `TcpIpTlsUserErrorCalloutFunction` parameter, which will then be called within the next main function cycle of the TcpIp. If the callout is invoked, the user can read the entries with the API `TcpIp_Tls_GetUserError` as described in Table 2-30.

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| TcpIp_TlsUserErrorType | struct | User error type | `FunctionId` API identifier |
| | | | `ErrorId` Error identifier |
| | | | `SocketId` Socket identifier |

Table 2-29 TlsUserErrorType

| **Prototype** | |
|---|---|
| `Std_ReturnType TcpIp_Tls_GetUserError( TcpIp_SocketIdType SocketId, TCPIP_P2V(TcpIp_TlsUserErrorType) UserErrorMemoryPtr, TCPIP_P2V(uint32) NumOfUserErrorPtr);` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource for which the user error entries shall be read. |
| UserErrorMemoryPtr [in/out] | Pointer to the destination memory where the TLS Stack should copy the user entries |
| NumOfUserErrorPtr [in] | Size of the destination memory (in user entries count) |
| **Return code** | |
| Std_ReturnType | E_OK if reading is successful<br>E_NOT_OK otherwise. |
| **Functional Description** | |
| API to read the occurred and stored user error of a specific TLS connection. | |
| **Particularities and Limitations** | |
| This is a Vector extension. | |
| Call context | |
| task level | |

Table 2-30 API TcpIp_Tls_GetUserError

### 2.2.3.4 Handling of Alerts with Level Warning

Errors with an alert level of warning indicate that an error occurred, that the sending endpoint is willing to accept. RFC5246 states that in case of an error alert with an alert level of warning, the receiving endpoint can decide whether to continue or to close the connection:

```
"If an alert with a level of warning is sent and received, generally
the connection can continue normally. If the receiving party decides
not to proceed with the connection […], it SHOULD send a fatal alert
to terminate the connection."
```

The TLS implementation for both TLS client and TLS server sends a fatal alert of the type 'internal_error' in reaction to a received error alert of level warning. The 'internal_error' alert is chosen, because the RFC5246 does not specify which alert should be used and the type 'internal_error' is the most fitting error alert.

### 2.2.3.1 Handling of errors during encryption of TLS packets

If a CSM (CSM - Crypto Service Manager) crypto operation fails during encryption or generating the HMAC, i.e. CSM-API does not return E_OK (see also 5.12.6.1.2), TLS implementation triggers an abort on TCP level. It is not possible to close the connection on

TLS layer (e.g. with a fatal alert) due restrictions in the protocol that every alert sent at this point in time must be encrypted and authenticated as well. Since the CSM is in an invalid state we cannot encrypt further messages.

**Caution**
The implementation triggers TCP to send a RST flag which results in an immediate closure of the connection.

### 2.2.3.2 DEPRECATED: Master Secret Access (TLS 1.2)

**Note**
This API is deprecated. Use Tls Secret Access instead.

If the parameter `TcpIpTlsSupportMasterSecretAccess` is enabled, the user has the possibility to read security related parameters of a specific TLS connection. The API grants access to the master secret, the client random value and the session id. These parameters are relevant to encrypt and decode the TLS connection, for example to debug the TLS stream.

**Caution**
If this API is enabled the cryptographic parameter are not securely stored anymore. This API should never be active in production.

If this parameter is enabled, the relevant Crypto key elements must be set to RA_ALLOWED, otherwise no read access to the elements is possible and the API call will fail.

| Relevant Crypto Key Elements |
|---|
| CryptoKeyElement_Tls_MasterSecret |
| CryptoKeyElement_Tls_PrfOutputValuePrivate |
| CryptoKeyElement_Tls_PrfResultPrivate |

Table 2-31 Relevant Crypto Key Elements for Master Secret Access.

| Prototype |
|---|
| `Std_ReturnType` **`TcpIp_Tls_GetMasterSecret(`** |
| `  TcpIp_SocketIdType SocketId,` |

```
  TCPIP_P2V(uint8)   MasterSecretPtr,
  TCPIP_P2V(uint8)   ClientRandomPtr,
  TCPIP_P2V(uint8)   SessionIdPtr,
  TCPIP_P2V(uint8)   SessionIdLenPtr);
```

| Parameter | |
|---|---|
| SocketId [in] | Socket identifier of the related local socket resource. |
| MasterSecretPtr [out] | Pointer to the user buffer, where Master Secret Key shall be stored.<br>**CONSTRAINT**: The pointed user buffer must have the size TCPIP_TLS_MASTER_SECRET_LEN |
| ClientRandomPtr [out] | Pointer to the user buffer, where Client Random Number shall be stored.<br>**CONSTRAINT**: The pointed user buffer must have the size TCPIP_TLS_RANDOM_LEN. |
| SessionIdPtr [out] | Pointer to the user buffer, where Session Id shall be stored.<br>**CONSTRAINT**: The pointed user buffer must have the size TCPIP_TLS_SESSIONIDMAX_LEN. |
| SessionIdLenPtr [out] | Pointer to the Session Id length. |
| **Return code** | |
| Std_ReturnType | E_OK If reading was successful.<br>E_NOT_OK If SocketId was invalid or CSM call failed. |
| **Functional Description** | |
| API to read the occurred and stored user error of on specific TLS connection. | |
| **Particularities and Limitations** | |
| This is a Vector extension. | |
| Call context | |
| task level | |

Table 2-32 API TcpIp_Tls_GetMasterSecret

### 2.2.3.3 TLS Secret Access (TLS 1.2 and TLS 1.3)

If the parameter `TcpIpTlsSupportSecretAccess` is enabled, the user has the possibility to read security related parameters of a specific TLS connection. If the used cipher worker uses TLS version 1.2, the API grants access to the master secret, the client random value and the session id. In case of TLS 1.3, the API allows reading the client/server handshake traffic secrets, the client/server application traffic secrets, and the client random value. These parameters are relevant to encrypt and decode the TLS connection, for example to debug the TLS stream.

The API always reads the secrets which are currently used for encryption/decryption at the calling time. Therefore, handshake traffic secrets can only be obtained if the API is invoked during the handshake. Reading of the application traffic secrets is only possible after a successfully established TLS connection. See TcpIpTlsUseKeyUpdateCallout for more information.

**Caution**
If this API is enabled the cryptographic parameter are not securely stored anymore. This API should never be active in production.

If this parameter is enabled, the relevant Crypto key elements must be set to RA_ALLOWED, otherwise no read access to the elements is possible and the API call will fail.

| Relevant Crypto Key Elements |
|---|
| CryptoKeyElement_Tls_MasterSecret |
| CryptoKeyElement_Tls_PrfOutputValuePrivate |
| CryptoKeyElement_Tls_PrfResultPrivate |
| Crypto_30_LibCv_KeyDerivation_Password |

Table 2-33 Relevant Crypto Key Elements for TLS Secret Access

**Prototype**

```
Std_ReturnType TcpIp_Tls_GetTlsSecrets(
  TcpIp_SocketIdType                 SocketId,
  TCPIP_P2C(TcpIp_Tls12_Secret_Type) Tls12_SecretPtr,
  TCPIP_P2C(TcpIp_Tls13_Secret_Type) Tls13_SecretPtr);
```

| Parameter | |
|---|---|
| SocketId [in] | Socket identifier of the related local socket resource. |
| Tls12_SecretPtr [out] | Pointer to the struct which holds the user buffer, where Master Secret Key, the Client Random value and the Session ID (incl. length) shall be stored.<br>**CONSTRAINT**: The pointed user buffer of Master Secret Key must have the size TCPIP_TLS_MASTER_SECRET_LEN<br>**CONSTRAINT**: The pointed user buffer for the client random value must have the size TCPIP_TLS_RANDOM_LEN.<br>**CONSTRAINT**: The pointed user buffer for the session id must have the size TCPIP_TLS_SESSIONIDMAX_LEN. |
| Tls13_SecretPtr [out] | Pointer to the struct which holds the user buffer, where Client Traffic Secret, the Server Traffic Secret and the Client Random value shall be stored.<br>**CONSTRAINT**: The pointed user buffer of Client/Server Secret Key must have the size TCPIP_TLS_MAX_SIZE_OF_TLS_SECRET<br>**CONSTRAINT**: The pointed user buffer for the client random value must have the size TCPIP_TLS_RANDOM_LEN. |

**Return code**

Std_ReturnType

**Functional Description**

API to read the TLS secrets used for encryption/decryption of the TLS connection at the point in time where the API is called.

| Particularities and Limitations |
|---|
| This is a Vector extension. |
| **Call context** |
| task level |

Table 2-34 API TcpIp_Tls_GetTlsSecrets

#### 2.2.3.4 Deletion of keys when closing a TLS 1.3 socket

When a socket / connection is closed, all related keys and secrets are deleted. The keys and secrets to be deleted depend on the state of the connection (in handshake, established) and the type of handshake ((EC)DHE, PSK).

This feature does not have any visible effect on the TLS connection but only reduces the risk that key material can be read after a connection has been closed.

### 2.2.4 DHCPv4 Server

In addition to a DHCPv4 client, the TcpIp optionally supports a DHCPv4 server according to IETF RFC2131 [20].

Independent DHCPv4 server instances can be configured for each IpV4 controller/VLAN.

According to AUTOSAR a DHCPv4 server instance optionally supports the assignment of IPv4 addresses based on the Ethernet Switch Port information. In this case the address assignment configuration is selected depending on the EthIf Switch Port on which the client DISCOVER/REQUEST message was received.

If no matching Ethernet Switch Port dependent address assignment configuration is found or configured a default address assignment configuration can be used.

**Configuration in DaVinci Configurator Pro**
The **TcpIpDhcpServerConfig** must be referenced by the **TcpIpDhcpServerConfigRef** parameter of the corresponding **TcpIp/TcpIpConfig/TcpIpCtrl**.

## 2.3 Initialization

The TcpIp is initialized by calling the `TcpIp_InitMemory()` service and afterwards calling the `TcpIp_Init()` service with the configuration as parameter.

The module supports the configuration variant 'pre-compile' in combination with the configuration of multiple variants (post-build variant support). Therefore, the address of the global configuration data structure of the module has to be passed to the `TcpIp_Init()` function.

**Example**

```
TcpIp_Init(TcpIp_Config_Ptr);
```

## 2.4 Initialization in Multicore Environment

When running on multicore system, the `TcpIp_InitMemory()` and `TcpIp_Init()` must be invoked only once from one core. Any one of the cores configured to run TcpIp may selected to perform the initialization.

## 2.5 States

The TcpIp is operational after the initialization. Sockets can be requested immediately but binding a socket to a local address is only possible only after a local IP address could be assigned to the controller.

Based on the communication mode which was requested via `TcpIp_RequestComMode()`, the TcpIp module will show the following behavior:

**`TCPIP_STATE_ONLINE`**

- IP addresses can be assigned or may get assigned automatically based on configuration.
- Socket owner will get informed if an IP address enters state `TCPIP_IPADDR_STATE_ASSIGNED`.

**`TCPIP_STATE_ONHOLD`**

- No communication is possible and transmission requests will be rejected.
- But obtained sockets stay valid.
- For previously assigned IP addresses, the socket owner will get informed about the new IP address state `TCPIP_IPADDR_STATE_ONHOLD`.
- If state `TCPIP_STATE_ONLINE` is requested again, previously assigned IP addresses are restored and their state change is forwarded to the socket owner.
- If state `TCPIP_STATE_OFFLINE` is requested, all IP addresses of the controller will be released and notified as `TCPIP_IPADDR_STATE_UNASSIGNED`.

| `TCPIP_STATE_OFFLINE` |
|---|
| > The TcpIp module is not functional.<br>> Previously assigned IP addresses will be notified as `TCPIP_IPADDR_STATE_UNASSIGNED`.<br>> Obtained sockets will be deleted and no communication is possible. |

## 2.6 Main Functions

The TcpIp can be configured to use either a single or three MainFunctions:

| Single MainFunction (default) | Multiple MainFunctions |
|---|---|
| TcpIp_MainFunction | TcpIp_MainFunctionRx<br>TcpIp_MainFunctionState<br>TcpIp_MainFunctionTx |

Table 2-35 MainFunctions of the TcpIp module

By default, only a single MainFunction needs to be called. This MainFunction processes state changes, transmission and reception of data for all submodules (IPv4, IPv6, UDP, TCP and DHCPv4 Server). The TcpIp_MainFunction wraps the calls to the Rx, Tx and State MainFunctions.

In order to optimize the response behavior for certain use cases it may be reasonable to split-up the Rx, Tx and state processing parts of the MainFunction. These separate MainFunctions may then be arranged individually inside the task.

**There are some restrictions that apply to the mapping of the MainFunctions:**

> All three MainFunctions must be called cyclically with the same period. Otherwise the timing of the protocols will be inaccurate.

> If the MainFunctions are mapped to different tasks these tasks must not interrupt each other.

**Configuration in DaVinci Configurator Pro**
See configuration parameters **TcpIpMainFunctionPeriod** and **TcpIpMainFunctionSplitEnabled**.

Depending on the value of TcpIpMainFunctionSplitEnabled one or three **Schedulable Entities** need to be mapped in the **Task Mapping** view of the Rte.

## 2.7 Main Functions in Multicore Environment

If running on multicore systems, TcpIp uses separate MainFunctions for each core. Depending on the configuration, the functions listed in Table 2-30 will be generated for each core. The integrator must map the respective MainFunction(s) to the task(s) of the respective cores.

**Example**
If TCPIP is configured to run on two cores, the following MainFunctions shall be generated for TCPIP.

| Single MainFunction (default) | Multiple MainFunctions |
|---|---|
| TcpIp_MainFunction_Core0<br>TcpIp_MainFunction_Core1 | TcpIp_MainFunctionRx_Core0<br>TcpIp_MainFunctionState_Core0<br>TcpIp_MainFunctionTx_Core0<br>TcpIp_MainFunctionRx_Core1<br>TcpIp_MainFunctionState_Core1<br>TcpIp_MainFunctionTx_Core1 |

Table 2-36 Example Main Function for Multicore

**Caution**
The different MainFunctions belonging to one core must not be mapped to tasks of a different core. This will lead to unexpected behavior.

## 2.8 Timing Considerations

The processing of all TcpIp internal state changes is decoupled by at least a MainFunction cycle which can lead to delays in forwarding the events. This concerns both the forwarding of information to the upper layer and the processing of received TCP events such as RST or FIN, because the events are received in the TcpIp Rx-context but are processed in the next TcpIp State-context.

This means, after closing a Socket or receiving an RST, a MainFunction cycle must be waited until it is available for a new communication again.

The processing/forwarding of received TCP segments/data as well as the transmission of TCP data is also decoupled via the MainFunction cycle. Consequently, the TCP stack acknowledges (ACK) all data received between two MainFunction cycles during the next MainFunction execution.

## 2.9 Error Handling

### 2.9.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `TCPIP_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator but must have the same signature as the service `Det_ReportError()`.

The reported TCPIP ID is 170.

The submodules use the following instance IDs:

| | |
|---|---|
| > TcpIp: | 1 |
| > IpV4: | 2 |
| > IpV6: | 3 |
| > DhcpV4Server: | 4 |

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services:

| Service ID | Service |
|---|---|
| 0x01 | TcpIp_Init |
| 0x02 | TcpIp_GetVersionInfo |
| 0x03 | TcpIp_GetSocketForUser (indirectly called by each configured socket owner) |
| 0x04 | TcpIp_Close |
| 0x05 | TcpIp_Bind |
| 0x06 | TcpIp_TcpConnect |
| 0x07 | TcpIp_TcpListen |
| 0x08 | TcpIp_TcpReceived |
| 0x0A | TcpIp_RequestIpAddrAssignment |
| 0x0B | TcpIp_ReleaseIpAddrAssignment |
| 0x0D | TcpIp_DhcpWriteOption |
| 0x0E | TcpIp_DhcpReadOption |
| 0x0F | TcpIp_ChangeParameter |
| 0x11 | TcpIp_GetPhysAddr |
| 0x12 | TcpIp_UdpTransmit |
| 0x13 | TcpIp_TcpTransmit |
| 0x14 | TcpIp_RxIndication |
| 0x15 | TcpIp_MainFunction |
| 0x16 | TcpIp_GetRemotePhysAddr |
| 0x17 | TcpIp_GetCtrlIdx |
| 0x19 | TcpIp_DhcpV6ReadOption |
| 0x1A | TcpIp_DhcpV6WriteOption |
| 0x1C | TcpIp_GetNdpCacheEntries |
| 0x1D | TcpIp_GetArpCacheEntries |
| 0x45 | TcpIp_GetAndResetMeasurementData |
| 0x80 | TcpIp_IpSecSaEntryPairPrepareDelete |

| Service ID | Service |
|---|---|
| 0x81 | TcpIp_ProvideTxBufferInt |
| 0x82 | TcpIp_ProvideTxBuffer |
| 0x83 | TcpIp_TransmitTo |
| 0x84 | TcpIp_TransmitToInt |
| 0x85 | TcpIp_GetLocNetAddr |
| 0x86 | TcpIp_GetLocNetMask |
| 0x87 | TcpIp_SetDhcpHostNameOption |
| 0x8B | TcpIp_ClearARCache |
| 0x8C | TcpIp_SendGratuitousArpReq |
| 0x8D | TcpIp_IpSecSaEntryPairAdd |
| 0x8E | TcpIp_IpSecSaEntryPairDelete |
| 0x8F | TcpIp_IpSec_VInvokeSpdCallout |
| | |
| *Services of the IpV4 submodule* | |
| 0x00 | IpV4_Init |
| 0x01 | IpV4_Ip_MainFunction |
| 0x03 | IpV4_GetVersionInfo |
| 0x04 | IpV4_Ip_GetLocNetAddrCfgSrc |
| 0x05 | IpV4_Ip_SetNetAddr |
| 0x07 | IpV4_Ip_IsAddrIdAcceptable |
| | |
| 0x20 | IpV4_Ip_ProvideTxBuffer |
| 0x21 | IpV4_Ip_Transmit |
| 0x22 | IpV4_Ip_GetLocNetAddr |
| 0x23 | IpV4_Ip_GetLocNetMask |
| 0x24 | IpV4_Ip_AddrLossInd |
| 0x25 | IpV4_Ip_Dhcp_AddrInd |
| 0x26 | IpV4_Ip_ProvideNextTxBuffer |
| | |
| 0x28 | IpV4_Ip_RxIndication |
| 0x29 | IpV4_Ip_TxConfirmation |
| | |
| 0x30 | IpV4_Ip_Init |
| 0x31 | IpV4_Ip_MainFunction |
| 0x32 | IpV4_Ip_Reset |
| 0x33 | IpV4_Ip_AddrConflictInd |
| 0x34 | IpV4_Ip_ResetSocket |
| 0x35 | IpV4_Ip_SetTimeToLive |

| Service ID | Service |
|---|---|
| 0x36 | IpV4_Ip_SetTypeOfService |
| 0x37 | IpV4_Ip_SetEthIfFramePrio |
| 0x38 | IpV4_Ip_RequestIpAddrAssignment |
| 0x39 | IpV4_GetLastDuplicateDhcpAddrDid |
| | |
| 0x40 | IpV4_Arp_RxIndication |
| | |
| 0x50 | IpV4_Arp_Init |
| 0x51 | IpV4_Arp_Reset |
| 0x52 | IpV4_Arp_MainFunction |
| 0x53 | IpV4_Arp_GetPhysicalAddress |
| 0x54 | IpV4_Arp_SendArpRequest |
| 0x55 | IpV4_Arp_MapIpToPhysMulticastAddr |
| 0x56 | IpV4_Arp_SendArpProbe |
| 0x57 | IpV4_Arp_SendArpAnnouncement |
| 0x58 | IpV4_Arp_SendGratuitousArpReq |
| | |
| 0x60 | IpV4_Icmp_SendEcho |
| | |
| 0x70 | IpV4_Icmp_Init |
| 0x71 | IpV4_Icmp_MainFunction |
| 0x72 | IpV4_Icmp_RxIndication |
| | |
| *Services of the IpV6 submodule* | |
| 0x00 | IpV6_GetVersionInfo |
| 0x01 | IpV6_InitMemory |
| 0x02 | IpV6_Init |
| 0x03 | IpV6_MainFunction |
| 0x04 | IpV6_ProvideTxBuffer |
| 0x05 | IpV6_Transmit |
| 0x06 | IpV6_RxIndication |
| 0x07 | IpV6_TxConfirmation |
| 0x09 | IpV6_SetAddress |
| 0x0C | IpV6_SetTrafficClass |
| 0x0D | IpV6_SetFlowLabel |
| 0x0E | IpV6_IsValidDestinationAddress |
| 0x0F | IpV6_GetLocalAddressCfgSrc |
| 0x10 | IpV6_GetCurrentHopLimit |

| Service ID | Service |
|---|---|
| 0x11 | IpV6_GetPhysAddr |
| 0x12 | IpV6_GetAsBcAddrId |
| 0x13 | IpV6_IsAddrIdAcceptable |
| 0x14 | IpV6_SetHopLimit |
| 0x15 | IpV6_SetEthIfFramePrio |
| 0x16 | IpV6_ResetSocket |
| 0x17 | IpV6_CancelTransmit |
| 0x18 | IpV6_GetLocalAddress |
| 0x19 | IpV6_GetDefaultRouterAddress |
| 0x20 | IpV6_Icmp_ProvideTxBuffer |
| 0x21 | IpV6_Icmp_TxEchoRequest |
| 0x23 | IpV6_Icmp_Transmit |
| 0x24 | IpV6_Icmp_MainFunction |
| 0x30 | IpV6_Ndp_MainFunction |
| 0x40 | IpV6_Mld_MainFunction |
| 0x41 | IpV6_Mld_Init |

Table 2-37 Service IDs

The errors reported to DET are described in the following table:

| Error Code | Description |
|---|---|
| 0x01 | TCPIP_E_NOTINIT |
| 0x02 | TCPIP_E_PARAM_POINTER |
| 0x03 | TCPIP_E_INV_ARG |
| 0x04 | TCPIP_E_NOBUFS |
| 0x07 | TCPIP_E_MSGSIZE |
| 0x08 | TCPIP_E_PROTOTYPE |
| 0x09 | TCPIP_E_ADDRINUSE |
| 0x0A | TCPIP_E_ADDRNOTAVAIL |
| 0x0B | TCPIP_E_ISCONN |
| 0x0C | TCPIP_E_NOTCONN |
| 0x0D | TCPIP_E_NOPROTOOPT |
| 0x0E | TCPIP_E_AFNOSUPPORT |
| 0x0F | TCPIP_E_INIT_FAILED |
| | |
| *Services of the IpV4 submodule* | |
| 0x01 | IPV4_E_NOT_INITIALIZED |
| 0x02 | IPV4_E_INV_POINTER |
| 0x03 | IPV4_E_INV_PARAM |

| Error Code | Description |
|---|---|
| 0x04 | IPV4_E_INV_CTRL_IDX |
| 0x05 | IPV4_E_INV_SOCK_IDX |
| 0x06 | IPV4_E_INV_CONFIG |
| 0x07 | IPV4_E_INV_ADDR_ID |
| | |
| *Services of the IpV6 submodule* | |
| 0x01 | IPV6_E_NOT_INITIALIZED |
| 0x02 | IPV6_E_INV_POINTER |
| 0x03 | IPV6_E_INV_SOCK_IDX |
| 0x04 | IPV6_E_INV_BUFF_IDX |
| 0x05 | IPV6_E_INV_CTRL_IDX |
| 0x06 | IPV6_E_INV_DATA_IDX |
| 0x07 | IPV6_E_INV_PARAM |
| 0x08 | IPV6_E_DLIST_BUFFER_EMPTY |

Table 2-38 Errors reported to DET

### 2.9.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. at least one reference to a DEM event is configured in the TcpIp by referencing a DEM event using this EcuC path: `/ActiveEcuC/Dem/DemConfigSet/DemEventParameter`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

| Error Code | Description |
|---|---|
| TCPIP_E_DUPLICATE_DHCP_ADDR | An ECU's IPv4 address was used by another network node, i.e., the ECU received a frame which uses the same source IPv4 address but a different source MAC address.<br>The DEM event is configured by creating a reference to a DEM event named `E DUPLICATE DHCP ADDR` in the EcuC-container:<br>`/ActiveEcuC/TcpIp/TcpIpConfig/TcpIpCtrl_0_Fix/TcpIpCtrlDemEventParameterRefs`<br>Refer to IpV4_GetLastDuplicateDhcpAddrDid for details on how to retrieve the related DID information. |

Table 2-39 Errors reported to DEM

**Note**
From all references contained in the EcuC container `/ActiveEcuC/TcpIp/TcpIpConfig/TcpIpCtrl_0_Fix/TcpIpCtrlDemEventParameterRefs` the TcpIp currently only supports:
TCPIP_E_DUPLICATE_DHCP_ADDR

### 2.9.3 Other Error Reporting and Diagnostic Options

The TcpIp offers the possibility to track the following OEM-specific error conditions:

#### 2.9.3.1 Tracking discarded / overwritten ARP entries

TcpIp ARP (IPv4) offers the possibility to notify the UL in case an existing ARP entry was replaced by a new entry or a new entry could not be added to the ARP table because the maximum number of ARP entries has been reached.

Implementing this scenario is supported by the IPv4-subcomponent as follows:

> Turning on EcuC switch TcpIpArpDiscardedEntryHandling leads to the handling of the ARP-cache as ARP-table, which is never updated for new IpV4 address once it is full.

> Turning off the switch TcpIpArpDiscardedEntryHandling leads to the overwriting of the oldest existing entry if ARP cache is full.

> If the MAC address of an existing valid entry changes, the old entry will be updated with the new MAC.

> Configuring the callback <Up_PhysAddrTableChg offers the possibility to track all new entries that are inserted into the ARP table.

> Configuring the callback <Up_PhysAddrTableEntryDiscarded> offers the possibility to track all IP addresses which are discarded / overwitten because the ARP table is already full. It also tracks the updated ARP entry with a new MAC.
(At the time when the address should have been resolved.)

#### 2.9.3.2 Triggering the publishing of IP-address via diagnostic services

Some diagnostic services may demand that the ECU sends a gratuitous ARP request packet to publish its IP-address. This possibility is offered by the function TcpIp_SendGratuitousArpReq.

#### 2.9.3.3 Get DHCP Status for IPv4

TcpIp DHCPv4 offers the possibility to retrieve the status of the DHCP handshaking for an IPv4 address that has DHCP configured as address assignment method.
This possibility is offered by the function TcpIp_DhcpV4_GetStatus.

#### 2.9.3.4 IPv6 Duplicate Address Detection Conflict

TcpIp supports the implementation of a DEM event that is equivalent to the runtime error TCPIP_E_DADCONFLICT described in [1].

Setting the DEM event to 'active' can be done by configuring the callback DADAddressConflict() (Refer to section 4.5.1.8).

In order to 'heal' the DEM event, the LocalAddrId parameter of the latter callback has to be stored and the state-change of the affected address back to a valid IP address has to be tracked using the configurable callback LocalIpAddrAssignmentChg()-callback (Refer to EcuC item: `/MICROSAR/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerLocalIpAddrAssignmentChgName`).

# 3 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic TCPIP into an application environment of an ECU.

## 3.1 Scope of Delivery

The delivery of the TCPIP contains the files which are described in the chapters 3.1.1 and 3.1.2:

### 3.1.1 Static Files

| File Name | Description |
|---|---|
| ***TcpIp Static Files*** | |
| TcpIp.(c\|h) | Implementation/Declaration of public TcpIp APIs |
| TcpIp_Cbk.h | Declaration of public call-back APIs |
| TcpIp_Types.h | Types header for ASR API of TcpIp |
| TcpIp_Tcp.(c\|h) | Implementation of TCP (internal functions) |
| TcpIp_Tcp_Cbk.h | Implementation of TCP (call-back function declarations) |
| TcpIp_Udp.(c\|h) | Implementation of UDP (internal functions) |
| TcpIp_Udp_Cbk.h | Implementation of UDP (call-back function declarations) |
| TcpIp_Priv.(c\|h) | Implementation of internal/private TcpIp functions |
| TcpIp_Priv_Types.h | TcpIp internal Type definitions |
| TcpIp_IpSec.(c\|h) | Implementation of IPsec |
| TcpIp_MemMap.h | Compatibility file which includes MemMap.h.<br>(File is generated starting from MICROSAR Classic release R28 and not statically delivered anymore.) |
| | |
| ***IpV4 Static Files*** | |
| TcpIp_IpV4.(c\|h) | Implementation of IPv4 (internal functions) |
| TcpIp_IpV4_Cbk.h | Implementation of IPv4 (call-back function declarations) |
| TcpIp_IpV4_Types.h | Internal type definitions of the IpV4 submodule |
| TcpIp_Arp.(c\|h) | Implementation of ARP for IPv4 (internal functions) |
| TcpIp_Arp_Cbk.h | Implementation of ARP for IPv4 (call-back function declarations) |
| TcpIp_DhcpV4.(c\|h) | Implementation of DHCP Client for IPv4 (internal functions) |
| TcpIp_DhcpV4_Cbk.h | Implementation of DHCP Client for IPv4 (call-back function declarations) |
| TcpIp_IcmpV4.(c\|h) | Implementation of ICMP for IPv4 (internal functions) |
| TcpIp_IcmpV4_Cbk.h | Implementation of ICMP for IPv4 (call-back function declarations) |
| TcpIp_IpV4_Priv.(c\|h) | Implementation of internal/private functions of the IpV4 submodule |
| | |
| ***DhcpV4Server Static Files*** | |

| File Name | Description |
|---|---|
| TcpIp_DhcpV4Server.(c\|h) | Implementation of DHCP Server for IPv4 |
| | |
| ***IpV6 Static Files*** | |
| TcpIp_IpV6.(c\|h) | Implementation of IPv6 (internal functions) |
| TcpIp_IpV6_Cbk.h | Implementation of IPv6 (call-back function declarations) |
| TcpIp_IpV6_Types.h | Internal type definitions of the IpV6 submodule |
| TcpIp_IcmpV6.(c\|h) | Implementation of ICMP for IPv6 (internal functions) |
| TcpIp_Ndp.(c\|h) | Implementation of NDP for IPv6 (internal functions) |
| TcpIp_Mld.(c\|h) | Implementation of MLDv2 for IPv6 (internal functions) |
| TcpIp_DhcpV6.(c\|h) | Implementation of DHCP Client for IPv6 (internal functions) |
| TcpIp_DhcpV6_Cbk.h | Implementation of DHCP Client for IPv6 (call-back function declarations) |
| TcpIp_IpV6_Priv.(c\|h) | Implementation of internal/private functions of the IpV6 submodule |
| | |
| ***Tls Static Files*** | |
| TcpIp_Tls.(c\|h)) | Implementation of public TcpIp_Tls APIs. |
| TcpIp_Tls_Cbk.h | Implementation of TLS (call-back function declarations). |
| TcpIp_Tls_Types.h | Internal type definitions of the TLS submodule. |
| TcpIp_Tls12Client.(c\|h) | Implementation of TLS 1.2 Client (internal functions). |
| TcpIp_Tls12Core.(c\|h) | Implementation of TLS 1.2 Core (internal functions). |
| TcpIp_Tls13Client.(c\|h) | Implementation of TLS 1.3 Client (internal functions). |
| TcpIp_Tls13Core.(c\|h) | Implementation of TLS 1.3 Core (internal functions). |
| TcpIp_TlsClientCommon.(c\|h) | Implementation of common TLS Client (internal functions). |
| TcpIp_TlsCoreCommon.(c\|h) | Implementation of common TLS Core (internal functions). |
| TcpIp_TlsPriv.(c\|h) | Implementation of private TLS functions. |
| TcpIp_TlsServer.(c\|h) | Implementation of TLS Server (internal functions). |

Table 3-1 Static files

### 3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator Pro.

| File Name | Description |
|---|---|
| *Dynamic Files (always generated if TcpIp module exists in configuration)* | |
| TcpIp_Cfg.h | TcpIp Pre-compile time parameter configuration |
| TcpIp_Lcfg.(c\|h) | TcpIp Link-time parameter configuration |
| TcpIp_PBcfg.(c\|h) | TcpIp Post-Build parameter configuration (empty file)<br>Post-build is currently not supported by the TcpIp. |

| File Name | Description |
|---|---|
| TcpIp_MemMap.h | TcpIp memory sections.<br>(File is generated starting from MICROSAR Classic release R28.) |

Table 3-2 Generated files

### 3.1.3 Mapping of memory sections

TcpIp (15.08.00 and newer) allows the mapping of bigger data arrays to dedicated memory sections.

| Array name | Memory section |
|---|---|
| TcpIp_ReassemblyBuffer | `TCPIP_(START|STOP)_SEC_REASSEMBLYBUFFER_VAR_NO_INIT_8` |
| TcpIp_IpV6ReassemblyDataBuffer | `TCPIP_(START|STOP)_SEC_IPV6REASSEMBLYDATABUFFER_VAR_NO_INIT_8` |
| TcpIp_IpV6FragmentTxDataBuffer | `TCPIP_(START|STOP)_SEC_IPV6FRAGMENTTXDATABUFFER_VAR_NO_INIT_8` |
| TcpIp_TcpTxBuffer | `TCPIP_(START|STOP)_SEC_TCPTXBUFFER_VAR_NO_INIT_8` |
| TcpIp_TcpRxBuffer | `TCPIP_(START|STOP)_SEC_TCPRXBUFFER_VAR_NO_INIT_8` |
| TcpIp_TcpRetryQElement | `TCPIP_(START|STOP)_SEC_TCPRETRYQELEMENT_VAR_NO_INIT_UNSPECIFIED` |
| TcpIp_TcpOooQElement | `TCPIP_(START|STOP)_SEC_TCPOOOQELEMENT_VAR_NO_INIT_UNSPECIFIED` |
| TcpIp_SocketTcpDyn | `TCPIP_(START|STOP)_SEC_SOCKETTCPDYN_VAR_NO_INIT_UNSPECIFIED` |
| TcpIp_TlsBufferRx | `TCPIP_(START|STOP)_SEC_TLSBUFFERRX_VAR_NO_INIT_8` |
| TcpIp_TlsBufferTx | `TCPIP_(START|STOP)_SEC_TLSBUFFERTX_VAR_NO_INIT_8` |
| TcpIp_SaEntry | `TCPIP_(START|STOP)_SEC_SAENTRY_VAR_NO_INIT_UNSPECIFIED` |
| TcpIp_RxProcessBuffer | `TCPIP_(START|STOP)_SEC_RXPROCESSBUFFER_VAR_NO_INIT_8` |

Table 3-3 Available memory sections

As default the new memory sections with the additional prefix could be mapped to the generic TcpIp memory sections.

For example:

`TCPIP_(START|STOP)_SEC`**`_IPV6REASSEMBLYDATABUFFER_`**`VAR_NO_INIT_8`

Could be mapped to

`TCPIP_(START|STOP)_SEC_VAR_NO_INIT_8`.

## 3.2 Cyclic & Background Task

The cyclic main functions of the TCPIP are handled automatically in the context of the `TcpIp_MainFunction()`, therefore no additional configuration must be performed.

The `TcpIp_Tls_MainFunctionLowPrio()` is an optional main function, which has to be mapped if the TCPIP module is configured for TLS. This main function performs the time-consuming operations, such as the calculation of cryptographical material or the handling of the certificates. Therefore, this task should be mapped to a periodical background task.

## 3.3 Critical Sections

The TCPIP calls service functions of upper layers to prevent interruption of critical sections.

These service functions must be provided (normally by the Schedule Manager) and configured accordingly.

To ensure data consistency and a correct function of the TCPIP module the exclusive areas `TCPIP_EXCLUSIVE_AREA_0` and `TCPIP_EXCLUSIVE_AREA_1` must be provided during the integration.

Considering the timing behavior of the system (e.g., depending on the CPU load of the system, priorities and interruptibility of interrupts and OS tasks and their jitter and delay times) the integrator must choose and configure a critical section solution in such way that it is ensured that the API functions do not interrupt each other.

It is recommended to use the functions `SuspendAllInterrupts()` and `ResumeAllInterrupts`() (core interrupt locks) for `TCPIP_EXCLUSIVE_AREA_0.` For single core environments the critical section `TCPIP_EXCLUSIVE_AREA_1` must be configured equally.

For both critical sections, nested calls of the same or the other critical section must be allowed.

### 3.3.1 Critical Sections in Multicore Environment

When running on multicore systems, the integrator must choose a critical section solution for `TCPIP_EXCLUSIVE_AREA_1` in such a way that API functions running on different cores will not interrupt each other. OS spinlocks shall be used to prevent different cores from interrupting each other inside critical sections.

**Note**

When running on a single core system, `TCPIP_EXCLUSIVE_AREA_1` must be configured the same way as `TCPIP_EXCLUSIVE_AREA_0`.

When running on a multicore system, `TCPIP_EXCLUSIVE_AREA_1` must be protected by OS spinlocks.

### 3.3.2 External calls within Critical Sections

The TcpIp module might call the following external APIs within the scope of a critical section:

- Csm *(Applicable if IPsec and TCP functionality is configured.)*
  - Csm_KeyElementSet
  - Csm_KeySetValid
  - Csm_MacGenerate
- EthIf *(Applicable if TCP functionality is configured.)*
  - EthIf_ProvideTxBuffer
  - EthIf_Transmit
- Det *(Applicable if development error reporting is configured.)*

  - Det_ReportError
- VStdLib
  - VStdLib_MemCpy
  - VStdLib_MemClr
- Configurable callouts (based on configuration)
  - TcpIpSocketOwnerCopyTxDataName
  - TcpIpIpSecAuditEventCalloutFunction
  - TcpIpSocketOwnerTcpIpEventName
  - TcpIpPhysAddrChgHandlerName
  - TcpIpSocketOwnerTcpConnectedName

It must be ensured, that these functions can be invoked while interrupts are locked. (Based on the configured scope of the critical sections.)

## 3.4 Preemption

TcpIp expects that its own main functions do not interrupt each other. Additionally, it expects that an own main function does not interrupt main functions of related modules and is not interrupted by main functions of related modules.

**Caution**
Consider main function expectations when using preemptive tasks.

## 3.5 Dependency to other components

### 3.5.1 CSM - Crypto Service Manager

The crypto service manager allows to perform crypto job and key operations used by the TLS and IPsec sub module. TLS requires the CSM module for cryptographic operations. All TLS and IPsec related CSM jobs and operations must be synchronous operations. In case a hardware crypto driver is used, the security related elements can be hidden from the host controller’s memory.

**Caution**
TLS requires a CSM/crypto stack implementation that implements the AUTOSAR 4.4 redirection feature and key copy feature.

TcpIp also supports usage of user defined CSM APIs (which are AUTOSAR 4.4 compliant). The configuration of this feature is explained in Section 5.7.

### 3.5.2 KeyM – Key Manager

The Key Manager module provides operations for certificate handling for the TLS and IPsec sub module.

# 4 API Description

## 4.1 Type Definitions

The types defined by the TCPIP are described in this chapter.

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| TcpIp_ConfigType | | Configuration data structure of the TcpIp module. | Configuration data structure of the TcpIp module. |
| TcpIp_DomainType | uint16 | TcpIp address families. | `TCPIP_AF_INET (0x02)` Use IPv4 |
| | | | `TCPIP_AF_INET6 (0x1c)` Use IPv6 |
| TcpIp_ProtocolType | Enum | Protocol type used by a socket. | `TCPIP_IPPROTO_TCP (0x06)` Use TCP |
| | | | `TCPIP_IPPROTO_UDP (0x11)` Use UDP |
| TcpIp_LocalAddrIdType | uint8 | Address identification type for unique identification of a local IP address and EthIf Controller configured in the TcpIp module. | |
| TcpIp_SocketIdType | uint8 or uint16 | socket identifier type for unique identification of a TcpIp stack socket. TCPIP_SOCKETID_INVALID shall specify an invalid socket handle | Uint8 for less than 250 configured sockets. Uint16 for more than 250 configured sockets. TCPIP_SOCKETID_INVALID is 0xFF or 0xFFFF depending on the C-Type. |
| TcpIp_StateType | Enum | Specifies the TcpIp state for a specific EthIf controller. | `TCPIP_STATE_ONLINE` TCP/IP stack state for a specific EthIf controller is ONLINE, i.e. communication via at least one IP address is possible. |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | `TCPIP_STATE_ONHOLD` TCP/IP stack state for a specific EthIf controller is ONHOLD, i.e. no communication is currently possible (e.g. link down). |
| | | | `TCPIP_STATE_OFFLINE` TCP/IP stack state for a specific EthIf controller is OFFLINE, i.e. no communication is possible. |
| | | | `TCPIP_STATE_STARTUP` TCP/IP stack state for a specific EthIf controller is STARTUP, i.e. IP address assignment in progress or ready for manual start, communication is currently not possible. |
| | | | `TCPIP_STATE_SHUTDOWN` TCP/IP stack state for a specific EthIf controller is SHUTDOWN, i.e. release of resources using the EthIf controller, release of IP address assignment. |
| TcpIp_IpAddrStateType | Enum | Specifies the state of local IP address assignment | `TCPIP_IPADDR_STATE_ASSIGNED` local IP address is assigned |
| | | | `TCPIP_IPADDR_STATE_ONHOLD` local IP address is assigned, but cannot be used as the network is not active |
| | | | `TCPIP_IPADDR_STATE_UNASSIGNED` local IP address is unassigned |
| TcpIp_EventType | Enum | Events reported by TcpIp. | `TCPIP_TCP_RESET` TCP connection was reset, TCP socket and all related resources have been released. |
| | | | `TCPIP_TCP_CLOSED` TCP connection was closed successfully, TCP socket and all related resources have been released. |
| | | | `TCPIP_TCP_FIN_RECEIVED` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | A FIN signal was received on the TCP connection, TCP socket is still valid. |
| | | | `TCPIP_UDP_CLOSED` UDP socket and all related resources have been released. |
| | | | `TCPIP_TLS_HANDSHAKE_SUCCEEDED` TLS handshake successfully established, TLS connection available. |
| TcpIp_IpAddrAssignmentType | Enum | Specification of IPv4/IPv6 address assignment policy. | `TCPIP_IPADDR_ASSIGNMENT_STATIC` Static configured IPv4/IPv6 address. |
| | | | `TCPIP_IPADDR_ASSIGNMENT_LINKLOCAL_DOIP` Linklocal IPv4/IPv6 address assignment using DoIP parameters. |
| | | | `TCPIP_IPADDR_ASSIGNMENT_DHCP` Dynamic configured IPv4/IPv6 address by DHCP. |
| | | | `TCPIP_IPADDR_ASSIGNMENT_LINKLOCAL` Linklocal IPv4/IPv6 address assignment. |
| | | | `TCPIP_IPADDR_ASSIGNMENT_IPV6_ROUTER` Dynamic configured IPv4/IPv6 address by Router Advertisement. |
| TcpIp_ReturnType | Enum | TcpIp specific return type. | `TCPIP_OK` Operation completed successfully. |
| | | | `TCPIP_E_NOT_OK` Operation failed. |
| | | | `TCPIP_E_PHYS_ADDR_MISS` Operation failed because of an ARP/NDP cache miss. |
| TcpIp_ParamIdType | uint8 | Type for the specification of all supported Parameter IDs. | `TCPIP_PARAMID_TCP_RXWND_MAX  0x00` Specifies the maximum TCP receive window for the socket. |
| | | | `TCPIP_PARAMID_FRAMEPRIO  (0x01)` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | Specifies the frame priority for outgoing frames on the socket. |
| | | | `TCPIP_PARAMID_TCP_NAGLE  (0x02)`<br>Specifies if the Nagle Algorithm according to IETF RFC 896 is enabled or not. |
| | | | `TCPIP_PARAMID_TCP_KEEPALIVE  (0x03)`<br>Specifies if TCP Keep Alive Probes are sent on the socket connection. |
| | | | `TCPIP_PARAMID_TTL  (0x04)`<br>Specifies the time to live value for outgoing frames on the socket. For IPv6 this parameter specifies the value of the HopLimit field used in the IPv6 header. |
| | | | `TCPIP_PARAMID_TCP_KEEPALIVE_TIME  (0x05)`<br>Specifies the time in [s] between the last data packet sent (simple ACKs are not considered data) and the first keepalive probe. |
| | | | `TCPIP_PARAMID_TCP_KEEPALIVE_PROBES_MAX (0x06)`<br>Specifies the maximum number of times that a keepalive probe is retransmitted. |
| | | | `TCPIP_PARAMID_TCP_KEEPALIVE_INTERVAL  (0x07)`<br>Specifies the interval in [s] between subsequent keepalive probes. |
| | | | `TCPIP_PARAMID_UDP_CHECKSUM  (0x0C)`<br>Specifies if UDP checksum 0 is allowed. |
| | | | `TCPIP_PARAMID_TLS_CONNECTION_ASSIGNMENT (0x0D)`<br>Specifies the TLS connection reference assigned to the TCP socket. |
| | | | `TCPIP_PARAMID_TLS_CONNECTIONBASED_SESSION_RESUMPTION (0x0E)`<br>Requests the usage of session resumption (connection based). |
| | | | `TCPIP_PARAMID_VENDOR_SPECIFIC  (0x80)` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | Start of vendor specific range of parameter IDs. |
| TcpIpIpAddrWildcardType | uint32 | IP address wildcard. | `TCPIP_IPADDR_ANY`<br>defines the value used as wildcard |
| TcpIpIp6AddrWildcardType | uint32 | IP6 address wildcard. | `TCPIP_IP6ADDR_ANY`<br>defines the value used as wildcard for all IP6 address parts |
| TcpIpPortWildcardType | uint16 | Port wildcard. | `TCPIP_PORT_ANY`<br>defines the value used as wildcard |
| TcpIpLocalAddrIdWildcardType | uint8 | LocalAddrId wildcard. | `TCPIP_LOCALADDRID_ANY`<br>defines the value used as wildcard |
| TcpIp_ArpCacheEntryType<br>TcpIp_NdpCacheEntryType | struct | Represents an entry inside the ARP/NDP cache | `TCPIP_(NDP\|ARP)_ENTRY_STATIC (0x00)`<br>defines the value used for static NDP/ARP cache entries used in TcpIp_Get(Ndp\|Arp)CacheEntries() API |
| | | | `TCPIP_(NDP\|ARP)_ENTRY_VALID (0x01)`<br>defines the value used for valid NDP/ARP cache entries used in TcpIp_Get(Ndp\|Arp)CacheEntries() API |
| | | | `TCPIP_(NDP\|ARP)_ENTRY_STALE (0x02)`<br>defines the value used for stale NDP/ARP cache entries used in TcpIp_Get(Ndp\|Arp)CacheEntries() API |
| TcpIp_DhcpEventType | uint8 | Type of the current DHCPv4/DHCPv6 event. | `TCPIP_DHCP_EVENT_TX_DISCOVER_SOLICIT (0x00)`<br>Discover (DHCPv4) or Solicit (DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_RX_OFFER_ADVERTISE (0x01)`<br>Offer (DHCPv4) or Advertise (DHCPv6) message was received. |
| | | | `TCPIP_DHCP_EVENT_TX_REQUEST (0x02)`<br>Request message (DHCPv4, DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_RX_ACK_REPLY (0x03)` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | Acknowledge (DHCPv4) or Reply (DHCPv6) message was received. |
| | | | `TCPIP_DHCP_EVENT_TX_V6_CONFIRM (0x04)` Confirm (DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_TX_V6_RENEW (0x05)` Renew (DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_TX_V6_REBIND (0x06)` Rebind (DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_TX_V6_RELEASE (0x07)` Release (DHCPv6) message will be transmitted. |
| | | | `TCPIP_DHCP_EVENT_TX_V6_DECLINE (0x08)` Acknowledge Decline (DHCPv6) message will be transmitted. |
| TcpIp_MeasurementIdxType | uint8 | Represents the index to select specific measurement data | `TCPIP_MEAS_DROP_TCP (0x01)` Measurement index of dropped TCP packets caused by invalid destination IP address or TCP port. |
| | | | `TCPIP_MEAS_DROP_UDP (0x02)` Measurement index of dropped UDP packets caused by invalid destination IP address or UDP port. |
| | | | `TCPIP_MEAS_DROP_IPV4 (0x03)` Measurement index of dropped datagrams caused by invalid destination IPV4 address. |
| | | | `TCPIP_MEAS_DROP_IPV6 (0x04)` Measurement index of dropped datagrams caused by invalid destination IPV6 address. |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_DROP_LAYER3_IPV6 (0x80)` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | Measurement index of dropped datagrams caused by invalid IPv6 packet. |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_DROP_LAYER4 (0x81)` Measurement index of dropped datagram caused by invalid TCP/UDP packets. |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_NR_SA_PAIRS (0x82)` Measurement index of number of valid SA entry pairs present in the SAD. (Only applicable if IPsec is enabled) |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_SA_PAIRS (0x83)` Measurement index of statuses of valid SA entry pairs in the SAD. (Only applicable if IPsec is enabled) |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_DROP_INSUFF_TCP_TX_BUFFER (0x84)` Measurement index of dropped datagram caused by insufficient TCP Tx buffer. |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_DROP_INSUFF_TCP_RX_BUFFER (0x85)` Measurement index of dropped datagram caused by insufficient TCP Rx buffer. |
| | | | `TCPIP_MEAS_VENDOR_SPECIFIC_DROP_INSUFF_IPV4_FRAGMENT_RX_BUFFER (0x86)` Measurement index of dropped datagram caused by insufficient IpV4 reassembly buffer. |
| | | | `TCPIP_MEAS_ALL (0xFF)` Represents all measurement indexes, only in case of reset. |
| TcpIp_IpSecPolicyType | uint8 | IPsec Security Policy Type | `TCPIP_IPSEC_POLICY_BYPASS (0x00)` Defines the bypass security policy which means the IP packet will not be protected by IPsec. |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | `TCPIP_IPSEC_POLICY_PROTECT (0x01)`<br>Defines the protect security policy which means the IP packet will be protected by IPsec. |
| | | | `TCPIP_IPSEC_POLICY_OPTIONAL (0x02)`<br>Defines the optional security policy which means the IP packet will be protected by IPsec if possible (if SA entry is present) or be unprotected otherwise. |
| | | | `TCPIP_IPSEC_POLICY_DISCARD (0x03)`<br>Defines the discard security policy which means the IP packet will be discarded without further processing. |
| TcpIp_IpSecHeaderType | uint8 | IPsec Headers | `TCPIP_IPSEC_HDR_AH_ESP (0x01)`<br>Both Authentication header and ESP header shall be used for IPsec protected traffic. (NOT supported) |
| | | | `TCPIP_IPSEC_HDR_AH (0x02)`<br>Only Authentication header shall be used for IPsec protected traffic. |
| | | | `TCPIP_IPSEC_HDR_ESP (0x03)`<br>Only ESP header shall be used for IPsec protected traffic. |
| TcpIp_IpSecEventType | uint8 | IPsec auditable events | `TCPIP_IPSEC_NO_EVENT (0x00)`<br>No IPsec Auditable event. |
| | | | `TCPIP_IPSEC_EVENT_INVALID_HDR (0x01)`<br>Invalid IPsec header. |
| | | | `TCPIP_IPSEC_EVENT_DISCARD_POLICY (0x02)`<br>Discard IPsec security policy. |
| | | | `TCPIP_IPSEC_EVENT_SEQ_OVERFLOW (0x03)`<br>AH sequence number overflow. |
| | | | `TCPIP_IPSEC_EVENT_ANTIREPLAY (0x04)` |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | AH anti replay attack. |
| | | | TCPIP_IPSEC_EVENT_VERIFY_FAILED (0x05) AH authentication data verification failed. |
| | | | TCPIP_IPSEC_EVENT_GEN_FAILED (0x06) AH authentication data generation failed. |
| | | | TCPIP_IPSEC_EVENT_FRAGMENT (0x07) Fragmented IPsec IP packet. |
| | | | TCPIP_IPSEC_EVENT_SAENTRY_NOTFOUND (0x08) SA entry lookup failure. |
| TcpIp_IpSecIntegrityTransformType | uint16 | Integrity algorithm transform Ids | Refer Section Transform Type 3 - Integrity Algorithm Transform IDs in [50] |
| TcpIp_IpSecEncrTransformType | uint16 | Encryption algorithm transform Ids | Refer Section Transform Type 1 - Encryption Algorithm Transform IDs in [50] |
| TcpIp_IpSecStateType | uint8 | State of IPsec SA entry | TCPIP_IPSEC_STATUS_UNSET (0x00u) Init Value |
| | | | TCPIP_IPSEC_STATUS_TX_RX (0x01u) SA entry is usable for Tx and Rx |
| | | | TCPIP_IPSEC_STATUS_RX_ONLY (0x02u) SA entry is usable for only Rx |
| TcpIp_CertificateIdType | uint16 | Certificate identifier used by TcpIp and KeyM | - |
| TcpIp_CertificateStatusType | uint8 | Information about the status of a certificate. | KEYM_CERTIFICATE_VALID (0) Certificate valid |
| | | | KEYM_CERTIFICATE_INVALID (1) Certificate invalid |
| | | | KEYM_CERTIFICATE_NOT_PARSED (2) |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | | Certificate not parsed |
| | | | `KEYM_CERTIFICATE_PARSED_NOT_VALIDATED (3)` Certificate parsed but not valid |
| | | | `KEYM_CERTIFICATE_NOT_AVAILABLE (4)` Certificate not loaded / available |
| TcpIp_CertValidationResultType | uint8 | Overall certificate validation result.<br><br>Used to provide detailed certificate information to the <UpperLayer> in TcpIp_SocketOwnerTlsValidationResult callout. | `TCPIP_TLS_VALIDATION_OK (0x00u)` The certificate chain is validated and ok, or the socket owner accepts the connection explicitly |
| | | | `TCPIP_TLS_VALIDATION_NOT_OK (0x01u)` The certificate chain is NOT valid, and TLS will not accept the connection |
| | | | `TCPIP_TLS_VALIDATION_UNKNOWN (0xFFu)` The certificate chain has not been processed |
| TcpIp_OcspModeType | uint8 | Mode of the received OCSP response. Part of struct TcpIp_CertificateStatusResponseType.<br><br>Used to provide detailed certificate information to the <UpperLayer> in TcpIp_SocketOwnerTlsValidationResult callout. | `TCPIP_TLS_OCSP_MODE_SINGLE (0x00u)` RFC6066 TLS Certificate Status Request |
| | | | `TCPIP_TLS_OCSP_MODE_MULTI (0x01u)` RFC6961 TLS Multiple Certificate Status Request Extension |
| | | | `TCPIP_TLS_OCSP_MODE_NONE (0xFFu)` No Certificate Status Request received (Init value) |
| TcpIp_OcspConnectionStateType | uint8 | Connection state of the OCSP response handling. Gives information about the received OCSP extension and response from the TLS server.<br>Part of struct TcpIp_CertificateStatusResponseType. | `TCPIP_TLS_OCSP_CONNECTION_STATE_INACTIVE (0u)` (default value) No OCSP Request active |
| | | | `TCPIP_TLS_OCSP_CONNECTION_STATE_REQUESTED (1u)` Certificate status requested by the TLS client |
| | | | `TCPIP_TLS_OCSP_CONNECTION_STATE_STATUS_RECEIVED (2u)` Certificate status received from the TLS server |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | | Used to provide detailed certificate information to the <UpperLayer> in TcpIp_SocketOwnerTlsValidationResult callout | |
| TcpIp_OcspResponseStatusType | uint8 | Status of the received OCSP response. Gives information about the OCSP response status, which is included in the OCSP response.<br>Part of struct TcpIp_CertificateStatusResponseType.<br><br>Used to provide detailed certificate information to the <UpperLayer> in TcpIp_SocketOwnerTlsValidationResult callout | Values from RFC6960 - 4.2.1. ASN.1 Specification of the OCSP Response:<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_SUCCESSFUL (0x00u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_MALFORMEDREQUEST (x01u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_INTERNALERROR (0x02u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_TRYLATER (0x03u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_SIGREQUIRED (0x05u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_UNAUTHORIZED (0x06u)`<br>`TCPIP_TLS_OCSP_RESPONSE_STATUS_INVALID (0xFFu)` |
| TcpIp_TlsClientCertInfoStatusType | enum | Used to determine the processing state of the client certificate callout | `TCPIP_TLS_CLIENT_CERT_INFO_STATUS_READY`<br>A client certificate chain is available and can be read from KeyM<br>`TCPIP_TLS_CLIENT_CERT_INFO_STATUS_PENDING`<br>The certificate request is accepted, the processing of the socket owner is still ongoing<br>`TCPIP_TLS_CLIENT_CERT_INFO_STATUS_NOT_AVAILABLE`<br>No matching client certificate is available<br>`TCPIP_TLS_CLIENT_CERT_INFO_STATUS_ERROR`<br>The socket owner can not process the cert request (e.g. erroneous parameters, internal error, ...) |

Table 4-1 Type definitions

#### TcpIp_SockAddrType

Generic structure used by APIs to specify an IP address. (A specific address type can be derived from this structure via a cast to the specific struct type.)

| Struct Element Name | C-Type | Description |
|---|---|---|
| domain | TcpIp_DomainType (uint16) | This is the code for the address format of this address |

Table 4-2 TcpIp_SockAddrType

### TcpIp_SockAddrInetType

This structure defines an IPv4 address type which can be derived from the generic address structure via cast.

| Struct Element Name | C-Type | Description |
|---|---|---|
| domain | TcpIp_DomainType (uint16) | This is the code for the address format of this address |
| port | uint16 | port number |
| addr | uint32[1] | IPv4 address in network byte order |

Table 4-3 TcpIp_SockAddrInetType

### TcpIp_CertificateStatusResponseType

This structure stores information about the received certificate status response (OCSP). This information is part of the TLS validation result and forwarded to the UpperLayer via callout <Up_TlsValidationResultCallout>.

| Struct Element Name | C-Type | Description |
|---|---|---|
| OcspCertStatus | TcpIp_CertificateStatusType | OCSP certificate status.<br>Only valid if OcspConnectionState != TCPIP_TLS_OCSP_CONNECTION_STATE_INACTIVE |
| OcspResponseStatus | TcpIp_OcspResponseStatusType | Status of the received OCSP response |

| Struct Element Name | C-Type | Description |
|---|---|---|
| OcspMode | TcpIp_OcspModeType | Received OCSP mode |
| OcspConnectionState | TcpIp_OcspConnectionStateType | OCSP state during the TLS handshake |

Table 4-4 TcpIp_CertificateStatusResponseType

### TcpIp_CertValidationStatusType

This structure stores information about the validation status of the certificate. The used certificate (for which the information is) is identified by the parameter CertId. All information refers to this certificate.

| Struct Element Name | C-Type | Description |
|---|---|---|
| CertId | TcpIp_CertificateIdType | Identifier of the used Certificate |
| CertStatus | TcpIp_CertificateStatusType | Status of the used Certificate |
| CertStatusResponse | TcpIp_CertificateStatusResponseType | Received certificate status information<br>(Only updated if OCSP status response received) |
| IsSelfSigned | boolean | Self signed flag of the used certificate. |

Table 4-5 TcpIp_CertValidationStatusType

### TcpIp_TlsDistiguishedNameInfoType

This structure stores information about a distinguished name. It is used in different TLS messages.

| Struct Element Name | C-Type | Description |
|---|---|---|
| DnPtr | uint8* | Pointer to DN in a received message (TLS RxBuffer) |
| DnLen | uint16 | The length of the DN list |

Table 4-6 TcpIp_TlsDistiguishedNameInfoType

### TcpIp_TlsCertRequestInfoType

This structure stores information about the certificate request received by the TLS client.

| Struct Element Name | C-Type | Description |
|---|---|---|
| SignatureAndHashAlgoListPtr | uint16* | Pointer to the sign and hash algorithm list of the received message in network byte order |
| DistinguishedNameListPtr[] | TcpIp_TlsDistiguishedNameInfoType | List of pointers to DN in a received message (each DN has its own length) |
| SignatureAndHashAlgoNum | uint16 | Number of SignAndHash elements |
| DistinguishedNamesNum | uint16 | Number of distinguished names |

Table 4-7 TcpIp_TlsCertRequestInfoType

### TcpIp_TlsClientCertInfoType

This structure stores information about the certificates the client shall send in its certificate message.

| Struct Element Name | C-Type | Description |
|---|---|---|
| SignatureAndHashAlgo | TcpIp_SignatureAlgorithmIdType | SignAndHash algorithm chosen by the socket owner |
| CertIdList[] | KeyM_CertificateIdType | List of cert IDs, starting with the client leaf cert |
| CertIdListLen | uint8 | In-out-parameter. Length of the cert ID list (number of elements) |
| PrivKeyId | uint32 | Key id of the private key (of leaf cert). |

Table 4-8 TcpIp_TlsClientCertInfoType

### TcpIp_Tls12_Secret_Type

This structure stores information related to the master secret in TLS 1.2.

| Struct Element Name | C-Type | Description |
|---|---|---|
| MasterSecretPtr | uint8* | Pointer to the master secret calculated during key derivation |
| SessionIdPtr | uint8* | Pointer to the session ID sent in the Hello messages |
| SessionIdLenPtr | uint8* | Pointer to the length of the session ID |
| ClientRandomPtr | uint8* | Pointer to the client random value sent/received in the ClientHello message |

Table 4-9 TcpIp_Tls12_Secret_Type

### TcpIp_Tls13_Secret_Type

This structure stores information related to the secrets in TLS 1.3.

| Struct Element Name | C-Type | Description |
|---|---|---|
| ClientSecretPtr | uint8* | Pointer to the client secret calculated during key derivation |
| ServerSecretPtr | uint8* | Pointer to the server secret calculated during key derivation |
| ClientRandomPtr | uint8* | Pointer to the client random value sent/received in the ClientHello message |

Table 4-10 TcpIp_Tls13_Secret_Type

### TcpIp_TlsSessionTicketUserInfoType

This structure stores information related to the session tickets in TLS 1.3.

| Struct Element Name | C-Type | Description |
|---|---|---|
| currentlyAvailableTicketsNum | uint32 | The number of session tickets which are currently available |

Table 4-11 TcpIp_TlsSessionTicketUserInfoType

### TcpIp_SockAddrInet6Type

This structure defines an IPv6 address type which can be derived from the generic address structure via cast.

| Struct Element Name | C-Type | Description |
|---|---|---|
| domain | TcpIp_DomainType (uint16) | This is the code for the address format of this address |
| port | uint16 | port number |
| addr | uint32[4] | IPv6 address in network byte order |

Table 4-12 TcpIp_SockAddrInet6Type

### TcpIp_IpSecKeyMaterialInfoType

This structure defines the parameters which describes the keying material for an SA entry Pair.

| Struct Element Name | C-Type | Description |
|---|---|---|
| InboundKeyMatPtr | uint8* | Key material for inbound job |
| OutboundKeyMatPtr | uint8* | Key material for outbound job |
| InboundKeyMatLen | uint16 | Key length for inbound job |
| OutboundKeyMatLen | uint16 | Key length for outbound job |

Table 4-13 TcpIp_IpSecKeyMaterialInfoType

### TcpIp_IpSecTransformInfoType

This structure defines the parameters which describes the SA entry pair.

| Struct Element Name | C-Type | Description |
|---|---|---|
| IntegTransformIdent | TcpIp_IpSecIntegrityTransformType | Identifier for the integrity transform |
| EncryptTransformIdent | TcpIp_IpSecEncrTransformType | identifier for the encryption transform |
| EsnEnabled | boolean | Esn support flag |

Table 4-14 TcpIp_IpSecTransformInfoType

### TcpIp_IpSecTrafficSelectorType

This structure defines the traffic selectors for the SA entry pair.

| Struct Element Name | C-Type | Description |
|---|---|---|
| RemoteAddr | TcpIp_SockAddrType | Remote Ip address (in network byte order) |
| LocalPortRangeStart | uint16 | Local port range start (in host byte order) |
| LocalPortRangeEnd | uint16 | Local port range start (in host byte order) |
| RemotePortRangeStart | uint16 | Remote port range start (in host byte order) |
| RemotePortRangeEnd | uint16 | Remote port range start (in host byte order) |

| Struct Element Name | C-Type | Description |
|---|---|---|
| IpProtocol | TcpIp_IpProtocolType | Upper layer protocols supported |

Table 4-15 TcpIp_IpSecTrafficSelectorType

### TcpIp_IpSecSaInfoType

This structure defines the parameters of the Security Association (SA) entry pair.

| Struct Element Name | C-Type | Description |
|---|---|---|
| KeyMaterial | TcpIp_IpSecKeyMaterialInfoType | Key material for the SA Entry pair |
| SecurityTransform | boolean | Specifies the security transforms configured of the SA entry pair |
| InboundSpi | uint32 | Spi of the inbound traffic |
| OutboundSpi | uint32 | Spi of the outbound traffic |
| TrafficSelectors | TcpIp_IpSecTrafficSelectorType* | List of traffic selectors specified |
| NumTrafficSelectors | uint16 | Number of traffic selectors specified |
| IpsecHdrType | TcpIp_IpSecHeaderType | IPsec header type |

Table 4-16 TcpIp_IpSecSaInfoType

### TcpIp_IpSecSaStatusType

| Type Name | C-Type | Description |
|---|---|---|
| InboundSpi | uint32 | Spi of SA for inbound traffic |
| OutboundSpi | uint32 | Spi of SA for outbound traffic |
| IncomingPktCnt | uint32 | Count of incoming IPsec packets |
| OutgoingPktCnt | uint32 | Count of outgoing IPsec packets |
| SecsSinceLastRx | uint16 | Seconds elapsed since last IPsec packet Rx [0: 0xFFFF] (No wraparound) |
| SecsSinceLastTx | uint16 | Seconds elapsed since last IPsec packet Tx [0: 0xFFFF] (No wraparound) |
| IpsecProtocol | TcpIp_IpSecHeaderType | IPsec header type |

Table 4-17 TcpIp_IpSecSaStatusType

## 4.2 Services provided by TCPIP

### 4.2.1 TcpIp_GetVersionInfo

| Prototype | |
|---|---|
| `void TcpIp_GetVersionInfo (Std_VersionInfoType *versioninfo)` | |
| **Parameter** | |
| versioninfo [in] | pointer for version information |

| **Return code** | |
|---|---|
| void | void |
| **Functional Description** | |
| Get Tcp/Ip Version. | |
| **Particularities and Limitations** | |
| The module version is given decimal coded. | |
| Call context | |
| > task level | |

Table 4-18 TcpIp_GetVersionInfo

### 4.2.2 TcpIp_InitMemory

| **Prototype** | |
|---|---|
| `void` **`TcpIp_InitMemory`** `(void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| Initialize the TcpIp-internal global module state variable. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > initialization | |

Table 4-19 TcpIp_InitMemory

### 4.2.3 TcpIp_Init

| Prototype | |
|---|---|
| `void TcpIp_Init (const TcpIp_ConfigType *ConfigPtr)` | |
| **Parameter** | |
| ConfigPtr [in] | pointer to module configuration |
| **Return code** | |
| void | void |
| **Functional Description** | |
| Initialize the TcpIp module.<br>Calls all TcpIp internal init functions and sets state to 'initialized'. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > initialization | |

Table 4-20 TcpIp_Init

### 4.2.4 TcpIp_<Up>GetSocket

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_<Up>GetSocket`**`(TcpIp_DomainType Domain, TcpIp_ProtocolType Protocol, TcpIp_SocketIdType *SocketIdPtr)` | |
| **Parameter** | |
| Domain [in] | Domain / address family that will be used with the socket.<br>TCPIP_AF_INET (IPv4) or TCPIP_AF_INET6 (IPv6). |
| Protocol [in] | Transport layer protocol that will be used with the socket.<br>TCPIP_IPPROTO_TCP or TCPIP_IPPROTO_UDP |
| SocketIdPtr [out] | Pointer to socket identifier representing the requested socket.<br>This socket identifier must be provided for all further API calls which require a SocketId. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted: no free socket |
| **Functional Description** | |
| Request a socket (TCP or UDP). By this API service the TCP/IP stack is requested to allocate a new socket. | |
| **Particularities and Limitations** | |
| SocketIdPtr is only valid if return value is E_OK.<br>Each accepted incoming TCP connection also allocates a socket resource.<br>Server sockets are special because they do not need any rx and tx buffer and a lot less socket description parameters. | |
| Call context | |
| > task level | |

Table 4-21 TcpIp_<Up>GetSocket

### 4.2.5 TcpIp_GetSocketForUser

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetSocketForUser`**`(TcpIp_DomainType Domain, TcpIp_ProtocolType Protocol, TcpIp_SocketIdType *SocketIdPtr, TcpIp_SocketOwnerConfigIterType SocketOwnerIdx)` | |
| **Parameter** | |
| Domain [in] | Domain / address family that will be used with the socket.<br>TCPIP_AF_INET (IPv4) or TCPIP_AF_INET6 (IPv6). |
| Protocol [in] | Transport layer protocol that will be used with the socket.<br>TCPIP_IPPROTO_TCP or TCPIP_IPPROTO_UDP |
| SocketIdPtr [out] | Pointer to socket identifier representing the requested socket.<br>This socket identifier must be provided for all further API calls which require a SocketId. |
| SocketOwnerIdx [in] | Index of a socket owner. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted: no free socket |
| **Functional Description** | |
| Request a socket (TCP or UDP). By this API service the TCP/IP stack is requested to allocate a new socket. | |
| **Particularities and Limitations** | |
| SocketIdPtr is only valid if return value is E_OK.<br>Each accepted incoming TCP connection also allocates a socket resource.<br>Server sockets are special because they do not need any rx and tx buffer and a lot less socket description parameters. | |
| Call context | |
| > task level | |

Table 4-22 TcpIp_GetSocketForUser

### 4.2.6 TcpIp_ChangeParameter

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_ChangeParameter`** `(TcpIp_SocketIdType SocketId, TcpIp_ParamIdType ParameterId, uint8 *ParameterValue)` | |
| **Parameter** | |
| SockHnd [in] | socket handle |
| ParameterId [in] | parameter identification |
| ParameterValue [in] | parameter value |
| **Return code** | |
| Std_ReturnType | > E_OK parameter changed<br>> E_NOT_OK parameter change failed |
| **Functional Description** | |
| change socket parameter configuration<br>`TCPIP_PARAMID_UDP_CHECKSUM`: This parameter can only be changed for UDP sockets based on IPv4. The default value is TRUE, which means that the UDP checksum value 0 is not allowed for received and not used for transmission of packets. If the value is set to FALSE, transmitted packets will have checksum value 0 and received packets with checksum value 0 will be accepted.<br>**If UDP checksum offloading is active checksum value 0 could be overwritten by the hardware.**<br>`TCPIP_PARAMID_TLS_CONNECTION_ASSIGNMENT:` This parameter is used to activate the TLS connection for a specific TCP socket at runtime.<br>`TCPIP_PARAMID_TLS_CONNECTIONBASED_SESSION_RESUMPTION:` This parameter is used to request session resumption for a TCP socket where TLS was already activated during runtime. The ParameterValue is the TCP socket of the resumption base connection.<br>`TCPIP_PARAMID_TCP_MSL:` This parameter is used to change the MSL value of TCP sockets at runtime. The socket must be in state closed or listen. The unit of ParameterValue is microseconds.<br>`TCPIP_PARAMID_TCP_RETRANSMIT_TIMEOUT:` This parameter is used to change the RTT value of TCP sockets at runtime. The socket must be in state closed. The unit of ParameterValue is microseconds. For values < main function cycle time, the main function cycle time is used. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-23 TcpIp_ChangeParameter

### 4.2.7 TcpIp_Bind

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_Bind`** `(TcpIp_SocketIdType SocketId, TcpIp_LocalAddrIdType LocalAddrId, uint16 *PortPtr)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource. |
| LocalAddrId [in] | IP address identifier representing the local IP address and EthIf controller to bind the socket to. |
| PortPtr [in, out] | Pointer to memory where the local port to which the socket shall be bound is specified. In case the parameter is specified as TCPIP_PORT_ANY, the TCP/IP stack shall choose the local port automatically from the range 49152 to 65535 and shall update the parameter to the chosen value. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted (e.g. address in use) |
| **Functional Description** | |
| By this API service the TCP/IP stack is requested to bind a UDP or TCP socket to a local resource. | |
| **Particularities and Limitations** | |
| Please ensure that the input parameter SocketId of the TcpIp_Bind function is valid and must be requested before the TcpIp_Bind call via the API TcpIp_<Up>GetSocket or TcpIp_GetSockerForUser.<br>If a port is specified, the combination of port and IP address must be unique for UDP sockets. TCP allows sockets to be bound to equal local port and IP addresses. For TCP clients, the uniqueness of the local and remote port and IP address combination will be tested in TcpIp_TcpConnect(). For TCP server, TcpIp_TcpListen() will verify that there is no other TCP server with the same local port and IP address. | |
| Call context | |
| > task level | |

Table 4-24 TcpIp_Bind

### 4.2.8 TcpIp_UdpTransmit

| **Prototype** | |
|---|---|
| `TcpIp_ReturnType` **`TcpIp_UdpTransmit`** `(TcpIp_SocketIdType SocketId, uint8 *DataPtr, TcpIp_SockAddrType *RemoteAddrPtr, uint16 TotalLength)` | |
| **Parameter** | |
| SocketId [in] | socket index |
| DataPtr [in] | Pointer to a linear buffer of TotalLength bytes containing the data to be transmitted. In case DataPtr is a NULL_PTR, TcpIp shall retrieve data from SoAd via callback SoAd_CopyTxData(). |
| RemoteAddrPtr [in] | IP address and port of the remote host to transmit to. |
| TotalLength [in] | indicates the payload size of the UDP datagram. |
| **Return code** | |
| TcpIp_ReturnType | > TCPIP_OK UDP message has been forwarded to EthIf for transmission.<br>> TCPIP_E_NOT_OK UDP message could not be sent because of a permanent error, e.g. message is too long.<br>> TCPIP_E_PHYS_ADDR_MISS UDP message could not be sent because of an ARP cache miss, ARP request has been sent and upper layer may retry transmission by calling this function later again. |
| **Functional Description** | |
| Transmit UDP message. This service transmits data via UDP to a remote node. The transmission of the data is immediately performed with this function call by forwarding it to EthIf. | |
| **Particularities and Limitations** | |
| This function will return TCPIP_E_PHYS_ADDR_MISS in case the physical address could not be resolved, or no Ethernet TxBuffer was available. Differing between 'phy addr resolution is missing' and 'no buffer available' does not imply any useful information for the user.<br>UDP frames can only be transmitted if the TcpIp communication mode was requested as TCPIP_STATE_ONLINE and the corresponding local IP address was notified as TCPIP_IPADDR_STATE_ASSIGNED. If the local IP address is in state TCPIP_IPADDR_STATE_ONHOLD, transmission requests will be rejected.<br>This function may not be called reentrant from one socket owner. | |
| Call context | |
| > task level | |

Table 4-25 TcpIp_UdpTransmit

### 4.2.9 TcpIp_TcpListen

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_TcpListen`** `(TcpIp_SocketIdType SocketId, uint16 MaxChannels)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource. |
| MaxChannels [in] | Maximum number of new parallel connections established on this listen connection. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| Set TCP socket into listen mode.<br>By this API service the TCP/IP stack is requested to listen on the TCP socket specified by the socket identifier. | |
| **Particularities and Limitations** | |
| The combination of the bound local port and IP address of the socket must not yet be in use by another TCP server socket. | |
| Call context | |
| > task level | |

Table 4-26 TcpIp_TcpListen

### 4.2.10 TcpIp_TcpConnect

| Prototype | |
|---|---|
| `Std_ReturnType **TcpIp_TcpConnect** (TcpIp_SocketIdType SocketId, const TcpIp_SockAddrType *RemoteAddrPtr)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource |
| RemoteAddrPtr [in] | IP address and port of the remote host to connect to |
| **Return code** | |
| Std_ReturnType | > E_OK Connect could be triggered<br>> E_NOT_OK Connect could be triggered |
| **Functional Description** | |
| By this API service the TCP/IP stack is requested to establish a TCP connection to the configured peer. | |
| **Particularities and Limitations** | |
| The combination of the local and remote port and IP address must not yet be in use by another TCP client. | |
| Call context | |
| > task level | |

Table 4-27 TcpIp_TcpConnect

### 4.2.11 TcpIp_TcpTransmit

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_TcpTransmit`** `(TcpIp_SocketIdType SocketId, uint8 *DataPtr, uint32 AvailableLength, boolean ForceRetrieve)` | |
| **Parameter** | |
| SocketId [in] | socket index |
| DataPtr [in] | Pointer to a linear buffer of AvailableLength bytes containing the data to be transmitted. In case DataPtr is a NULL_PTR, TcpIp shall retrieve data from SoAd via callback SoAd_CopyTxData(). |
| AvailableLength [in] | Available data for transmission in bytes |
| ForceRetrieve [in] | This parameter is only valid if DataPtr is a NULL_PTR. Indicates how the TCP/IP stack retrieves data from SoAd if DataPtr is a NULL_PTR.<br>TRUE: the whole data indicated by availableLength shall be retrieved from the upper layer via one or multiple SoAd_CopyTxData() calls within the context of this transmit function. FALSE: The TCP/IP stack may retrieve up to availableLength data from the upper layer. It is allowed to retrieve less than availableLength bytes. Note: Not retrieved data will be provided by SoAd with the next call to TcpIp_TcpTransmit (along with new data if available). |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| Transmit TCP message  This service requests transmission of data via TCP to a remote node. The transmission of the data is decoupled. | |
| **Particularities and Limitations** | |
| The TCP segment(s) are sent dependent on runtime factors (e.g. receive window) and configuration parameter (e.g. Nagle algorithm).<br>TCP data can only be transmitted if the TcpIp communication mode was requested as TCPIP_STATE_ONLINE and the corresponding local IP address was notified as TCPIP_IPADDR_STATE_ASSIGNED. If the local IP address is in state TCPIP_IPADDR_STATE_ONHOLD, transmission requests will be rejected. | |
| Call context | |
| > task level | |

Table 4-28 TcpIp_TcpTransmit

### 4.2.12 TcpIp_TcpReceived

| **Prototype** | |
|---|---|
| `Std_ReturnType **TcpIp_TcpReceived** (TcpIp_SocketIdType SocketId, uint32 Length)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource |
| Length [in] | Number of bytes finally consumed by the upper layer |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| By this API service the reception of socket data is confirmed to the TCP/IP stack. | |
| **Particularities and Limitations** | |
| Increase the TCP receive window of the socket specified by SocketId considering the number of finally consumed bytes specified by Length. | |
| Call context | |
| > interrupt or task level | |

Table 4-29 TcpIp_TcpReceived

### 4.2.13 TcpIp_Close

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_Close`** `(TcpIp_SocketIdType SocketId, boolean Abort)` | |
| **Parameter** | |
| SocketId [in] | Socket handle identifying the local socket resource. |
| Abort [in] | TRUE: connection will immediately be terminated by sending a RST-Segment and releasing all related resources.<br>FALSE: connection will be terminated after performing a regular connection termination handshake and releasing all related resources. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted.<br>> E_NOT_OK The request has not been accepted. |
| **Functional Description** | |
| Close the socket. By this API service the TCP/IP stack is requested to close the socket and release all related resources.<br>The service TcpIp_Close() shall perform the following actions for the socket specified by SocketId in case it is a TCP socket:<br>(a) if the connection is active and<br>(a1) abort = FALSE: the connection shall be terminated after performing a regular connection termination handshake and releasing all related resources.<br>(a2) abort = TRUE: connection shall immediately be terminated by sending a RST-Segment and releasing all related resources.<br>(b) if the socket is in the Listen state, the Listen state shall be left immediately and related resources shall be released. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-30 TcpIp_Close

### 4.2.14 TcpIp_IcmpV6Transmit

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_IcmpV6Transmit`**`(TcpIp_LocalAddrIdType LocalIpAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 HopLimit, uint8 Type, uint8 Code, uint16 DataLength, const uint8 *DataPtr)` | |
| **Parameter** | |
| LocalIpAddrId [in] | Local address identifier. (must reference an IPv6 address) |
| RemoteAddrPtr [in] | Destination address. (must be an IPv6 address) |
| HopLimit [in] | HopLimit value of the outgoing IPv6 packet. |
| Type [in] | Value of the Type field in the ICMPv6 message header. |
| Code [in] | Value of the Code field in the ICMPv6 message header. |
| DataLength [in] | Length of the message payload. |
| DataPtr [in] | Message payload. |
| **Return code** | |
| TcpIp_ReturnType | > E_OK Message was transmitted.<br>> E_NOT_OK Message was not transmitted due to pending link-layer address resolution. |
| **Functional Description** | |
| Sends a raw ICMPv6 message of specified Type and Code. | |
| **Particularities and Limitations** | |
| This function may also be used to send ICMPv6 Echo Requests.<br>Replies to send ICMPv6 messages can be received via the <Up>_IcmpMsgHandler callback.<br>ICMP frames can only be transmitted if the TcpIp communication mode was requested as TCPIP_STATE_ONLINE and the corresponding local IP address was notified as TCPIP_IPADDR_STATE_ASSIGNED. If the local IP address is in state TCPIP_IPADDR_STATE_ONHOLD, transmission requests will be rejected. | |
| Call context | |
| > task level | |

Table 4-31 TcpIp_IcmpV6Transmit

### 4.2.15 TcpIp_RequestIpAddrAssignment

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_RequestIpAddrAssignment`** `(TcpIp_LocalAddrIdType LocalAddrId, TcpIp_IpAddrAssignmentType Type, TcpIp_SockAddrType *LocalIpAddrPtr, uint8 Netmask, TcpIp_SockAddrType *DefaultRouterPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | IP address index specifying the IP address for which an assignment shall be initiated. |
| Type [in] | type of IP address assignment which shall be initiated |
| LocalIpAddrPtr [in] | pointer to structure containing the IP address which shall be assigned to the EthIf controller indirectly specified via LocalAddrId. Note: This parameter is only used in case the parameters Type is set to TCPIP_IPADDR_ASSIGNMENT_STATIC. To use the optionally configured default address of static addresses the address 0.0.0.0 or [::] has to be used. |
| Netmask [in] | Network mask of IPv4 address or address prefix of IPv6 address in CIDR Notation. |
| DefaultRouterPtr [in] | Pointer to struct where the IP address of the default router (gateway) is stored (struct member 'port' is not used and of arbitrary value) (IPv4 only) |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| Initiate the local IP address assignment for the IP address specified by LocalAddrId. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-32 TcpIp_RequestIpAddrAssignment

### 4.2.16 TcpIp_ReleaseIpAddrAssignment

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_ReleaseIpAddrAssignment`** `(TcpIp_LocalAddrIdType LocalAddrId)` | |
| **Parameter** | |
| LocalAddrId [in] | IP address index specifying the IP address for which an assignment shall be released. |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| Release the local IP address assignment for the IP address specified by LocalAddrId. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-33 TcpIp_ReleaseIpAddrAssignment

### 4.2.17 TcpIp_ReleaseSpecificIpAddrAssignment

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_ReleaseIpAddrAssignment`** `(TcpIp_LocalAddrIdType LocalAddrId, TcpIp_IpAddrAssignmentType AssignmentType)` | |
| **Parameter** | |
| LocalAddrId [in] | IP address index specifying the IP address for which an assignment shall be released. |
| AssignmentType [in] | Type of the assignment that shall be released:<br>TCPIP_IPADDR_ASSIGNMENT_STATIC<br>TCPIP_IPADDR_ASSIGNMENT_DHCP<br>TCPIP_IPADDR_ASSIGNMENT_LINKLOCAL |
| **Return code** | |
| Std_ReturnType | > E_OK The request has been accepted<br>> E_NOT_OK The request has not been accepted |
| **Functional Description** | |
| This API may be used if multiple IP address assignment methods are configured for an IPv4 address and only a specific address assignment shall be released.<br>If all assignments for a local address shall be released use TcpIp_ReleaseIpAddrAssignment(). | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-34 TcpIp_ReleaseIpAddrAssignment

### 4.2.18 TcpIp_GetCtrlIdx

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetCtrlIdx`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 *CtrlIdxPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | Local address identifier implicitely specifing the EthIf controller that shall be returned |
| CtrlIdxPtr [out] | Pointer to the memory where the index of the EthIf controller related to LocalAddrId is stored |
| **Return code** | |
| Std_ReturnType | > E_OK The request was successful<br>> E_NOT_OK The request was not successful |
| **Functional Description** | |
| TcpIp_GetCtrlIdx returns the index of the EthIf controller related to LocalAddrId. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-35 TcpIp_GetCtrlIdx

### 4.2.19 TcpIp_GetIpAddr

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetIpAddr`** `(TcpIp_LocalAddrIdType LocalAddrId, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr, TCPIP_P2V(uint8) NetmaskPtr, TCPIP_P2V(TcpIp_SockAddrType) DefaultRouterPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | local address identifier |
| IpAddrPtr [out] | Pointer to a struct where the IP address is stored. (only struct members family and address will be updated) |
| NetmaskPtr [out] | Pointer to memory where Network mask of IPv4 address or address prefix of IPv6 address in CIDR notation is stored. |
| DefaultRouterPtr [out] | Pointer to struct where the IP address of the default router (gateway) is stored (only struct members family and address will be updated) |
| **Return code** | |
| Std_ReturnType | > E_OK The request was successful<br>> E_NOT_OK The request was not successful. |
| **Functional Description** | |
| Obtains the local IP address actually used by LocalAddrId, the netmask and default router. | |
| **Particularities and Limitations** | |
| The output parameters IpAddrPtr, NetmaskPtr, DefaultRouterPtr may be set to NULL_PTR if the information is not required. | |
| Call context | |
| > task level | |

Table 4-36 TcpIp_GetIpAddr

### 4.2.20 TcpIp_GetIpAddrCfgSrc

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetIpAddrCfgSrc`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 *CfgSrcPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | local address identifier |
| AddrPtr [in] | address for which the configuration source shall be returned |
| CfgSrcPtr [out] | configuration source of the address |
| **Return code** | |
| Std_ReturnType | > E_OK address is valid and *CfgSrcPtr has be set<br>> E_NOT_OK address invalid or not assigned to this node. *CfgSrcPtr not modified. |
| **Functional Description** | |
| Get the configuration source of an IP address. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-37 TcpIp_GetIpAddrCfgSrc

### 4.2.21 TcpIp_GetRemNetAddr

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetRemNetAddr`** `(TcpIp_SockHndType SockHnd, TCPIP_P2V(TcpIp_SockAddrType)*SockAddrPtr)` | |
| **Parameter** | |
| SockHnd [in] | socket handle |
| SockAddrPtr [out] | Pointer to a socket address element |
| **Return code** | |
| Std_ReturnType | > E_OK reading the socket address was successful<br>> E_NOT_OK remote IP address could not be obtained |
| **Functional Description** | |
| The API provides the remote address of an established TCP socket connection. | |
| **Particularities and Limitations** | |
| SockAddrPtr is a pointer to a TcpIp memory area where the remote address is stored. The pointer is valid until the socket is closed. | |
| Call context | |
| > task level | |

Table 4-38 TcpIp_GetRemNetAddr

### 4.2.22 TcpIp_GetLocSockAddr

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetLocSockAddr`** `(TcpIp_SockHndType SockHnd, TcpIp_SockAddrType *SockPtr)` | |
| **Parameter** | |
| SockHnd [in] | socket handle |
| SockPtr [out] | pointer were the local socket address shall be written to |
| **Return code** | |
| Std_ReturnType | > E_OK socket address was written successfully<br>> E_NOT_OK error, socket address is not updated |
| **Functional Description** | |
| Read the local socket address of one socket. | |
| **Particularities and Limitations** | |
| : This is a Vector extension | |
| Call context | |
| > task level | |

Table 4-39 TcpIp_GetLocSockAddr

### 4.2.23 TcpIp_GetPhysAddr

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetPhysAddr`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 *PhysAddrPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | Local address identifier implicitly specifying the EthIf controller for which the physical address shall be obtained. |
| PhysAddrPtr [out] | Pointer to the memory where the physical source address (MAC address) in network byte order is stored |
| **Return code** | |
| Std_ReturnType | > E_OK The request was successful<br>> E_NOT_OK The request was not successful, e.g. no unique Ctrl specified via IpAddrId. |
| **Functional Description** | |
| Obtains the physical source address used by the EthIf controller implicitly specified via LocalAddrId. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-40 TcpIp_GetPhysAddr

### 4.2.24 TcpIp_GetRemotePhysAddr

| **Prototype** | |
|---|---|
| `TcpIp_ReturnType` **`TcpIp_GetRemotePhysAddr`** `(uint8 CtrlIdx, TcpIp_SockAddrType *IpAddrPtr, uint8 *PhysAddrPtr, boolean initRes)` | |
| **Parameter** | |
| CtrlIdx [in] | EthIf controller index to identify the related ARP/NDP table. |
| IpAddrPtr [in] | specifies the IP address for which the physical address shall be retrieved |
| initRes [in] | specifies if the address resolution shall be initiated (TRUE) or not (FALSE) in case the physical address related to the specified IP address is currently unknown. |
| PhysAddrPtr [out] | Pointer to the memory where the physical address (MAC address) related to the specified IP address is stored in network byte order. |
| **Return code** | |
| TcpIp_ReturnType | > TCPIP_E_OK specified IP address resolved, physical address provided via PhysAddrPtr<br>> TCPIP_E_PHYS_ADDR_MISS physical address currently unknown (address resolution initiated if initRes set to TRUE) |
| **Functional Description** | |
| Lookup the physical address for a remote network address.<br>TcpIp_GetRemotePhysAddr queries the IP/physical address translation table specified by CtrlIdx and returns the physical address related to the IP address specified by IpAddrPtr. In case no physical address can be retrieved and parameter initRes is TRUE, address resolution for the specified IP address is initiated on the local network. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-41 TcpIp_GetRemotePhysAddr

### 4.2.25 TcpIp_DhcpReadOption

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_DhcpReadOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint8 Option, uint8 *DataLength, uint8 *DataPtr)` | |
| **Parameter** | |
| LocalIpAddrId [in] | IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be read. |
| Option [in] | DHCP option, e.g. Host Name |
| DataLength [in, out] | As input parameter, contains the length of the provided data buffer. Will be overwritten with the length of the actual data. |
| DataPtr [out] | Pointer to memory containing DHCP option data. |
| **Return code** | |
| Std_ReturnType | > E_OK requested data retrieved successfully.<br>> E_NOT_OK requested data could not be retrieved. |
| **Functional Description** | |
| Read user option data obtained from incoming DHCPv4 messages.<br>By this API service the TCP/IP stack retrieves DHCP option data identified by parameter option for already received DHCP options. | |
| **Particularities and Limitations** | |
| The API can only be used to read out the content of statically configured DHCP options. | |
| Call context | |
| > interrupt or task level | |

Table 4-42 TcpIp_DhcpReadOption

### 4.2.26 TcpIp_DhcpWriteOption

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_DhcpWriteOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint8 Option, uint8 DataLength, const uint8 *DataPtr)` | |
| **Parameter** | |
| LocalIpAddrId [in] | IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be written. |
| Option [in] | DHCP option, e.g. Host Name |
| DataLength [in] | Length of DHCP option data in bytes. Set to 0 to clear previously stored data |
| DataPtr [in] | Pointer to memory containing DHCP option data |
| **Return code** | |
| Std_ReturnType | > E_OK no error occurred.<br>> E_NOT_OK DHCP option data could not be written. |
| **Functional Description** | |
| Set user option data for outgoing DHCPv4 messages.<br>Invoking the API with DataLength as 0 will clear the user option data previously stored. | |

| |
|---|
| By this API service the TCP/IP stack writes the DHCP option data identified by parameter option. |
| **Particularities and Limitations** |
| Old similar Vector API was TcpIp_SetDhcpV4TxOption.<br>The API can only be used to write the content of statically configured DHCP options. |
| Call context |
| > interrupt or task level |

Table 4-43 TcpIp_DhcpWriteOption

### 4.2.27 TcpIp_DhcpV6ReadOption

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_DhcpV6ReadOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint16 Option, uint16 *DataLength, uint8 *DataPtr)` | |
| **Parameter** | |
| LocalIpAddrId [in] | IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be read. |
| Option [in] | DHCP option, e.g. Host Name |
| DataLength [in,out] | As input parameter, contains the length of the provided data buffer. Will be overwritten with the length of the actual data. |
| DataPtr [out] | Pointer to memory containing DHCPv6 option data. |
| **Return code** | |
| Std_ReturnType | > E_OK requested data retrieved successfully.<br>> E_NOT_OK requested data could not be retrieved. |
| **Functional Description** | |
| Read user option data obtained from incoming DHCPv6 messages.<br>By this API service the TCP/IP stack retrieves DHCP option data identified by parameter option for already received DHCP options. | |
| **Particularities and Limitations** | |
| The API can only be used to read out the content of statically configured DHCPv6 options. | |
| Call context | |
| > interrupt or task level | |

Table 4-44 TcpIp_DhcpV6ReadOption

### 4.2.28 TcpIp_DhcpV6WriteOption

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_DhcpV6WriteOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint16 Option, uint16 DataLength, const uint8 *DataPtr)` | |
| **Parameter** | |
| LocalIpAddrId [in] | IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be read. |
| Option [in] | DHCP option, e.g. Host Name |
| DataLength [in] | Length of option data in bytes. Set to 0 to clear previously stored data |
| DataPtr [in] | Payload of the option. Pointer must be valid until it is updated by calling this function. (Ignored if DataLength is 0) |
| **Return code** | |
| Std_ReturnType | > E_OK No error occurred<br>> E_NOT_OK DHCP option data could not be written |
| **Functional Description** | |
| Set user option data for outgoing DHCPv6 messages.<br>Invoking the API with DataLength as 0 will clear the option data previously stored. | |
| **Particularities and Limitations** | |
| Old similar Vector API was TcpIp_SetDhcpV6TxOption.<br>The API can only be used to write the content of statically configured DHCPv6 options. | |
| Call context | |
| > interrupt or task level | |

Table 4-45 TcpIp_DhcpV6WriteOption

### 4.2.29 TcpIp_GetDhcpTimeoutInfo

| **Prototype** | |
|---|---|
| `boolean` **`TcpIp_GetDhcpTimeoutInfo`** `(uint8 CtrlIdx, uint8 IpVer)` | |
| **Parameter** | |
| EthIfCtrlIdx [in] | controller index |
| IpVer [in] | IP version |
| **Return code** | |
| boolean | > TRUE there was a DHCP timeout<br>> FALSE there was no DHCP timeout |
| **Functional Description** | |
| check if a DHCP address finding timeout occurred | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > interrupt or task level | |

Table 4-46 TcpIp_GetDhcpTimeoutInfo

### 4.2.30 TcpIp_DhcpV4_GetStatus

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_DhcpV4_GetStatus`**`( IpV4_AddrIdType IpAddrId, uint8* DhcpStatePtr )` | |
| **Parameter** | |
| IpAddrId [in] | local address identifier |
| DhcpStatePtr [out] | current, simplified, DHCP status of the IP address |
| **Return code** | |
| Std_ReturnType | > E_OK the DhcpState is valid<br>> E_NOT_OK given address is invalid or has no DHCP configured or the IP controller of the given address is not in state online active, |
| **Functional Description** | |
| Returns the current DCHP state of a given IP address.<br><br>The following simplified DHCP states are supported:<br>TCPIP_DHCP_SIMPLIFIED_STATE_NOT_ACTIVE: DHCP client not active.<br>TCPIP_DHCP_SIMPLIFIED_STATE_NO_DISCVR_SENT: No discover sent.<br>TCPIP_DHCP_SIMPLIFIED_STATE_DISCVR_SENT_NO_ANSWR_RCVD_YET: Discover sent but no answer received.<br>TCPIP_DHCP_SIMPLIFIED_STATE_OFFER_RCVD_WITHOUT_REQ: DHCP offer received, no request sent.<br>TCPIP_DHCP_SIMPLIFIED_STATE_REQ_SENT_NO_ACK_RCVD_YET: DHCP request sent, no ACK received.<br>TCPIP_DHCP_SIMPLIFIED_STATE_ACK_RCVD_DHCP_ADDR_ASSIGND: DHCP address assigned to current interface. | |
| **Particularities and Limitations** | |
| This is a Vector extension. | |
| Call context | |
| > task level | |

Table 4-47 TcpIp_DhcpV4_GetStatus

### 4.2.31 TcpIp_RequestComMode

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_RequestComMode`** `(uint8 CtrlIdx, TcpIp_StateType State)` | |
| **Parameter** | |
| CtrlIdx [in] | EthIf controller index to identify the communication network where the TcpIp state is requested |
| State [in] | Requested TcpIp state |
| **Return code** | |
| Std_ReturnType | > E_OK request was accepted<br>> E_NOT_OK request was not accepted |
| **Functional Description** | |
| By this API service the TCP/IP stack is requested to change the TcpIp state of the communication network identified by EthIf controller index. | |
| **Particularities and Limitations** | |
| After initializing TcpIp the TcpIp_Mainfunctions must be called at least once before a new ComMode can be requested.<br>After calling TcpIp_RequestComMode it is required that all TcpIp_Mainfunctions are called at least once before calling TcpIp_RequestComMode again. | |
| Call context | |
| > task level | |

Table 4-48 TcpIp_RequestComMode

### 4.2.32 TcpIp_DiagDataReadAccess

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_DiagDataReadAccess`** `(TcpIp_SockHndType SockHnd, TcpIp_DiagParamsType DataID, uint8 *DataPtr, uint16 *DataLenPtr)` | |
| **Parameter** | |
| SockHnd [in] | socket handle |
| DataID [in] | data identifier |
| DataPtr [out] | pointer for diagnostic data |
| DataLenPtr [in,out] | pointer for maximum / actual length of diagnostic data in bytes |
| **Return code** | |
| Std_ReturnType | > E_OK diagnostic data written<br>> E_NOT_OK invalid parameter (data identifier not found, NULL_PTR parameter, invalid length) |
| **Functional Description** | |
| diagnostic data read access | |
| **Particularities and Limitations** | |
| The memory fragment DataPtr points to should be aligned properly regarding the expected returned type / struct. Data is only written if RetValue is E_OK.<br>This is a Vector extension. | |
| Call context | |
| > task level | |

Table 4-49 TcpIp_DiagDataReadAccess

### 4.2.33 TcpIp_ClearARCache

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_ClearARCache`** `(TcpIp_LocalAddrIdType LocalAddrId)` | |
| **Parameter** | |
| LocalAddrId [in] | Local address identifier implicitly specifying the IPv4/IPv6 controller that shall be cleared. |
| **Return code** | |
| Std_ReturnType | > E_OK The request was successful<br>> E_NOT_OK The request was not successful |
| **Functional Description** | |
| TcpIp_ClearARCache clears the address resolution cache. | |
| **Particularities and Limitations** | |
| Currently only IPv6/NDP is supported | |
| Call context | |
| > task level | |

Table 4-50 TcpIp_ClearARCache

### 4.2.34 TcpIp_GetArpCacheEntries

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetArpCacheEntries`** `(uint8 ctrlIdx, uint32 *numberOfElements, TcpIp_ArpCacheEntryType *entryListPtr)` | |
| **Parameter** | |
| ctrlIdx [in] | EthIf controller index to identify the related ARP table. |
| numberOfElements[in,out] | In: Maximum number of entries that can be stored in the output buffer entryListPtr.<br>Out: Number of entries that are written into the output buffer entryListPtr<br>(Total number of all entries in the cache if input value is 0). |
| entryListPtr[out] | Pointer to memory where the list of cache entries shall be stored.<br>if *numberOfElements is 0 , the output may be NULL_PTR |
| **Return code** | |
| Std_ReturnType | > E_OK physical address cache could be read.<br>> E_NOT_OK physical address cache could not be read.<br>(i.e. no IPv4 instance active on this controller) |
| **Functional Description** | |
| Returns entries that are currently stored in the IPv4 address resolution cache.<br>Copies entries from the physical address cache of the IPv4 instance that is active on the EthIf controller specified by ctrlIdx into a user provided buffer. The function will copy all or numberOfElements into the output list. If input value of numberOfElements is 0 the function will not copy any data but only return the number of entries in the cache. entryListPtr may be NULL_PTR in this case. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > interrupt or task level | |

Table 4-51 TcpIp_GetArpCacheEntries

### 4.2.35 TcpIp_AddArpCacheEntry

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_AddArpCacheEntry`** `(uint8 ctrlIdx, const TcpIp_SockAddrType * RemoteAddrPtr, const TcpIp_uint8 * PhysAddrPtr)` | |
| **Parameter** | |
| CtrlIdx [in] | EthIf controller index to identify the related ARP table. |
| RemoteAddrPtr [in] | The IPv4 address of the new ARP entry. |
| PhysAddrPtr [in] | The physical address (MAC) of the new ARP entry. |
| **Return code** | |
| Std_ReturnType | > E_OK ARP entry is created.<br>> E_NOT_OK entry is not created. |

| Functional Description |
|---|
| RemoteAddrPtr must be a valid IPv4 Address and PhysAddrPtr must be a physical address (MAC address with 6 Bytes).<br>ARP entries created by this API will never expire and can not be updated via the normal ARP process. To remove those entries use the API TcpIp_RemoveArpCacheEntry(). |
| **Particularities and Limitations** |
| This API only works with IPv4 addresses.<br>The Address must not be part of the static ARP table. |
| **Call context** |
| > interrupt or task level |

Table 4-52 TcpIp_AddArpCacheEntry

### 4.2.36 TcpIp_RemoveArpCacheEntry

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_ RemoveArpCacheEntry`** `(uint8 ctrlIdx, const TcpIp_SockAddrType * RemoteAddrPtr)` | |
| **Parameter** | |
| CtrlIdx [in] | EthIf controller index to identify the related ARP table. |
| RemoteAddrPtr [in] | The IPv4 address of the ARP entry that shall be removed. |
| **Return code** | |
| Std_ReturnType | > E_OK ARP entry is removed.<br>> E_NOT_OK entry is not removed. |
| **Functional Description** | |
| RemoteAddrPtr must be a valid IPv4 address. | |
| **Particularities and Limitations** | |
| This API only works with IPv4 addresses.<br>The Address must not be part of the static ARP table. | |
| **Call context** | |
| > interrupt or task level | |

Table 4-53 TcpIp_RemoveArpCacheEntry

### 4.2.37 TcpIp_GetNdpCacheEntries

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetNdpCacheEntries`** `(uint8 ctrlIdx, uint32 *numberOfElements, TcpIp_NdpCacheEntryType *entryListPtr)` | |
| **Parameter** | |
| ctrlIdx [in] | EthIf controller index to identify the related NDP table. |
| numberOfElements[in,out] | In: Maximum number of entries that can be stored in Parameters output entryListPtr.<br>Out: Number of entries written to output entryListPtr<br>(Number of all entries in the cache if input value is 0). |
| entryListPtr[out] | Pointer to memory where the list of cache entries shall be stored.<br>May only be NULL_PTR if *numberOfElements is 0. |
| **Return code** | |
| Std_ReturnType | > E_OK physical address cache could be read.<br>> E_NOT_OK physical address cache could not be read.<br>(i.e. no IPv6 instance active on this controller) |
| **Functional Description** | |
| Returns entries that are currently stored in the IPv6 link layer address resolution cache.<br>Copies entries from the physical address cache of the IPv6 instance that is active on the EthIf controller specified by ctrlIdx into a user provided buffer. The function will copy all or numberOfElements into the output list. If input value of numberOfElements is 0 the function will not copy any data but only return the number of entries in the cache. EntryListPtr may be NULL_PTR in this case. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > interrupt or task level | |

Table 4-54 TcpIp_GetNdpCacheEntries

### 4.2.38 TcpIp_SendGratuitousArpReq

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_SendGratuitousArpReq`**`(TcpIp_LocalAddrIdType LocalAddrId)` | |
| **Parameter** | |
| LocalAddrId [in] | Local address identifier implicitly specifing the EthIf controller for which the physical address shall be obtained. |
| **Return code** | |
| Std_ReturnType | > E_OK the gratuitous ARP request was sent using the related lower layer APIs.<br>> E_NOT_OK otherwise. |
| **Functional Description** | |
| Sends a gratuitous ARP request packet as specified in [RFC2002 4.6. second indentation] | |
| **Particularities and Limitations** | |
| This is a Vector extension.<br>Gratuitous ARP requests can only be transmitted if the TcpIp communication mode was requested as TCPIP_STATE_ONLINE and the corresponding local IP address was notified as TCPIP_IPADDR_STATE_ASSIGNED. If the local IP address is in state TCPIP_IPADDR_STATE_ONHOLD, transmission requests will be rejected. | |
| Call context | |
| > task level | |

Table 4-55 TcpIp _SendGratuitousArpReq

### 4.2.39 IpV4_GetLastDuplicateDhcpAddrDid

| **Prototype** | |
|---|---|
| `FUNC(Std_ReturnType, IPV4_CODE) IpV4_`**`GetLastDuplicateDhcpAddrDid`**`(uint8 IpCtrlIdx, CONSTP2VAR(IpBase_AddrInType, AUTOMATIC, IPV4_APPL_DATA) IpAddrPtr, CONSTP2VAR(IpBase_EthPhysAddrType, AUTOMATIC, IPV4_APPL_DATA) PhysAddrPtr)` | |
| **Parameter** | |
| IpCtrlIdx [in] | Ip controller index |
| IpAddrPtr [out] | Specifies the memory where the IP address shall be stored for which a duplicated DHCP address assignment has been detected. |
| PhysAddrPtr [out] | Specifies the memory where the physical address shall be stored for which a duplicated DHCP address assignment has been detected. |
| **Return code** | |
| Std_ReturnType | > E_OK the given IpCtrlIdx was valid and the DID information was stored in the given data structures.<br>> E_NOT_OK otherwise. |
| **Functional Description** | |
| Returns the DID for DEM event TCPIP_E_DUPLICATE_DHCP_ADDR | |
| **Particularities and Limitations** | |
| This is a Vector extension. | |
| Call context | |
| > task level | |

Table 4-56 IpV4_GetLastDuplicateDhcpAddrDid

### 4.2.40 TcpIp_MainFunction

| **Prototype** | |
|---|---|
| `void `**`TcpIp_MainFunction`**` (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunction handles the module internal state handling.<br>The TcpIp_MultiPartitionMainFunction will be called with the TCPIP_SINGLECORE_APPLID.<br>Calls all TcpIp-internal MainFunction functions. This function has to be called cyclically. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is not enabled | |
| Call context | |
| > task level | |

Table 4-57 TcpIp_MainFunction

### 4.2.41 TcpIp_MainFunctionRx

| Prototype | |
|---|---|
| `void **TcpIp_MainFunctionRx** (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunctionRx handles all received data and forwards information to upper layers.<br>The TcpIp_MultiPartitionMainFunctionRX will be called with the TCPIP_SINGLECORE_APPLID.<br>In case this function is called, the TcpIp_MainFunctionTx and TcpIp_MainFunctionState HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is not enabled | |
| Call context | |
| > task level | |

Table 4-58 TcpIp_MainFunctionRx

### 4.2.42 TcpIp_MainFunctionTx

| Prototype | |
|---|---|
| `void **TcpIp_MainFunctionTx** (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunctionTx handles all pending transmit requests from the upper layers and transmits the data.<br>The TcpIp_MultiPartitionMainFunctionTx will be called with the TCPIP_SINGLECORE_APPLID.<br>In case this function is called, the TcpIp_MainFunctionRx and TcpIp_MainFunctionState HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is not enabled | |
| Call context | |
| > task level | |

Table 4-59 TcpIp_MainFunctionTx

### 4.2.43 TcpIp_MainFunctionState

| **Prototype** | |
|---|---|
| `void TcpIp_MainFunctionState (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunction handles the module internal state handling and timing actions.<br>The TcpIp_MultiPartitionMainFunctionState will be called with the TCPIP_SINGLECORE_APPLID.<br>In case this function is called, the TcpIp_MainFunctionRx and TcpIp_MainFunctionTx HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is not enabled | |
| Call context | |
| > task level | |

Table 4-60 TcpIp_MainFunctionState

### 4.2.44 TcpIp_MainFunction_<Partition Name>

| **Prototype** | |
|---|---|
| `void TcpIp_MainFunction_<Parition Name> (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunction_<Partition Name> handles the module internal state handling.<br>The TcpIp_MultiPartitionMainFunction will be called with the related AppId.<br>Calls all TcpIp-internal MainFunction functions. This function has to be called cyclically. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is not enabled | |
| Call context | |
| > task level | |

Table 4-61 TcpIp_MainFunction

### 4.2.45 TcpIp_MainFunctionRx_<Partition Name>

| Prototype | |
|---|---|
| `void TcpIp_MainFunctionRx_<Parition Name> (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunctionRx handles all received data and forwards information to upper layers.<br>The TcpIp_MultiPartitionMainFunctionRx will be called with the related AppId.<br>In case this function is called, the TcpIp_MainFunctionTx and TcpIp_MainFunctionState HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is enabled | |
| Call context | |
| > task level | |

Table 4-62 TcpIp_MainFunctionRx

### 4.2.46 TcpIp_MainFunctionTx_<Partition Name>

| Prototype | |
|---|---|
| `void TcpIp_MainFunctionTx_<Parition Name> (void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunctionTx handles all pending transmit requests from the upper layers and transmits the data.<br>The TcpIp_MultiPartitionMainFunctionTx will be called with the related AppId.<br>In case this function is called, the TcpIp_MainFunctionRx and TcpIp_MainFunctionState HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is enabled | |
| Call context | |
| > task level | |

Table 4-63 TcpIp_MainFunctionTx

### 4.2.47 TcpIp_MainFunctionState_<Partition Name>

| Prototype | |
|---|---|
| `void` **`TcpIp_MainFunctionState_<Partition Name>`** `(void)` | |
| **Parameter** | |
| void [in] | none |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MainFunction handles the module internal state handling and timing actions.<br>The TcpIp_MultiPartitionMainFunctionState will be called with the related ApplId.<br>In case this function is called, the TcpIp_MainFunctionRx and TcpIp_MainFunctionTx HAVE TO BE called as well and the TcpIp_MainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before<br>This API is only available if Multicore is enabled | |
| Call context | |
| > task level | |

Table 4-64 TcpIp_MainFunctionState

### 4.2.48 TcpIp_MultiPartitionMainFunction

| Prototype | |
|---|---|
| `void` **`TcpIp_MultiPartitionMainFunction`** `(TcpIp_OsApplicationType ApplId)` | |
| **Parameter** | |
| `ApplId` [in] | Application Id of the invoking application. |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MultiPartitionMainFunction handles the module internal state handling.<br>Calls all TcpIp-internal MainFunction functions. This function has to be called cyclically. | |
| **Particularities and Limitations** | |
| Init has to be called before | |
| Call context | |
| > task level | |

Table 4-65 TcpIp_MainFunction

### 4.2.49 TcpIp_MultiPartitionMainFunctionRx

| **Prototype** | |
|---|---|
| `void` **`TcpIp_MultiPartitionMainFunctionRx`** `(TcpIp_OsApplicationType ApplId)` | |
| **Parameter** | |
| `ApplId` [in] | Application Id of the invoking application. |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MultiPartitionMainFunctionRx handles all received data and forwards information to upper layers.<br>In case this function is called, the TcpIp_ MultiPartitionMainFunctionTx and TcpIp_ MultiPartitionMainFunctionState HAVE TO BE called as well and the TcpIp_ MultiPartitionMainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before | |
| Call context | |
| > task level | |

Table 4-66 TcpIp_MainFunctionRx

### 4.2.50 TcpIp_MultiPartitionMainFunctionTx

| **Prototype** | |
|---|---|
| `void` **`TcpIp_ MultiPartitionMainFunctionTx`** `(TcpIp_OsApplicationType ApplId)` | |
| **Parameter** | |
| `ApplId` [in] | Application Id of the invoking application. |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MultiPartitionMainFunctionTx handles all pending transmit requests from the upper layers and transmits the data.<br>In case this function is called, the TcpIp_ MultiPartitionMainFunctionRx and TcpIp_ MultiPartitionMainFunctionState HAVE TO BE called as well and the TcpIp_ MultiPartitionMainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before | |
| Call context | |
| > task level | |

Table 4-67 TcpIp_MainFunctionTx

### 4.2.51 TcpIp_MultiPartitionMainFunctionState

| Prototype | |
|---|---|
| `void` **`TcpIp_MultiPartitionMainFunctionState`** `(TcpIp_OsApplicationType ApplId)` | |
| **Parameter** | |
| `ApplId` [in] | Application Id of the invoking application. |
| **Return code** | |
| void | void |
| **Functional Description** | |
| The TcpIp MultiPartitionMainFunctionState handles the module internal state handling and timing actions. In case this function is called, the TcpIp_ MultiPartitionMainFunctionRx and TcpIp_ MultiPartitionMainFunctionTx HAVE TO BE called as well and the TcpIp_ MultiPartitionMainFunction MUST NOT be called. | |
| **Particularities and Limitations** | |
| Init has to be called before | |
| Call context | |
| > task level | |

Table 4-68 TcpIp_MainFunctionState

### 4.2.52 TcpIp_GetLocNetAddr

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetLocNetAddr`** `(TcpIp_LocalAddrIdType LocalAddrId, TcpIp_NetAddrType *NetAddrPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | local address identifier |
| NetAddrPtr [out] | pointer for the local network address |
| **Return code** | |
| Std_ReturnType | > E_OK local network address returned<br>> E_NOT_OK local network address access failed |
| **Functional Description** | |
| This function returns the current IP address for a given controller. | |
| **Particularities and Limitations** | |
| This API is deprecated and will be removed in future revisions of this module. | |
| Call context | |
| > task level | |

Table 4-69 TcpIp_GetLocNetAddr

### 4.2.53 TcpIp_GetLocNetMask

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetLocNetMask`** `(TcpIp_LocalAddrIdType LocalAddrId, TcpIp_NetAddrType *NetMaskPtr)` | |
| **Parameter** | |
| LocalAddrId [in] | local address identifier |
| NetMaskPtr [out] | pointer for the local network mask |
| **Return code** | |
| Std_ReturnType | > E_OK local network mask returned<br>> E_NOT_OK local network mask access failed |
| **Functional Description** | |
| This function returns the current network mask for a given controller. | |
| **Particularities and Limitations** | |
| This API is deprecated and will be removed in future revisions of this module. | |
| Call context | |
| > task level | |

Table 4-70 TcpIp_GetLocNetMask

### 4.2.54 TcpIp_GetAndResetMeasurementData

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetAndResetMeasurementData`** `(TcpIp_MeasurementIdxType MeasurementIdx, boolean MeasurementResetNeeded, uint32* MeasurementDataPtr)` | |
| **Parameter** | |
| `MeasurementIdx` [in] | Data index of measurement data |
| `MeasurementResetNeeded` [in] | Flag to trigger a reset of the measurement data |
| `MeasurementDataPtr` [out] | Reference to data buffer, where to copy measurement data |
| **Return code** | |
| Std_ReturnType | > E_OK successful<br>> E_NOT_OK failed |
| **Functional Description** | |
| This function allows to read and reset detailed measurement data for diagnostic purpose. It allows to read the number of dropped TCP/UDP packets, because those packets could not be mapped to valid destination (Local) Ip and Port. Additionally, the function allows to read the number of TCP/UDP packets, dropped due to Invalid header or Invalid checksum. It allows to read the number of IPV4/IPV6 packet, dropped due to invalid destination Ip address. Additionally, it also allows to read the number of IPV6 packets dropped, due to Incorrect packet format, Invalid extension header and Invalid codes and types for ICMPv6 packets. This function also allows to read the number of dropped TCP packets due to insufficient Tx/Rx buffer and the number of dropped IPV4 packets due to Insufficient reassembly buffer.<br>If IPsec support is enabled, the API can be invoked to read the statuses of all the SA entry pairs. The vendor specific indices, TCPIP_MEAS_VENDOR_SPECIFIC_NR_SA_PAIRS and TCPIP_MEAS_VENDOR_SPECIFIC_SA_PAIRS can be used to read the number of SA entries and the statuses of SA entries present in the SAD, respectively. To read the IPsec statuses, the API MUST be called first with TCPIP_MEAS_VENDOR_SPECIFIC_NR_SA_PAIRS, and the second call with TCPIP_MEAS_VENDOR_SPECIFIC_SA_PAIRS and enough size of `MeasurementDataPtr` buffer.<br>Get all Measurement Idxs at once is not supported. TCPIP_MEAS_ALL shall only be used to reset all Measurement Idxs (expect IPsec statuses) at once. Null pointer MUST be provided for `MeasurementDataPtr` in this case. Additionally, if API is called with parameter `MeasurementResetNeeded` set to TRUE and `MeasurementIdx` set to either TCP, UDP, IPV4 or IPV6 (but not TCPIP_MEAS_ALL), the respected measurement data will be first copied and then cleared. `MeasurementResetNeeded` parameter is ignored when called for IPsec status indices. | |
| **Particularities and Limitations** | |
| This API is only present, if configuration parameter TcpIp/TcpIpGeneral/TcpIpGetAndResetMeasurementDataApi == TRUE | |
| **Call context** | |
| > task level | |

Table 4-71 TcpIp_GetAndResetMeasurementData

**Caution**
To invoke TcpIp_GetAndResetMeasurementData() for TCPIP_MEAS_VENDOR_SPECIFIC_SA_PAIRS, the `MeasurementDataPtr` buffer MUST have enough space to store the statuses of all valid SA entries. The exact size of the buffer is calculated as :

*Size = NumOfSaEntryPair * (sizeof(TcpIp_IpSecSaStatusType))*

where *NumOfSaEntryPair* is the value returned by reading TCPIP_MEAS_VENDOR_SPECIFIC_NR_SA_PAIRS

### 4.2.55 TcpIp_IpSecSaEntryPairAdd

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_IpSecSaEntryPairAdd`** `(`<br>`TcpIp_LocalAddrIdType LocalAddrId,`<br>`TCPIP_P2C(TcpIp_IpSecSaInfoType) SaInfoPtr,`<br>`boolean IsInitiator)` | |
| **Parameter** | |
| LocalAddrId [in] | Local Address Id for which SA entry pair is created. |
| SaInfoPtr [in] | Pointer to struct of SA specific information.<br>RemoteAddr field is expected to be in Nbo (inside SaSel).<br>All port numbers are expected to be in Hbo (inside SaSel).<br>The Inbound SPI must be bigger than 0 |
| IsInitiator | Flag to indicate if the calling vIKE module is initiator<br>TRUE: IKE is the initiator<br>FALSE: IKE is the responder<br>For more information look at chapter 5.10.4 |
| **Return code** | |
| Std_ReturnType | > E_OK SA Entry Pair has been added successfully.<br>> E_NOT_OK SA Entry Pair could not be added successfully. |
| **Functional Description** | |
| This function Adds a pair of SA Entries for the controller referred by the Local Address Id. This API is invoked only by the IKE module during the key exchange. | |
| **Particularities and Limitations** | |
| IPsec is configured for the Controller | |
| Call context | |
| > task level | |

Table 4-72 TcpIp_IpSecSaEntryPairAdd

### 4.2.56 TcpIp_IpSecSaEntryPairDelete

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_IpSecSaEntryPairDelete`**`(uint32 InboundSpi)` | |
| **Parameter** | |
| InboundSpi [in] | SPI of the inbound SA entry. |
| **Return code** | |
| Std_ReturnType | > E_OK SA Entry Pair has been deleted successfully.<br>> E_NOT_OK SA Entry Pair could not be deleted. |
| **Functional Description** | |
| This function deletes a pair of SA Entries for the controller. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-73 TcpIp_IpSecSaEntryPairDelete

### 4.2.57 TcpIp_IpSecSaEntryPairPrepareDelete

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_IpSecSaEntryPairPrepareDelete`** `(uint32 InboundSpi)` | |
| **Parameter** | |
| InboundSpi [in] | SPI of the inbound SA entry. |
| **Return code** | |
| Std_ReturnType | > E_OK SA Entry Pair has been successfully prepared for deletion.<br>> E_NOT_OK SA Entry Pair could not be prepared for deletion. |
| **Functional Description** | |
| This function prepares a SA entry pair for deletion. Afterwards SA entries will only be used for reception of messages.<br>SA entries which are prepared for deletion will not be reactivated as described in chapter 5.10.4 IPsec SA Entry Tx Activation Timeout. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-74 TcpIp_IpSecSaEntryPairPrepareDelete

### 4.2.58 TcpIp_GetAvailableTxBufferSize

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_GetAvailableTxBufferSize`**`(TcpIp_SocketIdType SocketId, uint32* AvailableTxBufferSizePtr)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource. |
| AvailableTxBufferSizePtr [out] | Returns the free space in the tx buffer. Is only set if SocketId is OK. |
| **Return code** | |
| Std_ReturnType | > E_OK The available size could be read successfully.<br>> E_NOT_OK Reading was not successful. |
| **Functional Description** | |
| Returns the available size of the TCP or TLS tx buffer (= buffer size - fill level). If TLS is enabled for this socket, the size of the TLS tx buffer is checked, otherwise the size of the TCP buffer. | |
| **Particularities and Limitations** | |
| | |
| Call context | |
| > task level | |

Table 4-75 TcpIp_GetAvailableTxBufferSize

### 4.2.59 TcpIp_Tls_GetRootCertificateId

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`TcpIp_Tls_GetRootCertificateId`**`(TcpIp_SocketIdType SocketId, uint16* CertIdPtr)` | |
| **Parameter** | |
| SocketId [in] | Socket identifier of the related local socket resource. |
| CertIdPtr [out] | Pointer to return the root certificate ID. |
| **Return code** | |
| Std_ReturnType | > E_OK A root certificate was found. Returned value in CertIdPtr is valid.<br>> E_NOT_OK No root certificate was found. Returned value in CertIdPtr is invalid. |
| **Functional Description** | |
| This API is only invoked for TLS Client connections. It returns the ID of the root certificate that belongs to the received certificate chain. | |
| **Particularities and Limitations** | |
| Mode = TCPIP_TLS_CLIENT | |
| Call context | |
| > task level | |

Table 4-76 TcpIp_Tls_GetRootCertificateId

### 4.2.60 TcpIp_Tls_ServiceChainCertificate

| Prototype | |
|---|---|
| `Std_ReturnType` **`TcpIp_Tls_ServiceChainCertificate`**`(`<br>`TcpIp_TlsConnectionIterType TlsConIdx,`<br>`TCPIP_P2C(uint8) CertNamePtr,`<br>`uint32 CertNameLength,`<br>`TCPIP_P2C(TcpIp_CertDataType) CertDataPtr)` | |
| **Parameter** | |
| TlsConIdx [in] | TLS connection index. |
| CertNamePtr [in] | Name of the certificate which should be loaded. |
| CertNameLength [in] | Length of the certificate name. |
| CertDataPtr [in] | Certificate data which should be loaded into the KeyM. |
| **Return code** | |
| Std_ReturnType | > E_OK Certificate could be found and loaded.<br>> E_NOT_OK Certificate could not be found or loaded. |
| **Functional Description** | |
| This API is invoked for TLS Server connections with Server certificates and for TLS Client connections with Client certificates.<br>This API identifies the KeyM Certificate Slot by the given certificate name and loads the given certificate data into the corresponding slot. | |
| **Particularities and Limitations** | |
| MUST be called before the corresponding TLS connection is used for communication. | |
| Call context | |
| > task level | |

Table 4-77 TcpIp_Tls_ServiceChainCertificate

## 4.3 Services used by TCPIP

In the following table services provided by other components, which are used by the TCPIP are listed. For details about prototype and functionality refer to the documentation of the providing component.

| Component | API |
|---|---|
| DET | Det_ReportError |
| EthIf | EthIf_ProvideTxBuffer |
| EthIf | EthIf_Transmit |
| EthIf | EthIf_GetPhysAddr |
| IpBase | IpBase_Copy |
| IpBase | IpBase_Fill |

| Component | API |
|---|---|
| IpBase | IpBase_TcpIpChecksumAdd |
| IpBase | IpBase_DelSockAddr |
| IpBase | IpBase_CopySockAddr |
| IpBase | IpBase_SockPortIsEqual |

Table 4-52 Services used by the TCPIP

## 4.4 Callback Functions

This chapter describes the callback functions that are implemented by the TCPIP and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `TcpIp_Cbk.h` by the TCPIP.

### 4.4.1 TcpIp_RxIndication

| Prototype | |
|---|---|
| `void **TcpIp_RxIndication** (uint8 CtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, uint8 *PhysAddrPtr, uint8 *DataPtr, uint16 LenByte)` | |
| **Parameter** | |
| CtrlIdx [in] | Index of the EthIf controller. |
| FrameType [in] | frame type of received Ethernet frame |
| IsBroadcast [in] | parameter to indicate a broadcast frame |
| PhysAddrPtr [in] | pointer to Physical source address (MAC address in network byte order) of received Ethernet frame |
| DataPtr [in] | Pointer to payload of the received Ethernet frame (i.e. Ethernet header is not provided). |
| LenByte [in] | Length of received data. |
| **Return code** | |
| void | void |
| **Functional Description** | |
| RxIndication from the EthIf. By this API service the TCP/IP stack gets an indication and the data of a received frame. | |
| **Particularities and Limitations** | |
| This function will simply forward the incoming RxIndication to the corresponding RxIndications in the modules IpV4 and IpV6. | |
| Call context | |
| > interrupt or task level | |

Table 4-78 TcpIp_RxIndication

## 4.5 Configurable Interfaces

### 4.5.1 Notifications and Callouts

At its configurable interfaces the TCPIP defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the TCPIP but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

#### 4.5.1.1 <Up_IpV6MaxPayloadLenChanged>

| **Prototype** | |
|---|---|
| `void` **`<Up_IpV6MaxPayloadLenChanged>`**`(uint8 CtrlIdx, const IpBase_SockAddrType *DstAddrPtr, uint16 Mtu)` | |
| **Parameter** | |
| `CtrlIdx` | EthIf controller index |
| `DstAddrPtr` | IPv6 address of the destination |
| `Mtu` | Maximum Transmission Unit (MTU) for the destination. This is the maximum payload length of the IPv6 packet in bytes. If `IpV6_ProvideTxBuffer()` is called with a `BufLen` larger than `Mtu` the packet will be automatically fragmented by the IpV6 if possible. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| These callbacks will be called if an ICMPv6 Packet Too Big (Type 2) message has been received by the IpV6 and the message indicates that the MTU value is smaller than the currently used MTU value for a specific destination address. | |
| **Particularities and Limitations** | |
| > These callbacks will be called only if the PathMTU feature is enabled in the IpV6 configuration.<br>> The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV6Config/TcpIpIcmpV6Config/TcpIpIcmpV6MsgHandler/TcpIpIcmpV6MaxPayloadLenChgHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-79 Maximum Payload Length Change Callback

### 4.5.1.2 <Up_InvNdAddrListOptionHandler>

| **Prototype** | |
|---|---|
| `void <Up_InvNdAddrListOptionHandler>(uint8 CtrlIdx, Eth_PhysAddrType *RemoteLLAddrPtr, IpV6_AddrType *AddrListPtr, uint8 AddrCount)` | |
| **Parameter** | |
| `CtrlIdx` | EthIf controller index |
| `RemoteLLAddrPtr` | Link-layer (physical) address of the remote node that send the address list. |
| `AddrListPtr` | List of IPv6 addresses of the remote node |
| `AddrCount` | Number of IPv6 addresses in the address list |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| These Callbacks will be called when an Inverse Neighbor Discovery Solicitation or Advertisement with an Address List Option has been received (see [RFC4443 3.2. Packet Too Big Message). | |
| **Particularities and Limitations** | |
| > These callbacks will be called only if the “Inverse Solicitations“ or “Inverse Advertisements” are enabled in the IpV6 configuration.<br>> The callback will be called with an empty address list (`AddrListPtr == NULL_PTR && AddrCount == 0`) if there was no response to a user invoked Inverse Solicitation (see `IpV6_Ndp_SendInverseSolicitation()`).<br>> The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV6Config/TcpIpIcmpV6Config/TcpIpIcmpV6MsgHandler/TcpIpIcmpV6IndAddrListReceivedHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-80 Inverse Neighbor Discovery Address List Option Receive Callback

### 4.5.1.3 <Up_IcmpMsgHandler>

| **Prototype** | |
|---|---|
| `void <Up_IcmpMsgHandler>(TcpIp_LocalAddrIdType LocalAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 Ttl, uint8 Type, uint8 Code, uint16 DataLength, uint8 *DataPtr)` | |
| **Parameter** | |
| `LocalAddrId [in]` | Local address identifier representing the local IP address and EthIf controller where the ICMP message has been received |
| `RemoteAddrPtr [in]` | Pointer to struct representing the address of the ICMP sender. |
| `Ttl [in]` | Time to live value of the received ICMPv4 message. |
| `Type [in]` | Type field value of the received ICMP message<br>(Note: the value of the type field determines the format of the remaining ICMP message data) |
| `Code [in]` | Code field value of the received ICMP message. |
| `DataLength [in]` | Length of ICMP message. |
| `DataPtr [in]` | Pointer to the received ICMP message. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| By this API service the configured ICMP message handler function is called by the TCP/IP stack on reception of an ICMP message which is not handled by the TCP/IP stack. | |
| **Particularities and Limitations** | |
| > The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpIcmpConfig/TcpIpIcmpMsgHandler/TcpIpIcmpMsgHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-81 ICMP Destination Unreachable Message Callback

### 4.5.1.4 <Up_IcmpV6MsgHandler>

| **Prototype** | |
|---|---|
| `void <Up_IcmpV6MsgHandler>(TcpIp_LocalAddrIdType LocalAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 Ttl, uint8 Type, uint8 Code, uint16 DataLength, uint8 *DataPtr, uint16 MultiPartDataLength, uint8 *MultiPartDataPtr)` | |
| **Parameter** | |
| `LocalAddrId [in]` | Local address identifier representing the local IP address and EthIf controller where the ICMPv6 message has been received |
| `RemoteAddrPtr [in]` | Pointer to struct representing the address of the ICMPv6 sender. |
| `Ttl [in]` | IP Hop Limit value of the received ICMPv6 message. |
| `Type [in]` | Type field value of the received ICMPv6 message<br>(Note: the value of the type field determines the format of the remaining ICMPv6 message data) |
| `Code [in]` | Code field value of the received ICMPv6 message. |
| `DataLength [in]` | Length of ICMPv6 message. |
| `DataPtr [in]` | Pointer to the received ICMPv6 message. |
| `MultiPartDataLength [in]` | Length of multi-part data in bytes. |
| `MultiPartDataPtr [in]` | ICMPv6 multi-part data (see [RFC4884 8. ICMP Extension Objects]). |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| By this API service the configured ICMPv6 message handler function is called by the TCP/IP stack on reception of an ICMPv6 message which is not handled by the TCP/IP stack.<br><br>This implementation supports ICMPv6 multi-part messages according to IETF RFC 4884.<br>If a message contains multi-part data MultiPartDataLength is > 0 and MultiPartDataPtr points to the multi-part data structure.<br>The following ICMPv6 messages may contain multi-part data:<br>- ICMPv6 Destination Unreachable (Type 1) (see [30] 3.1. Destination Unreachable Message).<br>- ICMPv6 Time Exceeded message has been received (see [30] 3.3. Time Exceeded Message).<br><br>If the message has the ICMPv6 Multi-Part message format `MultiPartExtPtr` points to the first ICMP Extension Object (see [35] 8. ICMP Extension Objects). | |
| **Particularities and Limitations** | |
| > `MultiPartExtLen` is the length of all extensions objects in bytes and may be 0.<br>> The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV6Config/TcpIpIcmpV6Config/TcpIpIcmpV6MsgHandler/TcpIpIcmpV6MsgHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-82 ICMPV6 Destination Unreachable Message Callback

### 4.5.1.5 <Up_LinkLocalAddrCandidateCallout>

| **Prototype** | |
|---|---|
| `void **<Up_LinkLocalAddrCandidateCallout>** (TcpIp_LocalAddrIdType LocalAddrId, uint8 conflictCount, uint32 *addrCandidatePtr)` | |
| **Parameter** | |
| `LocalAddrId` | Identifier of the TcpIpLocalAddr that shall be configured using the LINKLOCAL address assignment method. |
| `conflictCount` | Number of conflicts that occurred since start of the address assignment for the local address. This value is 0 for the first probed address and is incremented every time a conflict has occurred. |
| `addrCandidatePtr` | Output parameter for the IP address candidate (IPv4 address in network-byte order). If this parameter is not changed inside the callout the IPv4 automatically generates a valid random IP address candidate. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| This callout is called when a local unicast address shall be configured using the LINKLOCAL (Auto-IP) assignment method. The function is called every time a link-local address candidate is required.<br>The callout is not required to modify *addrCandidatePtr if a random address according to IETF RFC 3927 shall be used by the IPv4. | |
| **Particularities and Limitations** | |
| > The returned address candidate must be in the range 169.254.1.0 to 169.254.254.255 (inclusive) in order to conform to IETF RFC 3927. The value is not checked by the IPv4.<br>> The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpAutoIpConfig/TcpIpAutoIpAddrCandidateCalloutFunction` | |
| Call context | |
| > interrupt or task context | |

Table 4-83 IPv4 Link local Auto Ip address assignment Callback

### 4.5.1.6 <Up_PhysAddrTableChg>

| Prototype | |
|---|---|
| `void` **`<Up_PhysAddrTableChg>`** `(uint8 CtrlIdx, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr, TCPIP_P2V(uint8) PhysAddrPtr, boolean Valid);` | |
| **Parameter** | |
| `uint8 CtrlIdx` | EthIf controller index to identify the related ARP/NDP table. |
| `IpAddrPtr` | Specifies the memory where the IP address is stored for which the assignment to a physical address has changed. |
| `PhysAddrPtr` | Specifies the memory where the physical source address (MAC address) in network byte order is stored. |
| `Valid` | Flag, whether the couple of Ip address and physical address is valid or not. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| This callback is called in order to notify upper layers about a physical address change. | |
| **Particularities and Limitations** | |
| > The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpPhysAddrConfig/TcpIpPhysAddrChgHandler/TcpIpPhysAddrChgHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-84 Physical Address Change Callback

### 4.5.1.7 <Up_PhysAddrTableEntryDiscarded>

| Prototype | |
|---|---|
| `void <Up_PhysAddrTableEntryDiscarded> (uint8 CtrlIdx, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr);` | |
| **Parameter** | |
| `uint8 CtrlIdx` | EthIf controller index to identify the related ARP/NDP table. |
| `IpAddrPtr` | Specifies the memory where the IP address is stored for which the assignment to a physical address has changed. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callback is called in order to notify upper layers about the fact that a Ip address change has been discarded because EcuC switch TcpIpArpDiscardedEntryHandling is true. Refer to section 2.9.3.1Tracking discarded / overwritten ARP entries for details. | |
| **Particularities and Limitations** | |
| > The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpPhysAddrConfig/TcpIpPhysAddrChgDiscardedHandler/TcpIpPhysAddrChgDiscardedHandlerName` | |
| Call context | |
| > interrupt or task context | |

Table 4-85 Physical Address Change Discarded Callback

### 4.5.1.8 <Up_DADAddressConflict>

| **Prototype** | |
|---|---|
| `void **<Up_DADAddressConflict>** (TcpIp_LocalAddrIdType IpAddrId,const TcpIp_SockAddrType* IpAddrPtr, const uint8* LocalPhysAddrPtr, const uint8* RemotePhysAddrPtr);` | |
| **Parameter** | |
| IpAddrId | local address identifier |
| IpAddrPtr | Specifies the memory where the IP address is stored for which a duplicate address has been detected. |
| LocalPhysAddrPtr | Specifies the memory where the local physical source address (MAC address) in network byte order is stored. |
| RemotePhysAddrPtr | Specifies the memory where the remote node’s physical address (MAC address) in network byte order is stored. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callback is called in order to notify upper layers about the detection of a duplicate IP address. | |
| **Particularities and Limitations** | |
| > This callout is currently issued only in case duplicate address detection via NDP is done when IPv6 is active.<br>> The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpDuplicateAddressDetectionConfig/TcpIpDuplicateAddressDetectionCalloutName` | |
| Call context | |
| > interrupt or task context | |

Table 4-86 Duplicate Address Detection Callback

#### 4.5.1.9 <Up_DhcpRequestedIpAddrCallout>

| Prototype | |
|---|---|
| `void` **`<Up_DhcpRequestedIpAddrCallout>`** `(TcpIp_LocalAddrIdType LocalAddrId, IpBase_AddrInType *RequestedAddrPtr)` | |
| **Parameter** | |
| `LocalAddrId` | Identifier of the TcpIpLocalAddr that shall be configured using the DHCP address assignment method. |
| `RequestedAddrPtr` | Input/Output parameter for the IP address that shall be requested in the DHCPDISCOVER message via the ‘Requested IP Address’ option (IPv4 address in network-byte order).<br>If this parameter is not changed or set to TCPIP_INADDR_ANY inside the callout the DHCP will not request a particular IP address in the DHCPDISCOVER messages. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callout is called after the DHCP address assignment has been started but before the first DISCOVER message is sent by the DHCPv4 client.<br>If an IP address is returned this address will be requested using the ‘Requested IP Address’ option in each DHCPDISCOVER message.<br>If the DHCP server offers a different IP address the DHCP client will request the offered address and not the address requested in the DHCPDISCOVER message.<br>According to IETF RFC 2131 “3.5 Client parameters in DHCP“ the requested IP address in the DHCPDISCOVER message is only a suggestion of a particular IP address. | |
| **Particularities and Limitations** | |
| > The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpDhcpConfig/TcpIpDhcpV4RequestedAddrCalloutFunction` | |
| Call context | |
| > interrupt or task context | |

Table 4-87 DHCPv4 Requested IP address

### 4.5.1.10 <Up_DhcpEvent>

| **Prototype** | |
|---|---|
| `void <Up_DhcpEvent> ( TcpIp_LocalAddrIdType LocalIpAddrId, TcpIp_DhcpEventType Event)` | |
| **Parameter** | |
| `LocalIpAddrId` | Identifier of the local IP address which received or which will transmit a DHCP message. |
| `Event` | Indicates the actual DHCP event. See “4.1 Type Definitions” for further details. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callout is only relevant for DHCPv4/DHCPv6 client functionality.<br>The callout is invoked for the following events:<br>> After reception of DHCPv4/DHCPv6 client message.<br>> Prior transmission of DHCPv4/DHCPv6 client message.<br>The callout can be used to synchronize the handling of DHCP options via the read and write APIs:<br>• TcpIp_DhcpWriteOption<br>• TcpIp_DhcpReadOption<br>• TcpIp_DhcpV6WriteOption<br>• TcpIp_DhcpV6ReadOption | |
| **Particularities and Limitations** | |
| > The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerDhcpEventName` | |
| Call context | |
| > interrupt or task context | |

Table 4-88 DHCP Event Callout

### 4.5.1.11 <Up_IpSecSpdCalloutFunction>

| **Prototype** | |
|---|---|
| `void **<Up_IpSecSpdCalloutFunction>** (`<br>`TCPIP_P2V(TcpIp_IpSecPolicyType)SecurityPolicyPtr,`<br>`uint8 Protocol,`<br>`TcpIp_SockAddrType RemSock,`<br>`TcpIp_SockAddrType LocSock)` | |
| **Parameter** | |
| `SecurityPolicyPtr` | Input/Output parameter for the Security policy which has been selected for the current IP packet processed (Tx or Rx). The values possible are PROTECT, BYPASS or OPTIONAL.<br>This parameter can be overwritten by the caller. The caller is allowed to overwrite with only the following values BYPASS or DISCARD. |
| `Protocol` | The upper layer protocol in the IP packet. |
| `RemSock` | Remote IP address and port. |
| `LocSock` | Local IP address and port. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callout is called for each IP packet processed by the controller (Tx and Rx), after the Security Policy for the traffic has been looked up.<br>The caller has the option of overwriting the security policy for the packet to either BYPASS in case they which to process the packet or DISCARD to drop the packet without further processing. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter:<br>`TcpIp/TcpIpConfig/TcpIpIpSecConfigSet/TcpIpIpSecSpdCalloutFunction` | |
| Call context | |
| interrupt or task context | |

Table 4-89 IPsec Security Policy Callout

### 4.5.1.12 <Up_IpSecAuditEventCalloutFunction>

| Prototype | |
|---|---|
| `void <Up_ IpSecAuditEventCalloutFunction> (`<br>`uint32 Spi,`<br>`uint32 SeqNum,`<br>`TcpIp_SockAddrType RemSock,`<br>`TcpIp_SockAddrType LocSock,`<br>`TcpIp_IpSecEventType ErrorId)` | |
| **Parameter** | |
| `Spi` | SPI index of the audited packet. |
| `SeqNum` | Sequence Number of the audited packet. |
| `RemSock` | Remote IP address and port. |
| `LocSock` | Local IP address and port. |
| `ErrorId` | Auditable Error that is reported. |
| **Return code** | |
| `Void` | - |
| **Functional Description** | |
| This callout is invoked whenever an auditable event reported for the IPsec, to notify the upper layer. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter:<br>`TcpIp/TcpIpConfig/TcpIpIpSecConfigSet/TcpIpIpSecAuditEventCalloutFunction` | |
| Call context | |
| interrupt or task context | |

Table 4-90 IPsec Audit Event Callout

### 4.5.1.13 <Up_TlsValidationResultCallout>

| Prototype | |
|---|---|
| `void <Up_ TlsValidationResultCallout> (`<br>`TcpIp_SocketIdType SocketId,`<br>`uint32 NumCerts,`<br>`TCPIP_P2C(TcpIp_CertValidationStatusType) CertChainStatusPtr,`<br>`TCPIP_P2V(TcpIp_CertValidationResultType) ValidationResultPtr);` | |
| **Parameter** | |
| `SocketId` | Socket index for which the callout is triggered. |
| `NumCerts` | Number of certificates for which the status is received |
| `CertChainStatusPtr` | Struct which contains the certificate validation information. |
| `ValidationResultPtr` | Overall result of the certificate validation. This Pointer is used by the UpperLayer to overrule the status of the validation. |

| Return code | |
|---|---|
| `Void` | - |
| **Functional Description** | |
| This callout is invoked during the TLS handshake to inform the UpperLayer about the status of the received certificate chain. The user can decide if the TLS handshake is accepted or should be canceled by overruling the validation result. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter: `/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerTlsValidationResult` | |
| Call context | |
| task context | |

Table 4-91 Tls Validation Result Callout

#### 4.5.1.14 <Up_TlsGetCurrentTimeStamp>

| Prototype | |
|---|---|
| `Std_ReturnType` **`<Up_ TlsGetCurrentTimeStamp>`** `( TCPIP_CP2V(uint32) GmtUnixTime);` | |
| **Parameter** | |
| `GmtUnixTime` | A pointer to provide the 4 byte timestamp. |
| **Return code** | |
| `Std_ReturnType` | Return E_OK if the function call was successful otherwise return E_NOT_OK. |
| **Functional Description** | |
| This callout is invoked during the TLS handshake at that time the random number for the ClientHello or the ServerHello message is calculated and a callout function is configured. If no callout function is configured the 4 byte timestamp is filled with 0. The timestamp is also filled with 0 in the case the configured function returns E_NOT_OK. Additionally, a user error is reported. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter: `TcpIp/TcpIpConfig/TcpIpTlsConfig/TcpIpTlsConnection/TcpIpTlsConnectionGetTimeFunc` | |
| Call context | |
| task context | |

Table 4-92 Tls Get Current Time Stamp

#### 4.5.1.15 TcpIp_Tls_ErrorIndicationCalloutFunction

| Prototype | |
|---|---|
| `void < **TcpIp_Tls_ErrorIndicationCalloutFunction**> ( TcpIp_SocketIdType SocketId, TcpIp_Tls_ErrorIndicationType ErrorIndication, TcpIp_TlsAlertDescriptionType AlertDescription);` | |
| **Parameter** | |
| `SocketId` | Socket index for which the callout is triggered. |
| `ErrorIndication` | Error indication, which informs the user about the raised error. |
| `AlertDescription` | Set to the received/transmitted TLS-Alert.<br>CONSTRAINT: [only valid if ErrorIndication is TCPIP_TLS_E_RX_ALERT or TCPIP_TLS_E_TX_ALERT]. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| Function which will be called to inform the user (upper layer) over raised TLS error indications. The API is invoked during the TLS handshake or TLS communication. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter: `/TcpIp/TcpIpConfig/TcpIpTlsConfig/TcpIpTlsSupportErrorIndicationCallout` | |
| Call context | |
| task context | |

Table 4-93 TcpIp_Tls_ErrorIndicationCalloutFunction

#### 4.5.1.16 <Up_TlsCertRequestCallout>

| Prototype | |
|---|---|
| `Std_ReturnType **<Up_ TlsCertRequestCallout>** ( TcpIp_SocketIdType SocketId, TCPIP_P2C(TcpIp_TlsCertRequestInfoType) RequestInfoPtr, TCPIP_P2V(TcpIp_TlsClientCertInfoType) ClientCertInfoPtr, TCPIP_P2V(TcpIp_TlsClientCertInfoStatusType) ClientCertInfoStatusPtr);` | |
| **Parameter** | |
| `SocketId` | Socket index for which the callout is triggered. |
| `RequestInfoPtr` | Struct which contains information about the received cert request, filled by TLS. |
| `ClientCertInfoPtr` | Struct which contains information about the client certificates that shall be sent by TLS. The number of certificates is an in-out parameter. The struct is filled by the socket owner. |
| `ClientCertInfoStatusPtr` | Status of the callout state. This is filled by the socket owner. |

| Return code | |
|---|---|
| `E_OK` | The callout was accepted by the socket owner |
| `E_NOT_OK` | The callout could not be accepted by the socket owner |
| **Functional Description** | |
| This callout is invoked during the TLS handshake to inform the UpperLayer about the received certificate request. The UpperLayer checks the request information and selects the appropriate client certificate chain. The callout works asynchronously, that means the callout it repeated in every MainFunction until the UpperLayer sets the ClientCertInfoStatus to a value different than PENDING. If the UpperLayer sets the status to NOT_AVAILABLE, the client will send an empty client certificate message (and therefore no certificate verify message). If the UpperLayer sets the status to ERROR, the client will send an alert InternalError and cancel the handshake. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter: `/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerTlsCertRequestName` | |
| Call context | |
| task context | |

Table 4-94 Tls Cert Request Callout

#### 4.5.1.17 <Up_ChangeableCipherCallback>

| Prototype | |
|---|---|
| `Std_ReturnType` **`<Up_ChangeableCipherCallback >`** `( TcpIp_SocketIdType SocketId, TCPIP_P2V(TcpIp_TlsCipherSuiteWorkingSetType) CipherSuiteWorkingSetPtr, TCPIP_P2V(TcpIp_TlsSignatureAlgorithmWorkingSetType) SignatureAlgorithmWorkingSetPtr, TCPIP_P2V(TcpIp_SccPushButtonModeOfTlsClientConnectionType) SccPushButtonModePtr, TCPIP_P2V(TcpIp_TlsChangeCipherStatusType) ChangeCipherStatusPtr);` | |
| **Parameter** | |
| `SocketId` | Socket index for which the callout is triggered. |
| `CipherSuiteWorkingSetPtr` | Pointer to the working set of cipher suites |
| `SignatureAlgorithmWorkingSetPtr` | Pointer to the working set of signature algorithms |
| `SccPushButtonModePtr` | Flag to indicate PushButtonMode from SCC |
| `ChangeCipherStatusPtr` | Status of the current operation |
| **Return code** | |
| `E_OK` | The callout was accepted by the socket owner |
| `E_NOT_OK` | The callout could not be accepted by the socket owner |

| Functional Description |
|---|
| This callout is invoked at the beginning of the TLS handshake to inform the UpperLayer about the supported cipher suites and signature algorithms. The UL is able to change the working sets of the cipher suites and/or signature algorithms. This callback also allows the user to activate the PushButtonMode (see 2.1.4.1.8). |
| **Particularities and Limitations** |
| The name of this function is specified by the following configuration parameter: `/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerTlsChangeableCipherName` |
| Call context |
| task context |

Table 4-95 Tls Changeable Cipher Callback

### 4.5.1.18 <Up_SessionTicketUpdateCallout>

| Prototype | |
|---|---|
| `void` **`<Up_SessionTicketUpdateCallout >`** `( TcpIp_SocketIdType SocketId,` `TCPIP_P2C(TcpIp_TlsSessionTicketUserInfoType) SessionTicketInfoPtr);` | |
| **Parameter** | |
| `SocketId` | Socket index for which the callout is triggered. |
| `SessionTicketInfoPtr` | Pointer to session ticket info struct. Currently this contains only the amount of currently available tickets (see Type Definitions). This pointer is only valid during the context of this callout. |
| **Return code** | |
| `void` | - |
| **Functional Description** | |
| This callout is invoked every time the amount of available tickets changes:<br>• A new ticket is available when a valid NewSessionTicket message is received, and a fitting storage option is still available.<br>• The number of available tickets decreases when a ticket's lifetime expires.<br>• The resumption base connection is closed and therefore related available tickets are deleted.<br>• A ticket is set to be used to resume a connection which is triggered via TcpIp_ChangeParameter. | |
| **Particularities and Limitations** | |
| The name of this function is specified by the following configuration parameter: `/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerTlsSessionTicketUpdateName` | |
| Call context | |
| task context | |

Table 4-96 Tls Session Ticket Update Callout

# 5 Configuration

The TCPIP is configured with the help of the configuration tool DaVinci Configurator Pro. The description of the parameters can be found in the BSWMD-file or in the properties window in the tool.

The following sections explain some general topics regarding the configuration.

## 5.1 Configuration Variants

The TCPIP supports the configuration variants

> `VARIANT-PRE-COMPILE`

The TCPIP supports the configuration of multiple variants (post-build variant support).

The configuration classes of the TCPIP parameters depend on the supported configuration variants. For their definitions please see the TcpIp_bswmd.arxml file.

## 5.2 TcpIp General Configuration

### 5.2.1 Random Number Function

The parameter `TcpIp/TcpIpGeneral/TcpIpRandNoFct` defines the function called for the random number generation. For several use cases the TcpIp needs to generate random numbers. For this generation a seed is needed. The randomness of a random number generator mainly depends on the randomness of the seed. To give the user control over the quality of the random value generation, this function is needed as an input for all further random number generations done by the TcpIp and all configured sub features as e.g., Ipv4, Ipv6, TLS… Random values are used e.g., for Cryptography, to generate Session IDs, for waiting times and other values.

Directly related to the parameter `TcpIp/TcpIpGeneral/TcpIpRandNoFct` is also the parameter `TcpIp/TcpIpGeneral/TcpIpRandNoFctIncludeFile`. This parameter defines the header file holding the declaration of the random number generation function defined in '`TcpIpRandNoFct`'.

Both parameters are mandatory and must be specified by the user.

## 5.3 Socket Owner Configuration

TCPIP allows multiple users above the TcpIp, and AUTOSAR calls them ‘socket owner’. The SocketAdaptor (SoAd) is the standard socket owner, other socket owners can be added easily as CDD (complex device driver). Vector furthermore adds a special socket owner ‘DhcpV4Server’ that is only used TCPIP-internal if the DhcpV4Server is enabled.

For each socket owner several callbacks have to be configured.

| Callback Name | Required |
|---|---|
| <Up>_RxIndication | Mandatory |
| <Up>_TcpIpEvent | Mandatory |
| <Up>_TxConfirmation | Only required if UDP TX confirmation is requested. |

| | |
|---|---|
| <Up>_TcpAccepted | Only required TCP is used. |
| <Up>_TcpConnected | Only required TCP is used. |
| <Up>_CopyTxData | Only required if indirect data provision is used. (see TcpIp_UdpTransmit and TcpIp_TcpTransmit) |
| <Up>_LocalIpAddrAssignmentChg | Optional |
| <Up>_DhcpEvent | Optional |

Table 5-1 Callbacks of TcpIp Socket Owners

**Note**
The prefix “<Up>” is replaced by the name of the TcpIp socket owner.
The API signature of the callbacks is specified by [6] Specification of Socket Adaptor.

If the socket owner shall act as a server, a special configuration is needed. On the one hand buffers need to be configured for the number of connections the socket owner wants to handle in parallel. On the other hand we need to configure the parameter ‘Socket Owner Tcp Listen Socket Max’. This value specifies the number of TCP listen sockets this socket owner will use simultaneously. These pure listen sockets will have no rx or tx buffers. So if a socket owner wants to offer a service using one server socket and accept two parallel connections to its offered service, the parameter ‘Socket Owner Tcp Listen Socket Max’ has to be set to ‘1’ and 2 pairs of rx and tx buffers have to be configured.

### 5.3.1 <Up>_CopyTxData Callback

If a TcpIpSocketOwner uses one of the APIs TcpIp_UdpTransmit() or TcpIp_TcpTransmit() with indirect data provision (parameter DataPtr == NULL_PTR) the CopyTxData-Callback must be provided by the socket owner.

AUTOSAR specifies that CopyTxData is called one or more times until the socket owner has provided the specified amount of bytes during the Transmit call.

Vector optionally supports an extension of this API using an in/out parameter BufLengthPtr instead of the in-only parameter BufLength. In this case the socket owner may update the value at BufLengthPtr with a lower value in order to indicate that it wants to transmit less data than specified by the Transmit call. Once a socket owner does not use the entire provided buffer in the CopyTxData call, the TcpIp will not request more data, transmit only the provided amount of bytes and release the unused buffer.

**Configuration in DaVinci Configurator Pro**
The type of the <Up>_CopyTxData callback can be configured individually for each TcpIpSocketOwner using the configuration parameter **TcpIpSocketOwnerCopyTxDataDynamicLengthEnabled**.

## 5.4 Unicast Address Assignment Methods

The configuration of multiple unicast address assignment methods differs between IPv4 and IPv6.

The reason is that the IPv6 can handle multiple unicast addresses on one IP controller at the same time while the IPv4 only supports one unicast address per IP controller.

Table 5-2 shows the supported configuration possibilities for IPv4 and IPv6:

| Configuration Possibilities | IPv4 | IPv6 |
|---|---|---|
| Number of unicast addresses per controller/VLAN | 1 | 1…N |
| Number of assignment methods per unicast address | 1…3 | 1 |
| Supported address assignment methods | STATIC, LINKLOCAL, DHCPv4 | STATIC, LINKLOCAL, DHCPv6, ROUTER |
| Unicast address assignments that can be stored into nonvolatile memory | STATIC, DHCPv4 | STATIC |

Table 5-2 Configuration possibilities for IPv4 and IPv6 unicast address assignments

### 5.4.1 Multiple Address Assignment Methods for IPv4 Unicast Addresses

According to AUTOSAR it is possible to configure up to three different IP address assignment methods for the IPv4 unicast address of an IP instance.

Supported address assignment methods are STATIC, LINKLOCAL (AUTO-IP) and DHCP.

The address assignment methods of a unicast address must be configured with different priorities (1…3, where 1 means highest priority). During runtime the value of the active unicast address is always set to the address of the assignment method with the highest priority that is triggered and ready.

Depending on the configuration parameter TcpIpAssignmentTrigger an address assignment method is either triggered automatically after the ethernet link is up or manually during runtime via the API TcpIp_RequestIpAddrAssignment().

After being triggered the two dynamic address assignment methods DHCP and LINKLOCAL need some time to negotiate an address or even may not be able to configure an address (e.g. because there is no DHCP server present in the network). During this time the address assignment methods are not ready and an address of a method with lower priority will be used if possible.

At startup, all configured IP address assignments will be triggered to configure a unique IP address. In case a higher priority assignment gets ready, the lower priority assignments will neither be aborted, nor be used. (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

An address with the STATIC address assignment method can be set in the configuration (see configuration parameter TcpIpStaticIpAddress) or during runtime via TcpIp_RequestIpAddrAssignment().

**AUTOSAR Extension**

In addition to AUTOSAR an address with assignment method STATIC may be changed even if the assignment trigger is configured to AUTOMATIC. In order to use this feature the configuration parameter TcpIpAssignmentLifetime must be set to TCPIP_STORE.

In this case the configured address value is used as a default value which may be persistently overwritten later. It is also possible to restore the default value again by requesting the address value TCPIP_ADDR_ANY. LINKLOCAL addresses cannot be persisted. (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

All manually triggered address assignment can be deactivated again by calling TcpIp_ReleaseIpAddrAssignment().

**AUTOSAR Extension**

The AUTOSAR API TcpIp_ReleaseIpAddrAssignment() only supports releasing all manually triggered address assignment methods of an address at once.

Vector provides the additional API TcpIp_ReleaseSpecificIpAddrAssignment() to release only a specific assignment method of an address.

The TcpIp socket owners are notified about each IP address assignment change via the LocalIpAddrAssignmentChg()-Callback.

The following two APIs may be used in context of the callback in order to obtain detailed information about the unassigned/assigned address:

- TcpIp_GetIpAddr
- TcpIp_GetIpAddrCfgSrc

### 5.4.2 Callout for provision of IPv4 link-local address candidates

The IPv4 link-local address assignment according to IETF RFC 3927 [27] uses a randomly generated address that is most likely unique on the local link. The uniqueness of the address is tested by sending ARP probes for an address candidate before it is assigned as unicast address of the node. If a conflict is detected, another address candidate is not generated and configured immediately but it is configured according to IETF RFC 3927 Conflict Detection and Defense strategy (b). (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

The optional configuration of the callout function <User>_LinkLocalAddrCandidateCallout() allows it to provide address candidates for the link-local address configuration.

Examples of usage scenarios for this callout are:

- Provide link-local address candidates that are derived from likely unique values and therefore reduce the probability of address conflicts.
- Provide stored address candidates that have been tested for uniqueness before and therefore are likely to be still unique on the local-link.

In any case the provided address candidates will be probed for uniqueness again before they are used as source address by the node.

The callout is not required to provide all address candidates. It may also provide only the first candidate. If the callout does not provide an address candidate, the IPv4 will automatically choose a random value according to IETF RFC 3927.

**Configuration in DaVinci Configurator Pro**
The <User>_LinkLocalAddrCandidateCallout is specified by the optional parameter TcpIpAutoIpAddrCandidateCalloutFunction inside the TcpIpAutoIpConfig container.

### 5.4.3 Callout for provision of DHCPv4 address that shall be requested

By default, the DHCPv4 address assignment does not request the assignment of a particular IP address in the DHCPDISCOVER message.

The optional configuration of the callout function <User>_RequestedDhcpAddrCallout() allows it to provide a particular IP address that shall be requested in the DHCPDISCOVER messages using the ‘Requested IP Address’ option.

This is useful if the client shall try to obtain the same IP address again that was assigned by a DHCP server before.

**Configuration in DaVinci Configurator Pro**
The <User>_ RequestedDhcpAddrCallout is specified by the optional parameter TcpIpDhcpV4RequestedAddrCalloutFunction inside the TcpIpDhcpConfig container.

## 5.5 IPv4

### 5.5.1 Fragmentation

IP fragmentation is only active for controllers that reference a `TcpIpIpFragmentationConfig`.

For reception, **Out of Order support** is available by disabling parameter `TcpIpIpReassInOrderEnabled`. This parameter defines whether all fragments of an IP datagram are expected to arrive in order. If enabled, reassembly of an IP datagram is aborted if a fragment arrives out of order so that the buffer is available for reassembly of other datagrams immediately. This is an optimization for local networks where all fragments of a fragmented IP datagram should arrive in order. The parameter `TcpIpIpReassTimeout` defines how long received fragments should stay inside the fragmentation buffer and wait for a complete reception of all missing fragments of the IP packet. The parameter `TcpIpIpNumReassDgrams` defines the maximum number of fragmented IP datagrams that can be reassembled in parallel.

For fragmentation, `TcpIpIpMaxTxDgramSize` specifies the maximum size of outgoing IP datagrams (including all headers). If a datagram does not fit into the ethernet MTU it will be split up into multiple IP fragments.

For the reassembly of IP packets, `TcpIpIpMaxReassDgramSize` specifies maximum size of IPv4 packets that can be reassembled.

### 5.5.2 ICMPv4

If the checkbox `TcpIpIcmpDestinationUnreachableEnabled` is enabled, TcpIp responds with a **Destination Unreachable Message** if a unicast packet with one of the following properties is received.

- The upper layer protocol is unknown.

- The UDP destination port is not open.

Received **ECHO** requests will be answered with **ECHO** replies. If the payload of a received ECHO request is bigger than `TcpIpIcmpEchoReplyMaxLen`, the reply will contain as much data as fit into the buffer and the remaining data will be discarded and not echoed back.

### 5.5.3 DHCPv4 Client

DHCPv4 supports a configurable amount of delay before the first DISCOVER is sent out. It also supports configurable number of DISCOVER (`TcpIpDhcpV4DiscoverMaxNum)` and REQUEST (`TcpIpDhcpV4RequestMaxNum)` messages. That means if the remote DHCP server does not respond with an OFFER, the configured number of DISCOVER messages will be send by the DHCP client before it discontinues the address assignment process. The same applies for REQUEST messages if the server does not respond with an ACK/NACK.

TcpIp supports restart on failure feature. If `TcpIpDhcpV4RestartOnFail` is enabled, TcpIp triggers address assignment process automatically if the previous process could not be succeeded or the lease of previously assigned IP address could not be extended.

Parameter `TcpIpDhcpV4HostNameLenMax` defines the number of bytes of the Host Name that could be set by the user via public API during runtime. The set hostname will be serialized to the transmitted DHCP messages.

#### 5.5.4 ARP

ARP is an address resolution protocol. It is used to resolve the physical address of the neighboring devices. TcpIp updates its ARP cache by reception of ARP replies. Number of ARP entries that can be saved in ARP cache at a time is configured by `TcpIpArpTableSizeMax.` Entries which are saved in ARP cache are valid as long as the timeout `TcpIpArpTableEntryTimeout` has not occurred. Once the entry is expired it will get renewed. TcpIp will try to renew/resolve the entry by sending ARP requests until the `TcpIpArpRetryTime` has not been reached. Once the retry time is expired the entry will be invalidated and the upper layer will be informed that the physical address of the requested device could not be resolved.

TcpIp also supports the feature of `TcpIpArpNumGratuitousARPonStartup` sending number of ARP gratuitous replies once the IP address is assigned on startup. It allows neighboring devices to update their ARP caches on startup. Gratuitous replies are reply packets whose target and seder IP address are identical.

#### 5.5.5 AutoIp

AutoIp allows TcpIp to configure an automatic link local IP address once full communication mode is requested. The IP address that is selected for configuration, is generated using a random number if not provided through the configured callout. The address is probed for its uniqueness before use. If there is no conflicting ARP probe or ARP request received for the probed IP address after sending `TcpIpAutoIpProbeNum` ARP probes this address will be used by the controller. The used IP address will be announced `TcpIpAutoIpAnnounceNum` times on the link local network. If the TcpIp receives conflicting packets for `TcpIpAutoIpMaxConflicts` number of times, then another random address is selected for address assignment process as stated above.

### 5.6 IPv6

#### 5.6.1 Fragmentation

IP fragmentation is only active for controllers that reference a `TcpIpIpV6FragmentationConfig`. Rx and Tx fragmentation can also be deactivated individually by setting `TcpIpIpV6ReassemblyBufferCount` or `TcpIpIpV6TxFragmentBufferCount` to 0.

For reception, **Out of Order support** is available, with a limited number of Out of Order segments, that can be configured by `TcpIpIpV6ReassemblySegmentCount`. If this parameter is set to 1, Out of Order support is deactivated.

The parameter `TcpIpIpV6ReassemblyTimeout` defines how long received fragments stay within the fragmentation buffer and wait for a complete reception of all missing fragments of the IP packet.

For the fragmentation or reassembly of IP packets, at least one buffer with more than 1500 bytes is required. The maximum size of the buffer(s) can be configured by `TcpIpIpV6TxFragmentBufferSize` or `TcpIpIpV6ReassemblyBufferSize`. This is also the maximum size of IPv6 packets that can be processed.

### 5.6.2 ICMPv6

Received **ECHO** requests will be answered if `TcpIpIcmpV6EchoReplyEnabled` is enabled. If the received ECHO payload is bigger than `TcpIpIcmpV6EchoDataBufferSize`, the reply will contain as much data as fit into the buffer and will discard the residual part of the initial ECHO. Own ECHO requests could be sent by using the `IpV6_Icmp_Transmit` API.

Error messages like "**Packet too Big**", "**Time Exceeded**" and "**Parameter Problem**" will be sent if `TcpIpIcmpV6ErrorMessagesEnabled` is enabled.

The TcpIp stack processes following ICMPv6 messages:

- Router Advertisement(134)
- Neighbor Solicitation(135)
- Neighbor Advertisement(136)
- Redirect(137)
- Packet Too Big(2)
  - If TcpIpIpV6PathMtuEnabled == true
- Echo Request(128)
  - If TcpIpIcmpV6EchoReplyEnabled == true
- Inverse Neighbor Discovery Solicitation Message(141)
  - If TcpIpNdpInverseNsEnabled== true
- Inverse Neighbor Discovery Advertisement Message(142)
  - If TcpIpNdpInverseNaEnabled == true

ICMPv6 messages that were not handled by the TcpIp stack (not supported or not configured) will be forwarded to the upper layer via the callout configured in TcpIpIcmpV6MsgHandlerName. If no callout is configured, the message will be dropped without increasing a measurement counter.

### 5.6.3 DHCPv6

DHCPv6 supports a configurable delay, before Solicit, Information Request and Confirm messages are transmitted for the first time. The used time is randomly chosen between the min and max value. To avoid simultaneous transmissions of messages by more than one DHCPv6 client, parameter `TcpIpDhcpV6RandomizeTimeouts` can be additionally activated.

The used transmit buffer `TcpIpDhcpV6TxMsgBufferLen` defines how big a DHCPv6 message could be. It must be big enough to contain all additionally configured `TcpIpDhcpV6UserOption`.

The RFC3315 advices to wait also if a valid Advertise message is received. To disable this behavior, `TcpIpDhcpV6UseFirstValidAdv` can be activated.

Additional DHCPv6 user options can be added.

Compared to DHCPv4, DHCPv6 does not support a restart on failure functionality.

### 5.6.4 NDP

TcpIp NDP does not support **MLD** and **Private Extensions**.

`TcpIpNdpV6DropRedirect` allows to configure if NDP messages of type Redirect (ICMPv6 type 137) should be dropped.

**SLAAC** (Stateless Address Autoconfiguration) can configure an IPv6 address. A configurable number (`TcpIpNdpSlaacDadNumberOfTransmissions`) of DAD messages will be sent to detect a possibly duplicated address. Till this test is successfully completed, the address can only be used if `TcpIpNdpSlaacOptimisticDadEnabled` is activated.

If **Prefix Router Discovery** functionality is used, the size of the internal list of all default routers can be specified by `TcpIpNdpDefaultRouterListSize`. The number of the prefix list, which is used to decide whether an address is on-link or not, can be modified with `TcpIpNdpPrefixListSize` and prefilled with `TcpIpNdpOnLinkPrefix`. Both lists can also be filled by received router advertisements during runtime. Each address could also have a static default router defined with `TcpIpLocalAddr/TcpIpStaticIpAddressConfig/TcpIpDefaultRouter`. The default router list is shared between all addresses of a `TcpIpIpV6Ctrl`. Similarly, the first 64 bits of a static IPv6 address can be added to the prefix list by activating the `TcpIpIpV6AddrPrefixIsOnLink` option. `TcpIpNdpDropRouterAdvertisement` allows to configure if Router Advertisement should be dropped.

The **NUD** (Neighbor Unreachability Detection) shall detect unreachable destination addresses if `TcpIpNdpNeighborUnreachabilityDetectionEnabled` is active.

`TcpIpNdpDefaultReachableTime` defines how long an address is considered as reachable. `TcpIpNdpDefaultRetransTimer` defines how long TcpIp waits till a Neighbor Solicitation shall be retransmitted if no Advertisement for a request is received. Those Neighbor Solicitations will be sent to a unicast address and will be sent `TcpIpNdpNumUnicastSolicitations` times. The first time till NUD shall probe an address is configured with `TcpIpNdpDelayFirstProbeTime`.

The number of Neighbor Solicitations for address resolution can be configured with `TcpIpNdpNumMulticastSolicitations`.

### 5.6.5 Configuration of Static On-Link Prefixes for IPv6

The IPv6 uses the on-link definition according to the IETF RFCs 4861 [33] and 5942 [41].

If an IPv6 packet shall be sent to a unicast destination the IPv6 determines whether or not the destination is on-link. A destination is on-link if the prefix of the destination address matches `fe80::/10` or any entry in the prefix list.

Packets to on-link destinations are sent directly to the target after the link layer address has been resolved (e.g. via an NDP Neighbor Solicitation).

Packets to destinations that are not known to be on-link must be sent to a router. If there is no router in the network, these IP packets are dropped.

According to IETF RFC 5942 the IPv6 does not treat a prefix derived from a statically or automatically configured address as on-link (except for the link-local prefix `fe80::/10`).

A prefix may be explicitly marked as on-link (added to the prefix list) via a received Router Advertisement containing a “Prefix Information” option with the “L”-flag set or via manual configuration of the prefix.

**Manual Configuration of On-Link Prefixes**

> The 64 bit prefix of a statically configured IPv6 address can be marked as on-link by setting the configuration parameter TcpIpIpV6AddrPrefixIsOnLink to “Enabled”.

> Alternatively any prefix can by statically added to the prefix list by specifying it via the configuration parameter TcpIpNdpOnLinkPrefix.

### 5.6.6 IPv6 address assignment

IPv6 addresses can have an assignment method that either triggered **automatically** after the ethernet link is up or **manually** during runtime via the API `TcpIp_RequestIpAddrAssignment()`. An address with the STATIC address assignment method can be set in the configuration or during runtime. For addresses that are set at runtime, the prefix must fit to the address type (Multicast: FF0X or Unicase: FE80). The address must also be unique for the IP controller/VLAN. For static addresses with `TcpIpStaticIpAddressConfig` container the default address can be assigned by using the IPv6 address [::] for the parameter `LocalIpAddrPtr`. The LinkLocal and AsAn address must be triggered automatically.

## 5.7 Static and Dynamic Link Layer Address Resolution

Every IPv4 instance has an independent dynamic address resolution cache that can hold a configurable number of entries. Each entry maps an IP address to the corresponding physical (MAC) address.

The entries in the cache are automatically added, updated and replaced by the Address Resolution Protocol (ARP, see [19]) during runtime based on received ARP Request and Reply packets. Entries in the ARP cache will not be updated on reception of gratuitous ARP request/reply due to security reasons. (OPEN_ALLIANCE_TC8_TEST_RELEVANT)

In addition to the dynamic address resolution cache a static table can be configured for each IPv4 instance. Static ARP tables may increase performance since no ARP packets need to be sent during runtime. Additionally, some ARP spoofing attacks can be prevented by using static ARP tables because static entries will not be overridden by information from received ARP packets.

A static ARP table may be shared by multiple IPv4 instances.

Each IPv4 instance can use one of the following address resolution modes:

> Dynamic ARP cache only

> Static ARP table only
> Static ARP table and dynamic ARP cache

**Configuration in DaVinci Configurator Pro**

The size of the dynamic ARP cache is defined by the parameter **[ARP Table Size]** of the **IpV4CtrlConfig**.

Static ARP Tables are defined in an **IpV4StaticArpTable** container that can referenced by the parameter **[Static ARP Table Ref]** of the **IpV4CtrlConfig**.

### 5.7.1 Prevention of ARP Flooding

Configuration parameter TcpIpArpRequestTimeout is introduced by AUROSAR in order to prevent ARP flooding. If upper layer triggers multiple transmission for the same destination within TcpIpArpRequestTimeout seconds, then only one ARP request will be sent to trigger address resolution. That implies, the ARP Request is sent only for those entries which are not yet valid but exists in Dynamic ARP table.

This feature is disabled if value of TcpIpArpRequestTimeout parameter is set to 0.

No ARP request is sent, and ARP entry will be discarded from the Dynamic ARP table, once TcpIpArpRetryTime for that ARP entry is expired.

**Configuration in DaVinci Configurator Pro**

/TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpArpConfig/TcpIpArpRequestTimeout

/TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpArpConfig/TcpIpArpRetryTime

**Note**

If the TcpIp module is configured for TCP, uses the UDP retry queue and/or ICMP Echo Reply functionality, the configuration of TcpIpArpRequestTimeout == 0 may lead to the transmission of ARP request every MainFunction cycle, if the destination address can not be resolved.

## 5.8 DHCP user options

The DHCPv4 as well as DHCPv6 client implementation of the TcpIp module supports configuration of DHCP user options. These user options are required to:

> Read out the value of received DHCP options by invoking TcpIp_DhcpReadOption() and TcpIp_DhcpV6ReadOption() functions.

> Add custom DHCP options to outgoing DHCP messages by invoking TcpIp_DhcpWriteOption() and TcpIp_DhcpV6WriteOption() functions.

This functionality enables the UpperLayer to include (TX), request (TX) and process (RX) DHCP options, which are not natively supported by the DHCP client implementation.

The configuration of user options shall be done via the configuration parameters listed within the following containers:

> `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV4Config/TcpIpDhcpConfig/TcpIpDhcpUserOption`

> `TcpIp/TcpIpConfig/TcpIpIpConfig/TcpIpIpV6Config/TcpIpDhcpV6Config/TcpIpDhcpV6UserOption`

The function <Up_DhcpEvent> (see 4.5.1.10) can be used to synchronize the write and read calls with the DHCP message reception and transmission.

If a DHCP user option is configured with direction `RX_REQUESTED`, the DHCP client will request this option via the Parameter Request List (DHCPv4) or the Option Request Option (DHCPv6).

**Example**

The functionality of DHCP user options can be uses to handle vendor class and vendor specific information which requires DHCP options as described in:

> DHCPv4: RFC3925: Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)

> DHCPv6: RFC3315 (chapter 22.16): Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

## 5.9 Support for user defined CSM APIs

TcpIp can be configured to use user defined CSM APIs instead of the MICROSAR Classic CSM APIs by using the parameter `TcpIpCsmAccessIndirectionPrefix.` The user must specify the prefix of the user defined CSM API. TcpIp will then use the CSM APIs defined by the user. By default, the MICROSAR Classic CSM prefix “Csm” is configured.

**Caution**

When this feature is enabled, all CSM APIs invoked by TcpIp are redirected to the user defined CSM API. The user must ensure that all CSM APIs are defined and declared in a header file. This header file must be included in the user config file configured in `TcpIpUserConfigFile.`

## 5.10 IPsec Configuration

**Note**
The IPsec implementation is limited to the functionality of Authentication Header (AH) processing in combination with IPv4 controllers.
IPv6 as well as Encapsulating Security Payload (ESP) Headers are not supported.

TcpIp allows each controller to optionally reference a `TcpIpIpSecConfigSet` via the `TcpIpIpSecConfigSetRef` which contains the following configurable elements. Each `TcpIpIpSecConfigSet` should specify a tentative number of remote devices that shall use the IPsec communication.

**Caution**
For all TcpIp controllers that use IPsec it must be ensured, that an Rx ethernet interrupt cannot be interrupted by another one.

If IPsec is used, the related Ethernet controller must operate in polling mode.

For transmitted packet with ESN the theoretically maximal payload size for IPsec-AH packets is reduced by 4 bytes that are used for calculations but are not transmitted.

### 5.10.1 SPD Configuration

The user can configure one or more SPD entries in each `TcpIpIpSecConfigSet` to specify security policies for different types of IP traffic.

While configuring an SPD entry, IP address ranges are configured by START and END parameters, which may be used in three different ways:

> Configure the START value to a valid IP address and do not configure the END value to link this SPD entry to this individual IP address.
> Configure the START as well as the END value to valid IP addresses to link this entry to this range of IP addresses.
> Do not configure START and END values to link this entry will all possible IP address values.

**Note**
The same configuration variants apply to the configuration of port ranges.

### 5.10.2 Integrity Algorithms

> The user must configure at least one integrity algorithm to use the IPsec module. The user can select one among the list of supported Integrity Transform Identifiers.

> The user must then configure at least one MAC Generate and Verify CSM job reference each from the CSM module, which matches the identifier selected.

**Caution**
The CSM module must be configured correctly to ensure functioning of IPsec module. Refer to the CSM technical reference for correct configuration of CSM module.

### 5.10.3 Encryption Algorithms

This container is not currently supported and must be left empty.

### 5.10.4 IPsec SA Entry Tx Activation Timeout

For each configured `TcpIpIpSecConfigSet`, the value of `TcpIpIpSecSaEntryTxActivationTimeout` determines the maximum time interval after which an SA entry created by IKE as responder will be activated for transmission. The default value of the parameter is 2 seconds.

**Note**
If an IPsec packet is received on a responder SA entry before `TcpIpIpSecSaEntryTxActivationTimeout` expires, then the SA entry is activated for transmission as soon as the packet is processed.

SA entries created as an initiator will be directly available after it is created.
The timeout influents only the Tx SA entry. It does not affect the Rx SA entry, that is directly available in both cases (IKE as initiator and responder).

This behavior avoid that the responder starts to transmit packets using the key material before the initiator successfully created the entries on his side.

## 5.11 Static Default Router Configuration

Default router addresses for IpV6 can be configured statically in the Static IPv6 address container `/TcpIp/TcpIpConfig/TcpIpLocalAddr/TcpIpStaticIpAddressConfig`. If more than one Static IPv6 address with Default Router is configured for the same controller, any of the default router may be selected during runtime (based on reachability and source address selected).

## 5.12 TLS Configuration

Figure 5-1 shows an overview of the configuration of the TCPIP module with TLS. Each TLS connection must refer to at least one `TlsCiphersuiteWorker`, at least one `TlsBufferConfig` and at least one `TlsCertificateIdentity`. All parameters of the mentioned container are described in the next chapters.

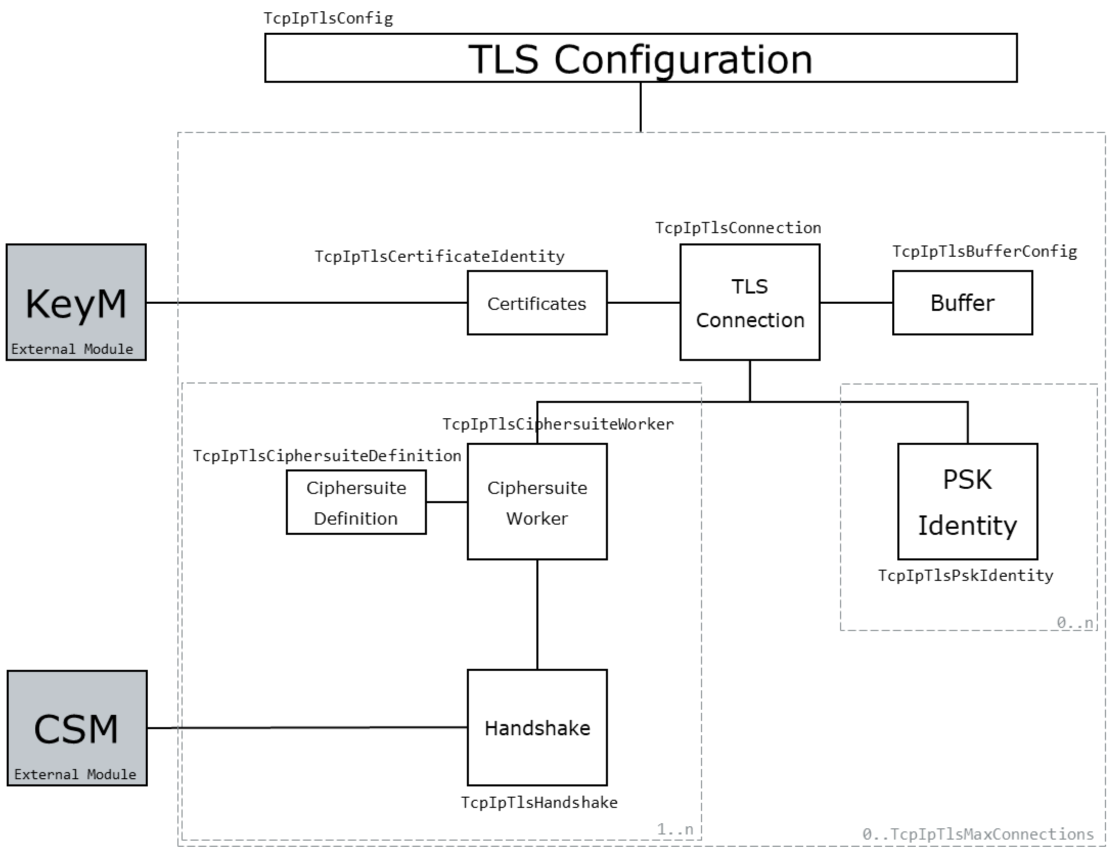


Figure 5-1 Overview TLS Configuration

**Note**
Due to incorrect definitions in the AUTOSAR model, it is necessary to configure parameters even if they are not used for the current use case (for example `TcpIpTlsCertificateIdentity`)

### 5.12.1 Container TcpIpTlsConfig

The container `TcpIpTlsConfig` holds the generic information for all configured TLS connections, such as the maximum number of server instances, the user error reporting (see User error reporting) or the secret access. Additionally, the CSM job configuration for the generic random value generator is located in this container (see Additional initializations – Random Generator).

#### 5.12.1.1 TLS Error Indication callout

The parameter `TcpIpTlsSupportErrorIndicationCallout` enables the detailed TLS error indication, which informs the user (upper layer) over raised TLS errors during the handshake or TLS data communication.

The Error Indication can be set to the following values:

| ErrorIndication | Value | Meaning |
|---|---|---|
| TCPIP_TLS_E_NOT_SET | 0x00 | No error indication is set. |
| TCPIP_TLS_E_SERVER_CERT_EXPIRED | 0x01 | The server's certificate has expired. |
| TCPIP_TLS_E_OCSP_RESP_NOT_GOOD | 0x02 | The server's OCSP response contains a certificate which is not 'good'. |
| TCPIP_TLS_E_UNKNOWN_CA | 0x03 | The issuing CA of the received server certificate is unknown. |
| TCPIP_TLS_E_SERVER_TERMINATED | 0x04 | The server terminated. |
| TCPIP_TLS_E_TX_ALERT | 0x80 | Parameter 'AlertDescription' is set to the transmitted alert. |
| TCPIP_TLS_E_RX_ALERT | 0x81 | Parameter 'AlertDescription' is set to the received alert. |

Table 5-3 TLS Error Indication values

**Note**
The parameter `AlertDescription` is only valid if the `ErrorIndication` is set to TCPIP_TLS_E_TX_ALERT or TCPIP_TLS_E_RX_ALERT. The value of the parameter is set to the received/transmitted TLS-Alert.

**Caution**
If the service is enabled, the User (upper layer) is responsible for implementing the API. The TLS stack provides only the prototype of the callout function.

#### 5.12.1.2 TLS Key Update Callout

The parameter `TcpIpTlsUseKeyUpdateCallout` enables the callout to the upper layer whenever a TLS key is updated.

**Caution**
If the service is enabled, the User (upper layer) is responsible for implementing the API. The TLS stack provides only the prototype of the callout function.

**Prototype**

```
typedef void (*TcpIp_SocketOwnerTlsKeyUpdateType)(
  TcpIp_SocketIdType       SocketId,
  TcpIp_TlsKeyUpdateType   KeyUpdateType);
```

**Parameter**

| | |
|---|---|
| `SocketId` | Socket identifier of the related local socket resource. |
| `KeyUpdateType` | Type of key update. One of:<br>`TCPIP_TLS_KEYUPDATE_NONE_TO_HANDSHAKE,`<br>`TCPIP_TLS_KEYUPDATE_NONE_TO_APPLICATION,` |

| | |
|---|---|
| | `TCPIP_TLS_KEYUPDATE_HANDSHAKE_TO_APPLICATION,` `TCPIP_TLS_KEYUPDATE_APPLICATION_TO_APPLICATION` |

Table 5-4 TLS Key Update Callout

From this callout the TLS Secrets API can be called by the user to ensure that every TLS secret (handshake/application) after any TLS key update is read.
The function should be defined in the implementation of the upper layer. The name of the function can be configured via `TcpIpSocketOwnerTlsKeyUpdateName`.

**Note**
Additional header files can be included via `TcpIpAdditionalIncludeFiles`.

### 5.12.2 Container TcpIpTlsConnection

The container `TcpIpTlsConnection` holds the information for each individual TLS connection, such as the Endpoint of this TLS connection (TLS Server or TLS Client), the TCP Port assignment and the relevant references to the Cipherworker and Certificates. Each TLS connection could only be used by one upper layer (user).

The parameter `TcpIpTlsConnectionPskIdentityRef` and `TcpIpTlsConnectionPskDefaultIdentityRef` both configure the TLS connection for pre shared keys in TLS 1.2. For further information on the selection of the correct PSK Identity see chapter *Behavior of MICROSAR Classic TCPIP with multiple PSK* Identities.

#### 5.12.2.1 Parameters for TLS Connection Assignment

To activate TLS on a specific TCP connection three parameters are considered:

- `TcpIpTlsIpAddressAssignment`
- `TcpIpTlsLocalAddrsRef`
- `TcpIpTlsPortAssignment`

All three parameters may be configured, but it is not mandatory that they are configured. `TcpIpTlsIpAddressAssignment` and `TcpIpTlsLocalAddrsRef` cannot be configured for the same TLS connection.

During connection assignment it is checked, if a TLS connection exits, where these parameters match to the TCP connection. If this is the case, TLS is activated for this TCP connection.

The selection of the TLS connection that is assigned to the TCP connection has the following priority:

**TLS Client**

1. A TLS connection was dynamically assigned via a ChangeParameter call (see 4.2.6 TcpIp_ChangeParameter)
2. A TLS connection has a matching `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` AND a matching port
3. A TLS connection has a no configured `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` parameter AND a matching port

**TLS Server**

1. A TLS connection was dynamically assigned via a ChangeParameter call (see 4.2.6 TcpIp_ChangeParameter)
2. A TLS connection has a matching `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` AND a matching port
3. A TLS connection has a matching `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` AND no configured port (wildcard port)
4. A TLS connection has a no configured `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` parameter AND a matching port
5. A TLS connection has a no configured `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` parameter AND no configured port (wildcard port)

**Note**
Only one of the parameters `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` may be configured for the same TLS connection. It is recommended to use the parameter `TcpIpTlsLocalAddrsRef`, because the parameter `TcpIpTlsIpAddressAssignment` only allows a reference to a static IP address.

#### 5.12.2.1.1 TLS Ip Address Assignment

The parameter `TcpIpTlsIpAddressAssignment` is an AUTOSAR parameter and may be used to specify the local IP address. The parameter only allows the configuration of static Ip addresses.

If the parameter is configured, it is used to activate TLS on a specific TCP connection during connection assignment. When assigning a TLS connection to a TCP socket, TLS connections with a matching `TcpIpTlsIpAddressAssignment` parameter will be given a higher priority over TLS connections without a specified `TcpIpTlsIpAddressAssignment` parameter.

**Note**
Only one of the parameters `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` may be configured for the same TLS connection. It is recommended to use the parameter `TcpIpTlsLocalAddrsRef`, because the parameter `TcpIpTlsIpAddressAssignment` only allows a reference to a static IP address.

##### 5.12.2.1.2 TLS Local Address Ref

The parameter `TcpIpTlsLocalAddrsRef` may be used to specify the local IP address. It has a multiplicity of 0:1. The parameter allows the configuration of all different kinds of IP addresses e.g., STATIC, LINKLOCAL (AUTO-IP) and DHCP.

If the parameter is configured, it is used to activate TLS on a specific TCP connection during connection assignment. When assigning a TLS connection to a TCP socket, TLS connections with a matching `TcpIpTlsLocalAddrsRef` parameter will be given a higher priority over TLS connections without a specified `TcpIpTlsLocalAddrsRef` parameter.

**Note**
Only one of the parameters `TcpIpTlsIpAddressAssignment` or `TcpIpTlsLocalAddrsRef` may be configured for the same TLS connection. It is recommended to use the parameter `TcpIpTlsLocalAddrsRef`, because the parameter `TcpIpTlsIpAddressAssignment` only allows a reference to a static IP address.

##### 5.12.2.1.3 TLS Port Assignment

The parameter `TcpIpTlsPortAssignment` is used to activate the TLS on a specific TCP connection, or if not set, the first option is to prepare the corresponding TLS connection for dynamic port usage. The second option, which is only available when this TLS connection is configured as a TLS server, is to use this TLS connection as a wildcard. See the following chapter for further configuration information.

###### 5.12.2.1.3.1 Static TLS port assignment (Via configuration)

This value represents either the **destination port** if the TLS connection is configured as a TLS client, or the **listen port** if the TLS connection is configured as a TLS server.

###### 5.12.2.1.3.2 Dynamic TLS port assignment (During runtime)

There are also possibilities to activate TLS on specific socket at runtime, which is done by not setting the parameter `TcpIpTlsPortAssignment` in the configuration. During runtime, the corresponding TLS connection can be assigned to a User Socket (which is already mapped to a specific port) via the 4.2.6 TcpIp_ChangeParameter API.

##### 5.12.2.1.3.3 Wildcard TLS port assignment (During runtime)

For the other possibility to activate TLS on specific socket at runtime, the parameter `TcpIpTlsPortAssignment` in the configuration also must not be set. During runtime, the corresponding TLS connection can be assigned to a User Socket when no free TLS connection with either static or dynamic port assignment can be found.

### 5.12.2.2 TLS Max Fragment Length

`TcpIpTlsMaxFragmentLength` parameter will be configured automatically to the available TLS RxBuffer size. This parameter is used to negotiate the maximum Record Size limit (RFC8449) if the corresponding TLS extension is enabled. See chapter 5.12.2.5.6 Extension 'Record Size Limit for TLS'.

**Caution**
Maximum fragment length negotiation (RFC8449) is NOT supported for TLS-Client connections (L002). It MUST be ensured that the remote peer is aware of a lower maximum fragment length. If remote peer sends a RL frame greater than the `TcpIpTlsMaxFragmentLength`, the message will be dropped.

### 5.12.2.3 TLS Certificate Identity Ref

The parameter `TcpIpTlsCertificateIdentityRef` references to the container that contains the certificate and identity information. This reference can be used from multiple different TcpIpTlsConnections. For further information see 5.12.3.1 TcpIpTlsCertificateIdentity.

**Caution**
Due to the multiplicity of 1:n of the container `TcpIpTlsCertificateIdentityRef` a valid reference must be set, even if the selected TLS connection does not use any certificates (PSK).

### 5.12.2.4 TcpIpTlsBufferConfig

The container `TcpIpTlsBufferConfig` holds the information about the sizes of the TLS transmit and receive buffers. The buffer configuration contains two parameters, one for the TLS Application data buffer size and one for the TLS handshake data buffer size. The receive and transmit sizes of each buffer can be configured separately and must not be same for each direction.

**TcpIpTlsAd<Rx/Tx>BufferSize**

This size gives information about the maximum number of plaintext (Application) data, a UpperLayer wants to transmit/receive. The size of the TLS Application data should therefore fit the requirements of the UpperLayer.

**TcpIpTlsHs<Rx/Tx>BufferSize**

The size of the TLS handshake buffer must be big enough to hold all received frames within the handshake (even all the certificates, if used). Otherwise, the TLS handshake will fail. This size is automatically calculated depending on the used configuration and the numbers of referenced certificates.

**Note**

The actual used generated size of the <Rx/Tx> Buffer is determined by taking the maximum of the configured TcpIpTlsAd<Rx/Tx>BufferSize and TcpIpTlsHs<Rx/Tx>BufferSize.

**TCP Socket Buffer Reference**

Via the parameter *TcpIpTlsBufferConfig/TcpIpTcpSocketBufferRef* a reference between the TLS connection buffers and the TCP socket buffers is configured. The TCP socket buffer must be greater than the maximum calculated TLS buffer, to ensure that the TCP can hold the encrypted TLS frame, which is larger than the plaintext frame. The offset (padding, HMAC) depends on the selected cipher suite. The configured size of the TCP socket buffer should be used for the assignment of the RX and TX windows via the 4.2.6 TcpIp_ChangeParameter API.

#### 5.12.2.5 TLS Extension

##### 5.12.2.5.1 Extension ‘EC Point Format’

Defined by ‘*RFC8422 - 5.1.2. Supported Point Formats Extension*’ this extension is added automatically by the TLS client when using an ECC based cipher suite. The only supported EC point format is the uncompressed(0) value of point formats.

##### 5.12.2.5.2 Extension ‘Signature Algorithm’

Defined by '*RFC5246 - 7.4.1.4.1. Signature Algorithms*' and '*RFC8422 - 5.1.3 The signature_algorithms Extension and EdDSA*' this extension is added automatically by the TLS client when using an ECC based cipher suite. The signature/hash algorithm pairs are determined for each configured *TcpIpTlsHandshake* container, referenced by the used TLS connection.

##### 5.12.2.5.3 Extension ‘Supported Groups’

Defined by '*RFC8422 - 5.1.1. Supported Elliptic Curves Extension'* this extension is added automatically by the TLS client when using an ECC based cipher suite. The named curve values are determined for each configured *TcpIpTlsHandshake* container, referenced by the used TLS connection.

##### 5.12.2.5.4 Extension ‘Trusted CA Indication’

Defined by ‘*RFC6066 - 6. Trusted CA Indication*’ this extension is only added to the ClientHello message when activated. To enable this extension, the parameter ‘*TcpIpTlsUseExtensionTrustedCaIndication’* must be set to TRUE. Currently the two types cert_sha1_hash(3) and key_sha1_hash(1) are supported. For all configured Issuer certificates the calculated hash will then be added to the extension. Which type of hash (key or certificate) is added, depends on the configuration of the issuer certificate.

**Note**

The calculation of the Issuer hash is performed in the KeyM. Therefore, all issuer certificates (KeyMCertificateGroupIssuerRef) must be configured with an corresponding CSM hash job (KeyMCertCsmHashJobRef).

**Note**

RFC6066 chapter 6 states the following:

"key_sha1_hash: contains the SHA-1 hash of the CA root key. For Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) keys, this is the hash of the "subjectPublicKey" value.”

It is not further specified which format the value of the subjectPublicKey should have. The current TLS implementation calculates the sha1 hash of the public key including the compression flag, but excluding the tag, length, and number of unused bits.

This is compliant to RFC5280 chapter 4.2.1.2. where it says:

“The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).”

##### 5.12.2.5.5 Extension ‘Status Request’

Defined by ‘*RFC6066 - 8. Certificate Status Request’* this extension is only added to the ClientHello message when activated. To enable this extension, the parameter ‘*TcpIpTlsUseExtensionCertificateStatusRequest’* must be set to TRUE. If the server replies with a certificate status extension as well as an OCSP certificate status, the TLS client will forward this information to the KeyM and update the status of the certificates. The UpperLayer is informed via TLS Validation Result callout if the TLS server sends a valid certificate status and the updated status of the certificate.

**Note**
To support OCSP responses the KeyM MUST be configured accordingly. Otherwise, the TLS handshake will fail.

#### 5.12.2.5.6 Extension 'Record Size Limit for TLS'

Defined by '*RFC8449 - Record Size Limit for TLS*' this extension is only added to the ServerHello message when activated. To enable this extension, the parameter '*TcpIpTlsUseSecurityExtensionRecordSizeLimit*' must be set to TRUE. The TLS-Server can only offer its own record size limit, if the received ClientHello message from the ClientHello also contains the extension. The value of the "*record_size_limit*" extension is the maximum size of record in octets that the endpoint is willing to receive. This value is used to limit the size of records that are created when encoding application data and the protected handshake message into records. The value is automatically set to the available TLS RxBuffer of this connection and indicated by the additional parameter '*TcpIpTlsMaxFragmentLength*'.

When the "*record_size_limit*" extension is negotiated, an endpoint MUST NOT generate a protected record with plaintext that is larger than the RecordSizeLimit value it receives from its peer. Unprotected messages are not subject to this limit. This value is the length of the plaintext of a protected record.

**Caution**
The record size limit negotiation defined by RFC8449 is only available for TLS-Server connections.

#### 5.12.2.5.7 Extension 'Server Name Indication'

Defined by *'RFC6066 - 3. Server Name Indication'* this extension is added automatically by the TLS-Client implementation when a server name is configured in the parameter `TcpIpTlsCertificateIdentity/TcpIpTlsServerNameIdentification`, referenced by the used TLS connection.

**Note**

The TLS-Client implementation does not check, if the endpoint uses the information provided in the server name indication extension. If necessary, this should be done by the upper layer.

**Caution**

The server name indication extension defined by '*RFC6066 - 3. Server Name Indication'* is only available for TLS-Client implementations.

#### 5.12.2.6 TLS external time callout function

The parameter `TcpIpTlsConnectionGetTimeFunc` allows the user to configure a callout function `<Up_TlsGetCurrentTimeStamp>` to provide a 4 byte timestamp. This timestamp is added to the random function (first 4 bytes) of the ClientHello or ServerHello message during the TLS handshake. The header file that holds the callout function could be included using the parameter `TcpIp/TcpIpGeneral/TcpIpAdditionalIncludeFiles`. If no callout function is provided by the user, the 4 byte timestamp in the random number is filled with 0.

**Consideration of the Internet-Draft: “Deprecating gmt_unix_time in TLS”**

In [51] it is recommended that the usage of the gmt_unix_time in the Hello messages of a TLS connection should be deprecated. Instead, TLS implementors MUST by default set the entire value of the ClientHello.Random and ServerHello.Random fields, including gmt_unix_time, to a cryptographically random sequence. To support this recommendation, the external time callout function can be used. Therefore, the provided time stamp from the callout function is not checked against validity or format correctness. This means, if the external time callout function provides the TLS stack with 4 Bytes of additional random data instead of a Unix time stamp, those Bytes will be added to the corresponding Hello message random fields.

### 5.12.3 Container TcpIpTlsCiphersuites

#### 5.12.3.1 TcpIpTlsCertificateIdentity

The container `TcpIpTlsCertificateIdentity` holds the information about the certificates if the connection is configured with an ECC cipher and reference to the external KeyM module.

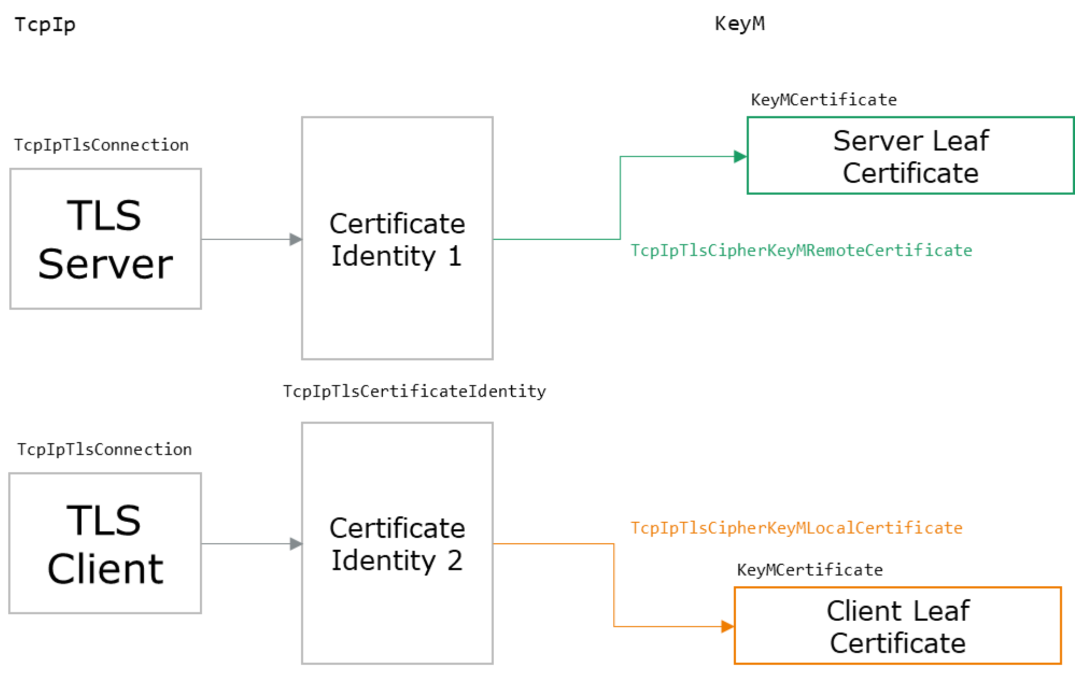


Figure 5-2 Configuration of TLS Certificate Identity

**Note**
Due to the multiplicity of the AUTOSAR model, each TLS connection MUST refer to a `TcpIpTlsCertificateIdentity`, even if there are no Certificates in use (e.g PSK cipher suites). In this case, the `TcpIpTlsCipherKeyMRemoteCertificate` and `TcpIpTlsCipherKeyMLocalCertificate` MUST be empty.

#### 5.12.3.2 TLS connection used as TLS Server

In case of an TLS Server, the corresponding `TcpIpTlsCipherKeyMLocalCertificate` must be configured. The local certificate reference MUST point to a valid and installed TLS Server certificates for this TLS connection. The installed TLS Server certificate chain MUST be valid at the point the TLS handshake is established. Also see chapter 2.1.5.1.3 Update of stored certificates for information on the certificate update during runtime of the ECU.

##### 5.12.3.2.1 Client Authentication

If the TLS Server connection has `TcpIpTlsUseClientAuthenticationRequest` enabled, `TcpIpTlsCipherKeyMRemoteCertificate` must also be configured. It shall point to the peer's connection Client leaf certificate, which is configured in the KeyM. A valid KeyM configuration must be used, depending on the expected number of received Client certificates (chain depth). The used certificates MUST be configured in a certificate group (`KeyMCertificateGroup`). In the same certificate group, all certificates which are expected to be received by the TLS Server during the TLS handshake shall be referenced regardless of their elliptic curve.

**Caution**
The TLS Server is not responsible for adding and installing the certificate chain. This is part of the <UpperLayer> TLS startup.

#### 5.12.3.3 TLS connection used as TLS Client

In case of an TLS Client, the corresponding `TcpIpTlsCipherKeyMRemoteCertificate` must be configured accordingly to the expected received remote certificate chain. The remote certificate reference should point to the peer's connection Server leaf certificate, which is configured in the KeyM. A valid KeyM configuration must be used, depending on the expected number of received Server certificates (chain depth). The used certificates MUST be configured in a certificate group (`KeyMCertificateGroup`) with one or more referenced certificate issuers (`KeyMCertificateGroupIssuerRef`). In the same certificate group all certificates which are expected to be received by the TLS Client during the TLS handshake shall be referenced regardless of their elliptic curve.

##### 5.12.3.3.1 Client Authentication

If the TLS Client connection has `TcpIpTlsUseClientAuthenticationRequest` enabled, `TcpIpTlsCipherKeyMLocalCertificate` must also be configured. It shall point to a Client leaf certificate, which is configured in the KeyM.

##### 5.12.3.3.2 Server Name Identification

The parameter `TcpIpTlsServerNameIdentification` is only used for TLS-Client implementations (see limitation L005 – Server Name Indication).

#### 5.12.3.4 TcpIpTlsCiphersuiteDefinition

The container `TcpIpTlsCiphersuiteDefinition` holds the information about the cipher suite ID, for example the IANA name and the Hex code of the ID. This container is statically predefined and could not be changed. All currently supported cipher suites will be present there.

**Note**
The TCPIP will not use the priority of the `TcpIpTlsCiphersuitePriority` parameter. Instead, the priority will be set directly in the cipher suite worker. See chapter TcpIpTlsCiphersuiteWorker

#### 5.12.3.5 TcpIpTlsCiphersuiteWorker

The container `TcpIpTlsCiphersuiteWorker` combines the TLS handshake part with the TLS streaming (de-, encryption) and authentication (HMAC).

### Cipher based TLS streaming

Depending on the selected `TcpIpTlsCiphersuiteDefinitionRef` the cipher worker must be configured to support the used streaming method (e.g., AES128-CBC). Therefore, the decrypt and encrypt CSM Job/Keys must reference to a valid CSM configuration. In case of a NULL cipher these references could be left empty.

### Cipher based TLS authentication

Depending on the selected `TcpIpTlsCiphersuiteDefinitionRef` the cipher worker must be configured to support the used authentication method (e.g., SHA256 based HMAC). Therefore, the MAC Generate, and MAC Verify CSM Job/Keys must reference to a valid CSM configuration. If the configured streaming method (e.g. AES in the GCM operation mode) already uses an authentication mechanism, these parameters are optional.

If an TLS connection should be configured for multiple different cipher suites, then an individual `TcpIpTlsCiphersuiteWorker` parameter must be created for each unique cipher suite that is referred to the TLS connection. If the TLS handshake is the same for each cipher suite (e.g., PSK) then the individual cipher worker could refer to the same '`TcpIpTlsHandshake`' *(see* Figure 5-3 Example TcpIpTlsCiphersuiteWorker configuration*)*.

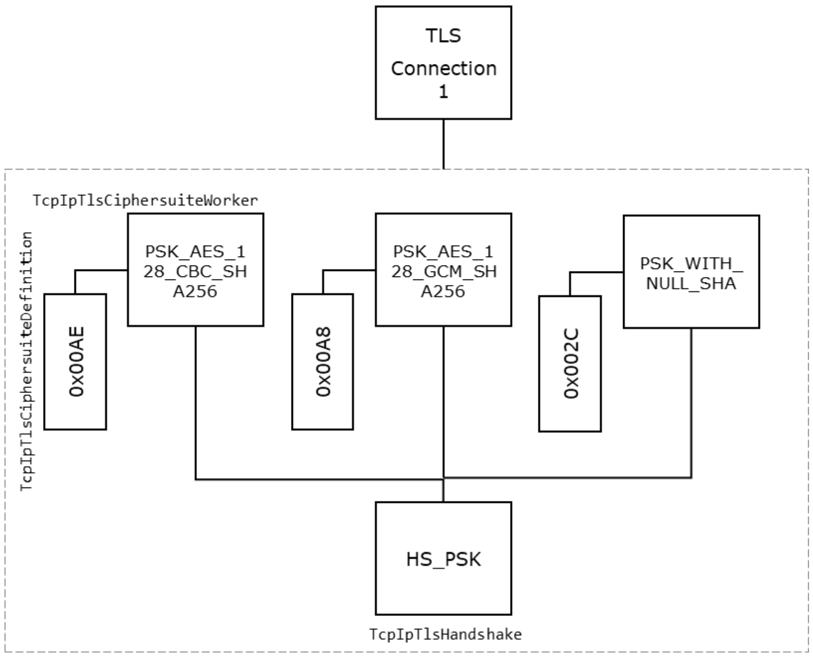


Figure 5-3 Example TcpIpTlsCiphersuiteWorker configuration

The parameter `TcpIpTlsConnectionHandshakeRef` reference to container that contains the information which is used to calculate the key exchange for this TLS connection. The configuration must match with the selected `TcpIpTlsCiphersuiteDefinitionRef`.

#### 5.12.3.6 TcpIpTlsHandshake

The container `TcpIpTlsHandshake` combines information that is needed to process a TLS handshake. It contains the appropriate references to the CSM jobs and keys, which are needed to perform the key exchange cryptographic such as the ECC Diffie Hellman operations. In this container also the relevant settings for the Hash operation, which is used to calculate the ‘Finish Message’, are stored.

The two additional Vector parameters `TcpIpTlsHsSignatureAlgorithm` and `TcpIpTlsHsSupportedEllipticCurve` must be set according to the selected cipher suite of this handshake. Those values are used to determine the correct handshake reference during runtime.

##### 5.12.3.6.1 Selection of handshake reference during runtime

The TLS stack is able to determine the correct handshake reference automatically during runtime, if multiple different certificates should be supported. Therefore, multiple configured `TcpIpTlsHandshake` container with different curve types can be added to one cipher worker. During the parsing of the ClientHello message, the TLS-Server analyzes the received client extensions and automatically selects the correct handshake reference, which is capable of performing the TLS handshake. For each added `TcpIpTlsHandshake` container a suitable certificate must be configured. This means, if a cipher worker should perform a TLS handshake with the SECP256R1 curve and the ED25519 handshake, two separate configured `TcpIpTlsHandshake` containers must be configured, as well as two different `TcpIpTlsCertificateIdentityRef`.

**Caution**
The dynamically selection of handshake references is only supported by the TLS-Server. TLS-Client therefore must be configured with exactly one matching handshake reference. If multiple curves should be supported, multiple cipher worker must be configured.

The `TcpIpTlsCsmKeyExchangeKeyRef` reference must be set to an appropriate CSM configuration which is used to perform the key exchange of the TLS connection. This must match with the used TLS cipher suite and the used ECC parameters of the used TLS certificate.

The `TcpIpTlsCsmKeyExchangeSignature*` reference must be set to an appropriate CSM configuration which is used to generate or validate the signature of the ServerKeyExchange or the CertificateVerify message in case of client authentication during the TLS handshake. This must match with the used TLS cipher suite and the used ECC parameters of the used TLS certificate.

**Note**
Since TcpIp version 17.01.01 `TcpIpTlsCsmKeyExchangeSignatureVerifyJobRef` and `TcpIpTlsCsmKeyExchangeSignatureVerifyKeyRef` should not be configured anymore. The job/key is derived dynamically during runtime.

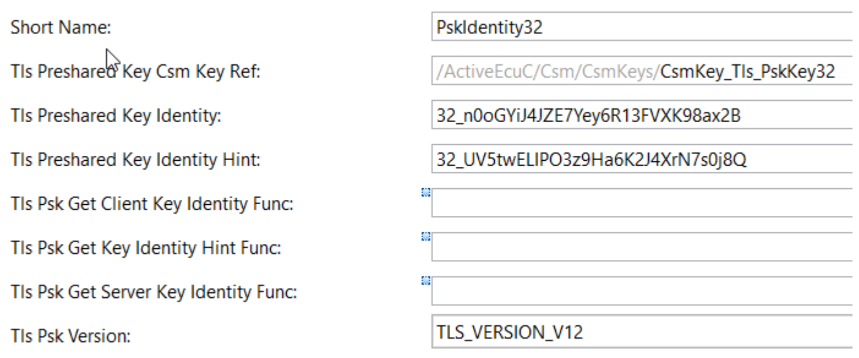
For further information on the Crypto configuration see Chapter 5.12.6 Crypto stack configuration.

**Note**
Due to the multiplicity of the AUTOSAR model, the Key Exchange reference MUST be set even if they are not used (e.g. with PSK). In this case the reference should point to an CSM Dummy element.

#### 5.12.3.7 TcpIpTlsPskIdentity

The container `TcpIpTlsPskIdentity` can be used either for TLS 1.2 for TLS 1.3. This is indicated by the parameter `TcpIpTlsPskVersion`. For the TLS 1.3 configuration see TLS 1.3 Session Resumption Configuration.

In TLS 1.2 the container holds the information about the pre shared key identity, such as the reference to the used CSM key `TcpIpTlsPresharedKeyCsmKeyRef`, the actual PSK Identity as string `TcpIpTlsPresharedKeyIdentity` *and optionally* the PSK hint as string `TcpIpTlsPresharedKeyIdentityHint`.



| | |
|---|---|
| Short Name: | PskIdentity32 |
| Tls Preshared Key Csm Key Ref: | /ActiveEcuC/Csm/CsmKeys/CsmKey_Tls_PskKey32 |
| Tls Preshared Key Identity: | 32_n0oGYiJ4JZE7Yey6R13FVXK98ax2B |
| Tls Preshared Key Identity Hint: | 32_UV5twELIPO3z9Ha6K2J4XrN7s0j8Q |
| Tls Psk Get Client Key Identity Func: | |
| Tls Psk Get Key Identity Hint Func: | |
| Tls Psk Get Server Key Identity Func: | |
| Tls Psk Version: | TLS_VERSION_V12 |

Figure 5-4 Example PSK Identity configuration

**Note**
The configured key size of the `TcpIpTlsPresharedKeyCsmKeyRef` MUST match with the actual size of the key, which will be loaded by the user. Otherwise, the TLS handshake will fail.

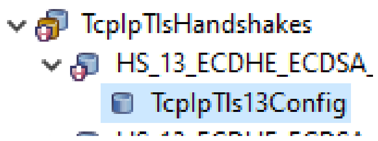
For further information how the PSK key can be loaded see chapter 5.12.7.3 Additional initializations – Loading the PSK keys.

For further information how the PSK Identity can be mapped to a specific TLS connection see Chapter 5.12.1.2.

### 5.12.4 TLS 1.3 Configuration

Additional to the configuration parameter that are required for TLS 1.2, some parameters are required for TLS 1.3 connections.

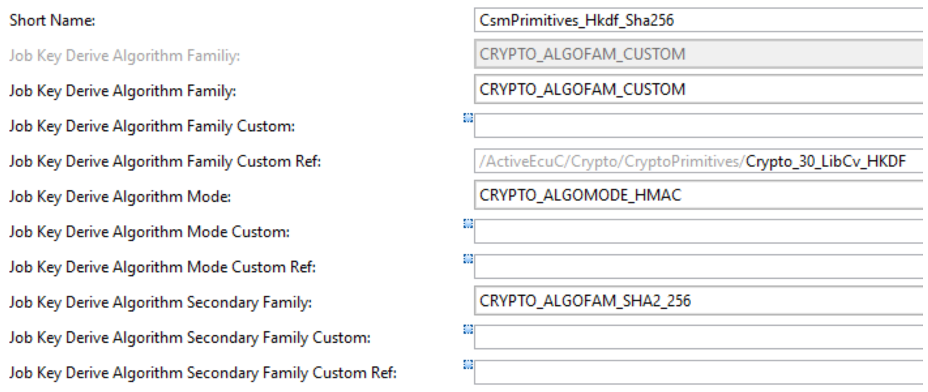
These are collected under the TcpIpTls13Config container of the TcpIpTlsHandshakes.



- TcpIpTlsHandshakes
  - HS_13_ECDHE_ECDSA_
    - TcpIpTls13Config

#### 5.12.4.1 TLS 1.3 HKDF Configuration

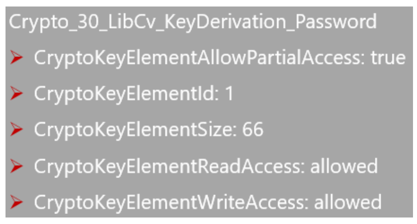
For TLS 1.3 for the HKDF key calculation a CSM job must be configurated. This job must be of type CsmJobKeyDerive and have a fitting primitive:



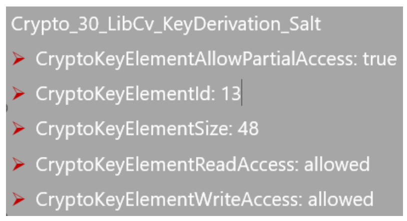
| | |
|---|---|
| Short Name: | CsmPrimitives_Hkdf_Sha256 |
| Job Key Derive Algorithm Familiy: | CRYPTO_ALGOFAM_CUSTOM |
| Job Key Derive Algorithm Family: | CRYPTO_ALGOFAM_CUSTOM |
| Job Key Derive Algorithm Family Custom: | |
| Job Key Derive Algorithm Family Custom Ref: | /ActiveEcuC/Crypto/CryptoPrimitives/Crypto_30_LibCv_HKDF |
| Job Key Derive Algorithm Mode: | CRYPTO_ALGOMODE_HMAC |
| Job Key Derive Algorithm Mode Custom: | |
| Job Key Derive Algorithm Mode Custom Ref: | |
| Job Key Derive Algorithm Secondary Family: | CRYPTO_ALGOFAM_SHA2_256 |
| Job Key Derive Algorithm Secondary Family Custom: | |
| Job Key Derive Algorithm Secondary Family Custom Ref: | |

For algorithm that use a SHA2-384 algorithm the secondary algorithm family must be changed.

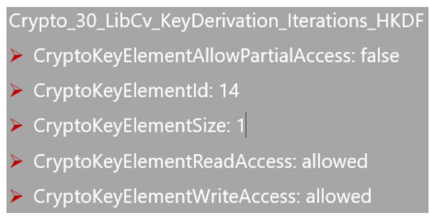
Configuration of the key for the HKDF job:



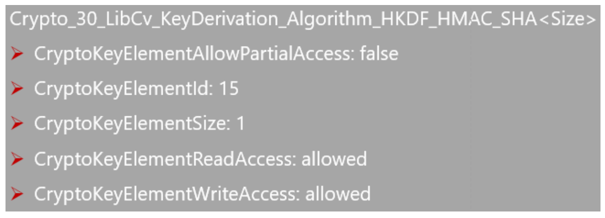
Crypto_30_LibCv_KeyDerivation_Password
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 66
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed



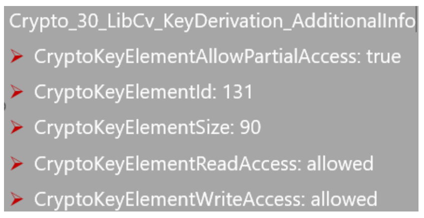
Crypto_30_LibCv_KeyDerivation_Salt
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 13
- CryptoKeyElementSize: 48
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed



Crypto_30_LibCv_KeyDerivation_Iterations_HKDF
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 14
- CryptoKeyElementSize: 1
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed



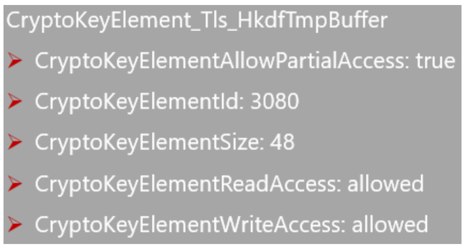
Crypto_30_LibCv_KeyDerivation_Algorithm_HKDF_HMAC_SHA<Size>
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 15
- CryptoKeyElementSize: 1
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed



Crypto_30_LibCv_KeyDerivation_AdditionalInfo
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 131
- CryptoKeyElementSize: 90
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

**Caution**
Make sure that the key elements KeyDerivation_Password, KeyDerivation_Salt and KeyDerivation_AdditionalInfo are configured big enough for the used algorithm. Otherwise, a wrong secret is calculated, and messages cannot be decrypted successfully. Configuring the parameters bigger than needed is OK since the ECU will shorten them to the needed length during runtime.

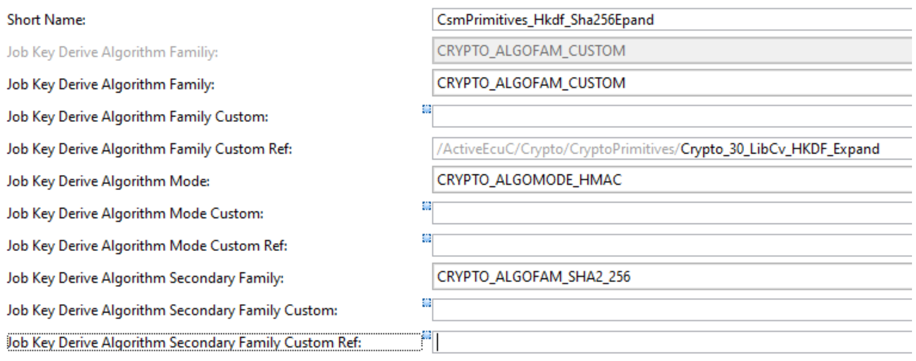
Configuration of key for TcpIpTlsHsTemporaryHdfkKeyRef:


CryptoKeyElement_Tls_HkdfTmpBuffer
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 48
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

The HKDF- job can be used for more than one Handshake as long the HMAC type is equal.

### 5.12.4.2 TLS 1.3 HKDF-Expand Configuration

For TLS 1.3 for the HKDF key expansion a CSM job must be configurated. This job must be of type CsmJobKeyDerive and have a fitting primitive:

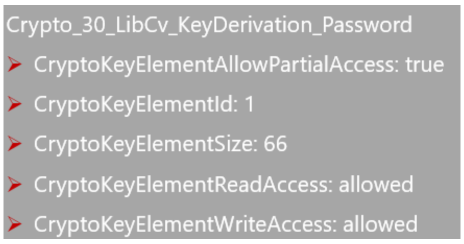


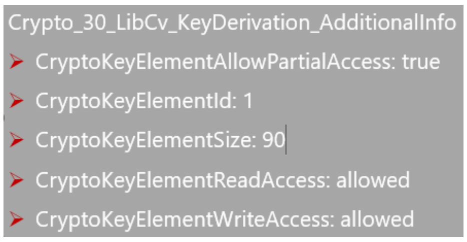
| | |
|---|---|
| Short Name: | CsmPrimitives_Hkdf_Sha256Epand |
| Job Key Derive Algorithm Familiy: | CRYPTO_ALGOFAM_CUSTOM |
| Job Key Derive Algorithm Family: | CRYPTO_ALGOFAM_CUSTOM |
| Job Key Derive Algorithm Family Custom: | |
| Job Key Derive Algorithm Family Custom Ref: | /ActiveEcuC/Crypto/CryptoPrimitives/Crypto_30_LibCv_HKDF_Expand |
| Job Key Derive Algorithm Mode: | CRYPTO_ALGOMODE_HMAC |
| Job Key Derive Algorithm Mode Custom: | |
| Job Key Derive Algorithm Mode Custom Ref: | |
| Job Key Derive Algorithm Secondary Family: | CRYPTO_ALGOFAM_SHA2_256 |
| Job Key Derive Algorithm Secondary Family Custom: | |
| Job Key Derive Algorithm Secondary Family Custom Ref: | |

For Jobs that use a HMAC-SHA2-384 the secondary family must be changed.

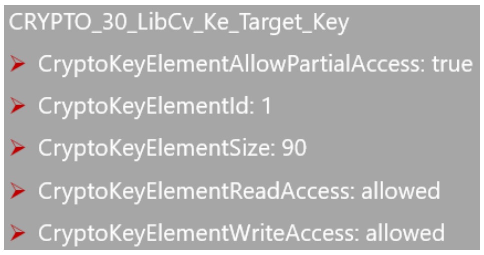
The key-elements should be configured like that:

Input-key:


Crypto_30_LibCv_KeyDerivation_Password
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 66
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed


Crypto_30_LibCv_KeyDerivation_AdditionalInfo
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 90
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

Output-key:


CRYPTO_30_LibCv_Ke_Target_Key
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 90
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

The input key is configured in the HKDF-expand job while the output key is configured in the TLS13Config container.

The usage of shorter keys can result in a runtime issue at the handshake.

The HKDF-Expand job can be used for more than one Handshake as long the HMAC type is equal.

#### 5.12.4.3 TLS 1.3 Session Resumption Configuration

To use the TLS 1.3 Session Resumption first a resumption base connection needs to be configured. To do this configure a regular TLS 1.3 connection and additionally configure the parameter `TcpIpTlsCsmResumptionMasterSecretKeyRef`. Furthermore, it needs to reference at least one TLS 1.3 `TcpIpTlsPskIdentity` container in `TcpIpTlsConnectionPskIdentityRef`. More precisely the number of referenced `TcpIpTlsPskIdentity` containers should be at least the number of resumed connections which may be used in parallel times the amount of different hash algorithms used by the referenced ciphersuites. Then at least one resumed connection needs to be configured. The resumed connection is also a TLS 1.3 connection with `TcpIpTlsCsmResumptionMasterSecretKeyRef` configured. The difference to the resumption base connection is that it does not reference any TLS 1.3 `TcpIpTlsPskIdentity` containers in `TcpIpTlsConnectionPskIdentityRef` and that `TcpIpTlsResumptionBaseConnectionId` is configured to the `TcpIpTlsConnectionId` of the resumption base connection.

To configure a `TcpIpTlsPskIdentity` container for session resumption `TcpIpTlsPskVersion` must be set to TLS 1.3. Also, different references to CSM keys and jobs must be configured which are needed for the PSK key calculation. The parameter `TcpIpTls13PskHashAlgorithm` defines the hash algorithm for which this PSK identity can be used. For each hash algorithm in the ciphersuites of the resumption base connection there should be one PSK identity with `TcpIpTls13PskHashAlgorithm` set to this hash algorithm.

Finally, in the `TcpIpTlsConfig` container some parameters for different timings, a maximum length and extension handling have to be set.

Additionally, a callout can be configured where the user is updated about available tickets for the resumption base connection (see <Up_SessionTicketUpdateCallout>).

### 5.12.5 Pseudo Random Function (PRF) configuration

The TLS stack implements the Pseudo Random Function (PRF) as described in RFC5246 – Section 5 HMAC and the Pseudorandom Function using the CSM and its input/ output redirection feature. This chapter gives an overview how to configure the necessary CSM Jobs and Keys accordingly.

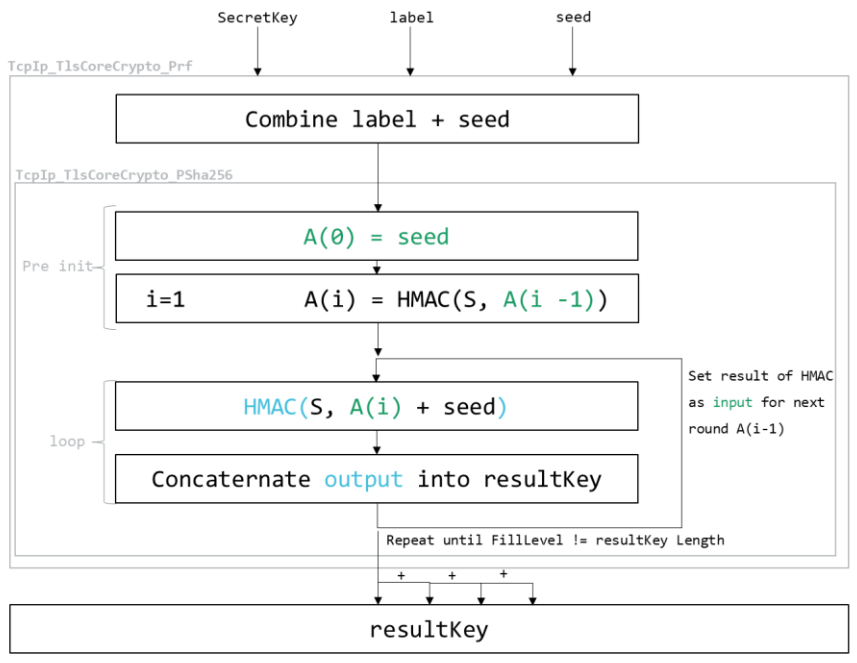

SecretKey
label
seed
TcpIp_TlsCoreCrypto_Prf
Combine label + seed
TcpIp_TlsCoreCrypto_PSha256
Pre init
A(0) = seed
i=1
A(i) = HMAC(S, A(i -1))
Set result of HMAC as input for next round A(i-1)
loop
HMAC(S, A(i) + seed)
Concaternate output into resultKey
Repeat until FillLevel != resultKey Length
+
+
+
resultKey

Figure 5-5 PRF Algorithm Overview

Figure 5-5 gives an overview of the implemented PRF algorithm, which main goal is to take a secret, a seed, and an identifying label and produces an output of arbitrary length, which is then used in different contexts of the TLS connection (Figure 5-6).

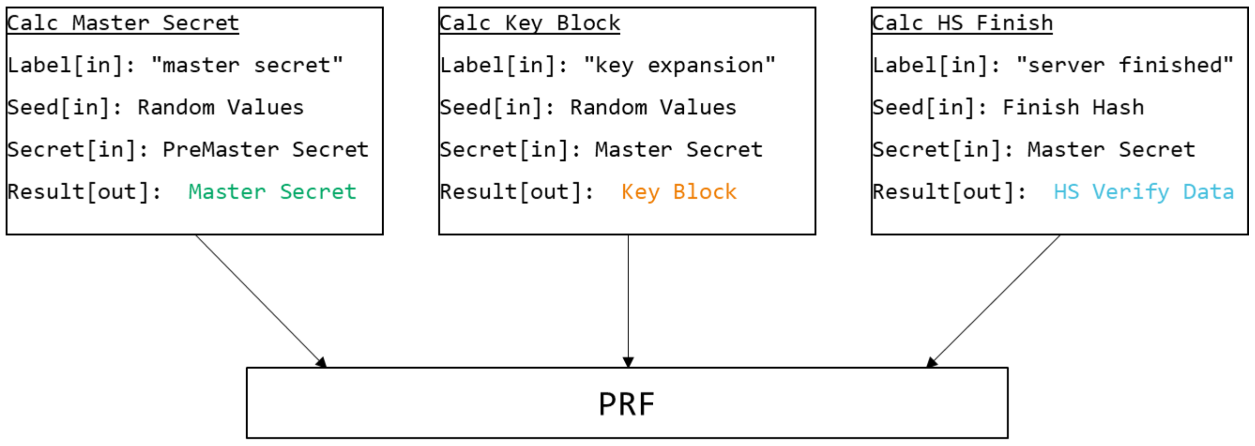

Calc Master Secret
Label[in]: "master secret"
Seed[in]: Random Values
Secret[in]: PreMaster Secret
Result[out]: Master Secret
Calc Key Block
Label[in]: "key expansion"
Seed[in]: Random Values
Secret[in]: Master Secret
Result[out]: Key Block
Calc HS Finish
Label[in]: "server finished"
Seed[in]: Finish Hash
Secret[in]: Master Secret
Result[out]: HS Verify Data
PRF

Figure 5-6 Different PRF contexts within TLS

The produced output result of the PRF is used in combination with the following CSM Keys:

TcpIpTlsCsmMasterSecretKeyRef: Stores the calculated master secret of this TLS connection. TcpIpTlsCsmPrfResultPrivateKeyRef: Stores the calculated Key Block of this TLS connection. TcpIpTlsCsmPrfResultPublicKeyRef: Stores the calculated handshake Verify Data, which is transmitted during TLS handshake.

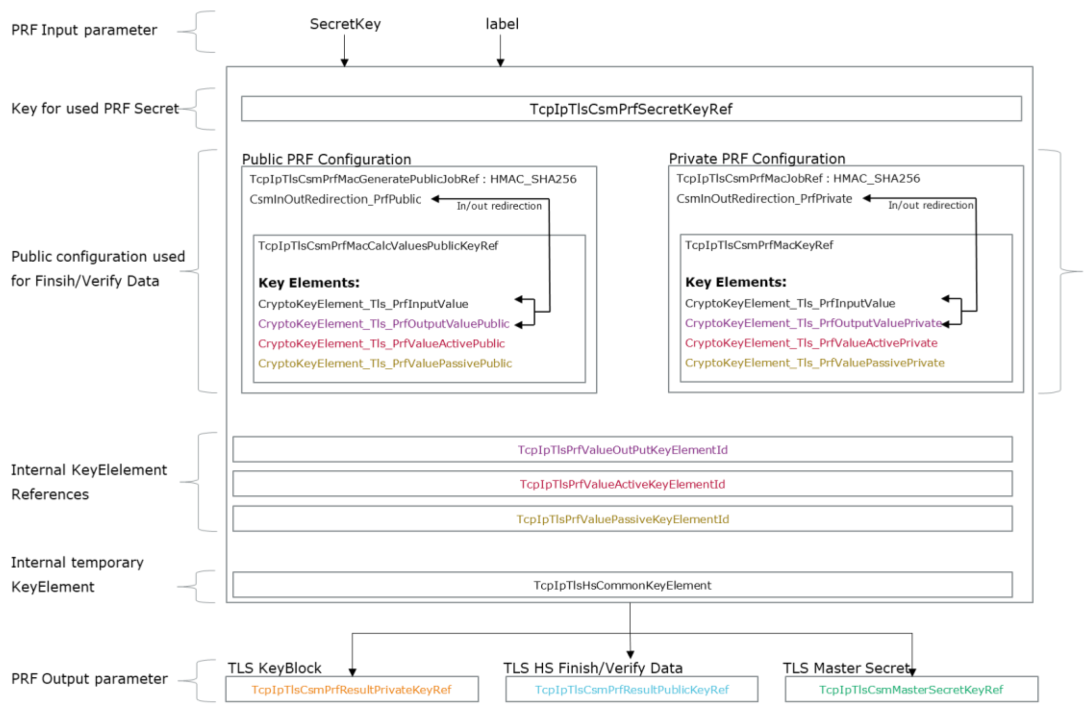

PRF Input parameter
SecretKey
label
Key for used PRF Secret
TcpIpTlsCsmPrfSecretKeyRef
Public PRF Configuration
TcpIpTlsCsmPrfMacGeneratePublicJobRef : HMAC_SHA256
CsmInOutRedirection_PrfPublic
In/out redirection
TcpIpTlsCsmPrfMacCalcValuesPublicKeyRef
Key Elements:
CryptoKeyElement_Tls_PrfInputValue
CryptoKeyElement_Tls_PrfOutputValuePublic
CryptoKeyElement_Tls_PrfValueActivePublic
CryptoKeyElement_Tls_PrfValuePassivePublic
Private PRF Configuration
TcpIpTlsCsmPrfMacJobRef : HMAC_SHA256
CsmInOutRedirection_PrfPrivate
In/out redirection
TcpIpTlsCsmPrfMacKeyRef
Key Elements:
CryptoKeyElement_Tls_PrfInputValue
CryptoKeyElement_Tls_PrfOutputValuePrivate
CryptoKeyElement_Tls_PrfValueActivePrivate
CryptoKeyElement_Tls_PrfValuePassivePrivate
Public configuration used for Finsih/Verify Data
Private configuration used for Key Block and Master Secret
Internal KeyElelement References
TcpIpTlsPrfValueOutPutKeyElementId
TcpIpTlsPrfValueActiveKeyElementId
TcpIpTlsPrfValuePassiveKeyElementId
Internal temporary KeyElement
TcpIpTlsHsCommonKeyElement
PRF Output parameter
TLS KeyBlock
TcpIpTlsCsmPrfResultPrivateKeyRef
TLS HS Finish/Verify Data
TcpIpTlsCsmPrfResultPublicKeyRef
TLS Master Secret
TcpIpTlsCsmMasterSecretKeyRef

Figure 5-7 Detailed PRF CSM Configuration

The ‘*TcpIpTlsCsmPrfSecretKeyRef’* is used to temporarily store the secret (Key) for the HMAC operation, since the PRF is used in different contexts. The referenced Key should configured to be RA_INTERNAL_COPY and must have the element size of the master secret (48 Bytes).

The ‘*TcpIpTlsHsCommonKeyElement’* is used to temporarily combine the result of the first HMAC operation with the seed to prepare the HMAC operation of the next round (HMAC( A(i) + seed)). This concatenation is performed by using the internal CSM copy partial API and thus, the key is configured to be RA_INTERNAL_COPY and must have the element size to store the operation of the concatenation (e.g., 80 Bytes).

**Public and Private PRF Configuration**

Since the PRF is used in different contexts the requirements to the security of the used CSM Keys differ. When using the PRF to calculate the Finish message verify data, which is transmitted during the TLS handshake, the public PRF configuration is used. When calculation the internal Master Secret and derivate the TLS Key Block, the private PRF configuration is used. Both configurations must contain the same Key Elements IDs ‘*CryptoKeyElementId’* but can differ in the read and write access controls (See Figure 5-9). The internal Key Element references should point to the public PRF configuration.

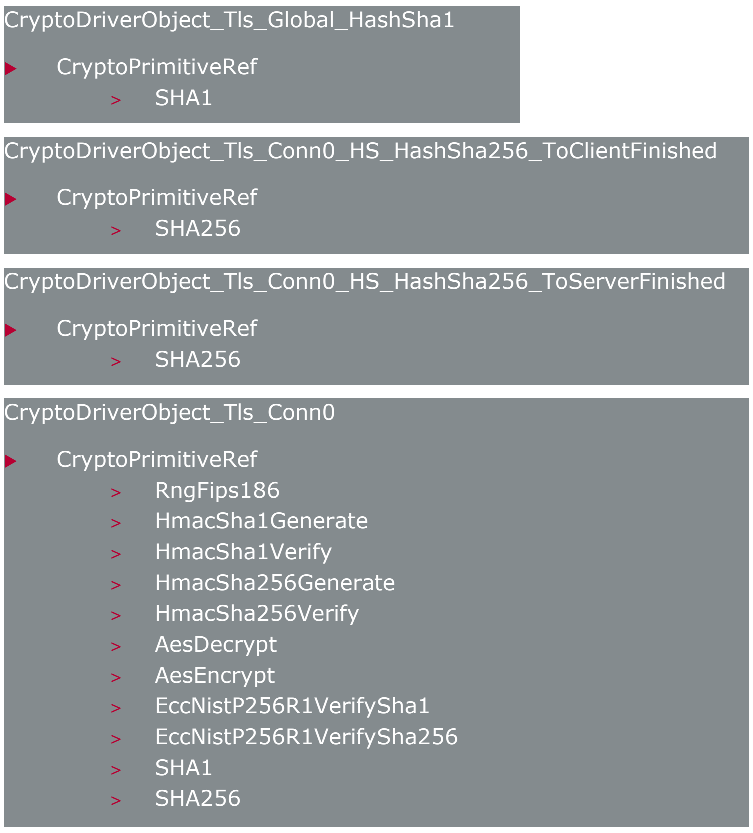
### 5.12.6 Crypto stack configuration

This chapter gives a detailed overview over the crypto stack modules configuration (the Csm, CryIf and the Crypto driver on the host CPU). It is required to enable the relevant cryptographic algorithms in the used crypto hardware or software library.

#### 5.12.6.1 Crypto driver

##### 5.12.6.1.1 Crypto driver objects configuration

It is required to have multiple driver objects, because of possible parallel crypto operations. There is one driver object for the global hashes of certificate meta information, and there are 3 driver objects for the connection specific operations for each connection.



```
CryptoDriverObject_Tls_Global_HashSha1
    CryptoPrimitiveRef
        SHA1
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToClientFinished
    CryptoPrimitiveRef
        SHA256
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToServerFinished
    CryptoPrimitiveRef
        SHA256
CryptoDriverObject_Tls_Conn0
    CryptoPrimitiveRef
        RngFips186
        HmacSha1Generate
        HmacSha1Verify
        HmacSha256Generate
        HmacSha256Verify
        AesDecrypt
        AesEncrypt
        EccNistP256R1VerifySha1
        EccNistP256R1VerifySha256
        SHA1
        SHA256
```

Figure 5-8 Crypto driver objects configuration

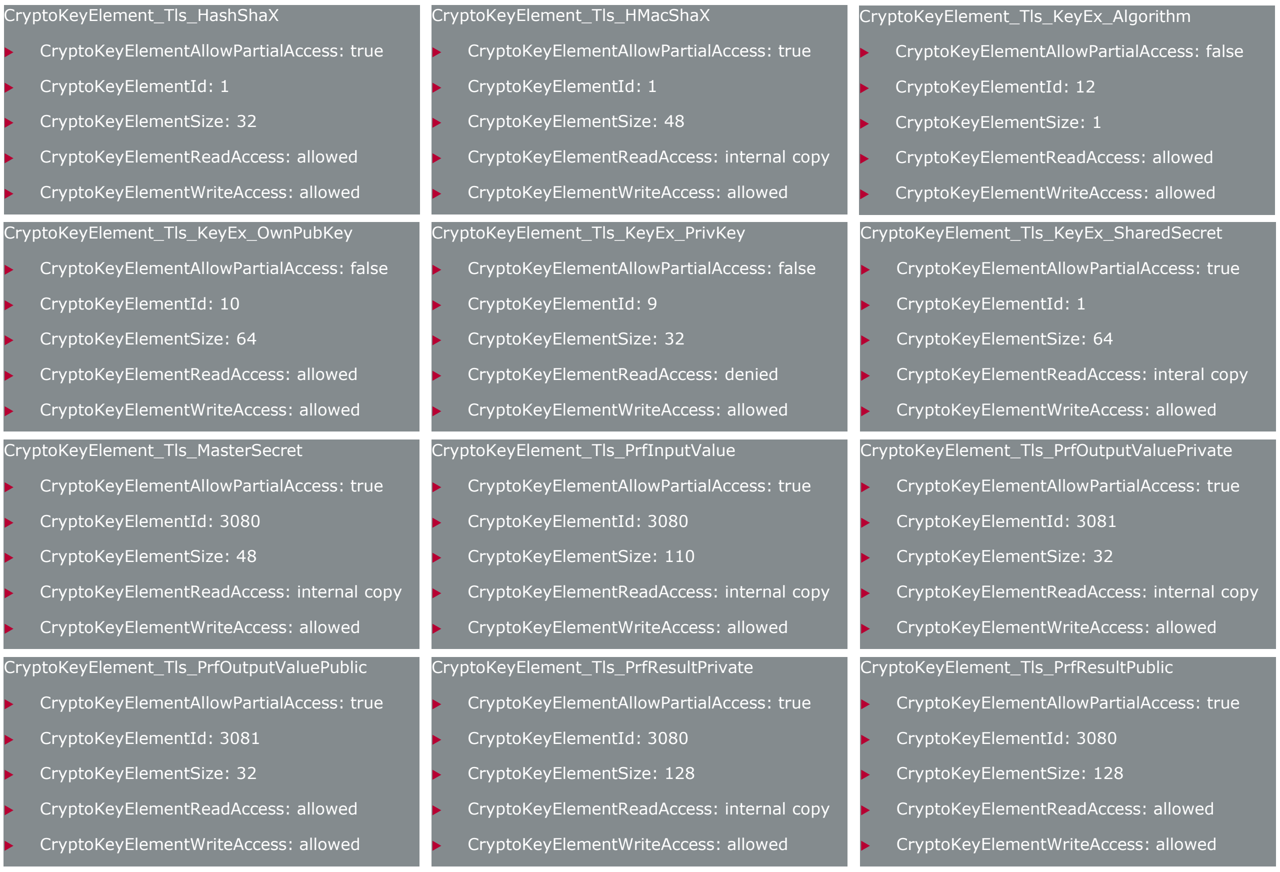
##### 5.12.6.1.2 Crypto driver objects when using HSM crypto driver

If the TLS stack uses a dedicated hardware module (HSM) as crypto driver, instead of a crypto software implementation (LibCv) not only the number of crypto driver objects is relevant, but also the number of configured IPC channels (CryptoIpcChannels) in the HSM configuration. The number of IPC channels must be sufficient to handle all simultaneous accesses to the HSM (both from the TLS and from other SWCs). Otherwise, the call to a CSM API may return CRYPTO_E_BUSY error, which will result in closing the TLS connection. For further information see the technical documentation of the crypto driver.

**Caution**
The TLS implementation cannot handle busy crypto driver or CSM calls (CRYPTO_E_BUSY). If any CSM function call returns CRYPTO_E_BUSY, the TLS connection will get interrupted and closed via a TLS-Alert.

#### 5.12.6.1.3 Crypto driver key element configuration

The key elements are shown in Figure 5-9 and Figure 5-10.



CryptoKeyElement_Tls_HashShaX
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_HMacShaX
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 48
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_KeyEx_Algorithm
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 12
- CryptoKeyElementSize: 1
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_KeyEx_OwnPubKey
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 10
- CryptoKeyElementSize: 64
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_KeyEx_PrivKey
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 9
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: denied
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_KeyEx_SharedSecret
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 64
- CryptoKeyElementReadAccess: interal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_MasterSecret
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 48
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfInputValue
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 110
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

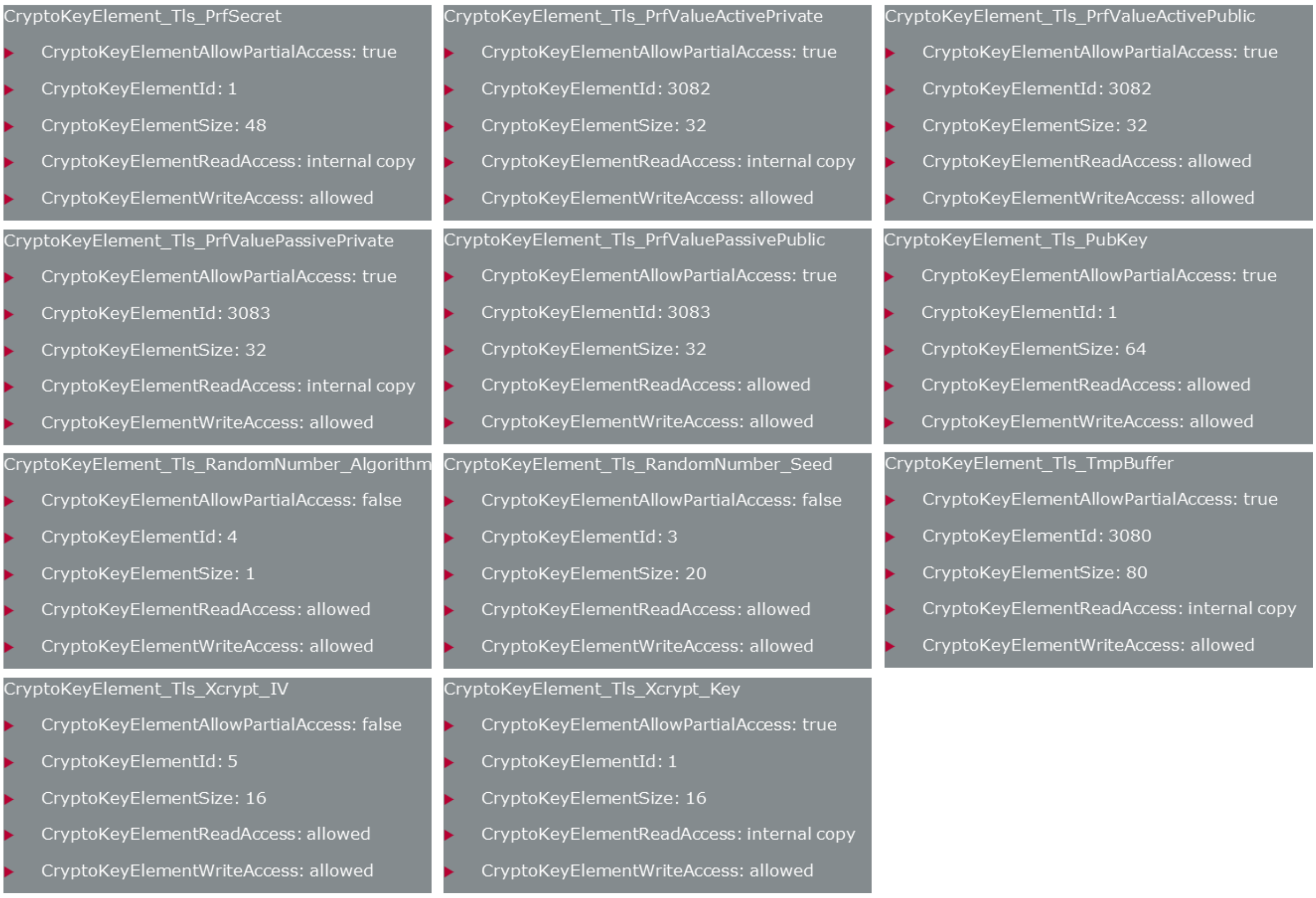
CryptoKeyElement_Tls_PrfOutputValuePrivate
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3081
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfOutputValuePublic
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3081
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfResultPrivate
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 128
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfResultPublic
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 128
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

Figure 5-9 Crypto driver key elements – part 1



CryptoKeyElement_Tls_PrfSecret
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 48
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfValueActivePrivate
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3082
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfValueActivePublic
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3082
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfValuePassivePrivate
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3083
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PrfValuePassivePublic
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3083
- CryptoKeyElementSize: 32
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_PubKey
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 64
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_RandomNumber_Algorithm
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 4
- CryptoKeyElementSize: 1
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_RandomNumber_Seed
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 3
- CryptoKeyElementSize: 20
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_TmpBuffer
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 3080
- CryptoKeyElementSize: 80
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_Xcrypt_IV
- CryptoKeyElementAllowPartialAccess: false
- CryptoKeyElementId: 5
- CryptoKeyElementSize: 16
- CryptoKeyElementReadAccess: allowed
- CryptoKeyElementWriteAccess: allowed

CryptoKeyElement_Tls_Xcrypt_Key
- CryptoKeyElementAllowPartialAccess: true
- CryptoKeyElementId: 1
- CryptoKeyElementSize: 16
- CryptoKeyElementReadAccess: internal copy
- CryptoKeyElementWriteAccess: allowed

Figure 5-10 Crypto driver key elements – part 2

#### 5.12.6.1.4 Crypto driver key types configuration

Figure 5-11 shows the mapping of `CryptoKeyTypes` to the `CryptoKeyElements`.

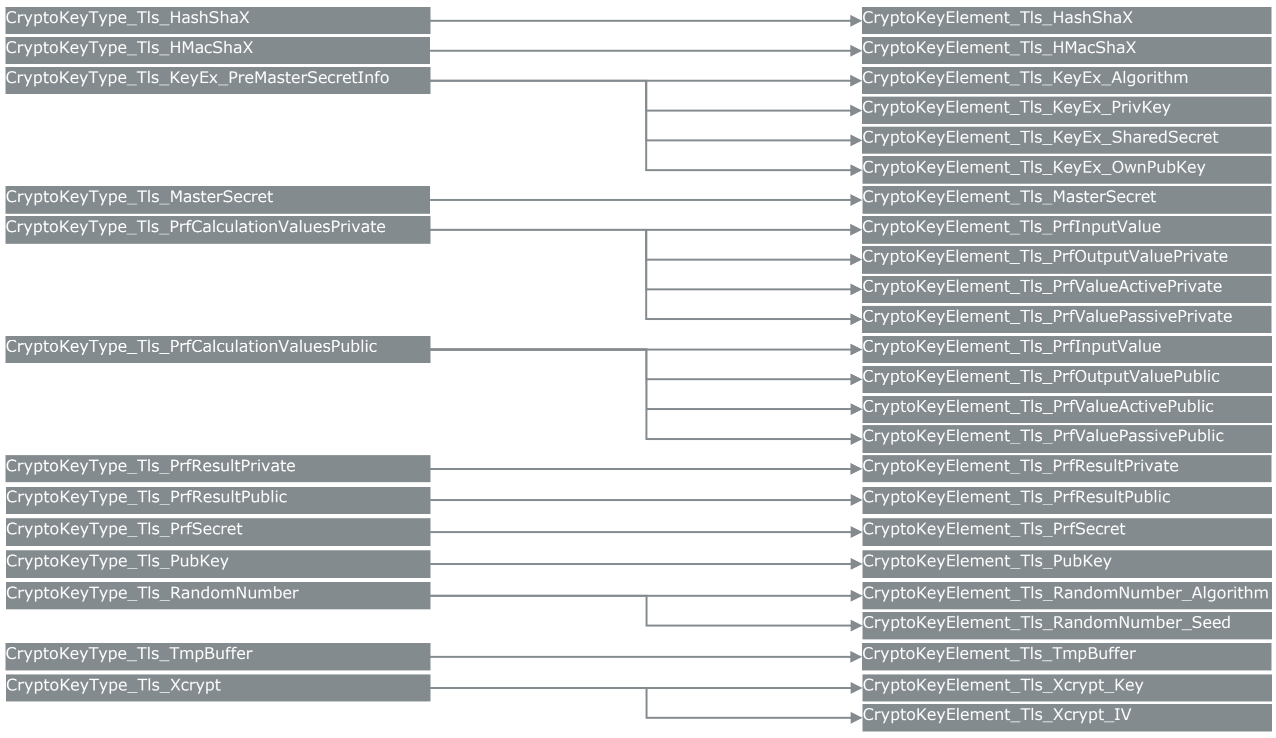

CryptoKeyType_Tls_HashShaX
CryptoKeyType_Tls_HMacShaX
CryptoKeyType_Tls_KeyEx_PreMasterSecretInfo
CryptoKeyType_Tls_MasterSecret
CryptoKeyType_Tls_PrfCalculationValuesPrivate
CryptoKeyType_Tls_PrfCalculationValuesPublic
CryptoKeyType_Tls_PrfResultPrivate
CryptoKeyType_Tls_PrfResultPublic
CryptoKeyType_Tls_PrfSecret
CryptoKeyType_Tls_PubKey
CryptoKeyType_Tls_RandomNumber
CryptoKeyType_Tls_TmpBuffer
CryptoKeyType_Tls_Xcrypt
CryptoKeyElement_Tls_HashShaX
CryptoKeyElement_Tls_HMacShaX
CryptoKeyElement_Tls_KeyEx_Algorithm
CryptoKeyElement_Tls_KeyEx_PrivKey
CryptoKeyElement_Tls_KeyEx_SharedSecret
CryptoKeyElement_Tls_KeyEx_OwnPubKey
CryptoKeyElement_Tls_MasterSecret
CryptoKeyElement_Tls_PrfInputValue
CryptoKeyElement_Tls_PrfOutputValuePrivate
CryptoKeyElement_Tls_PrfValueActivePrivate
CryptoKeyElement_Tls_PrfValuePassivePrivate
CryptoKeyElement_Tls_PrfInputValue
CryptoKeyElement_Tls_PrfOutputValuePublic
CryptoKeyElement_Tls_PrfValueActivePublic
CryptoKeyElement_Tls_PrfValuePassivePublic
CryptoKeyElement_Tls_PrfResultPrivate
CryptoKeyElement_Tls_PrfResultPublic
CryptoKeyElement_Tls_PrfSecret
CryptoKeyElement_Tls_PubKey
CryptoKeyElement_Tls_RandomNumber_Algorithm
CryptoKeyElement_Tls_RandomNumber_Seed
CryptoKeyElement_Tls_TmpBuffer
CryptoKeyElement_Tls_Xcrypt_Key
CryptoKeyElement_Tls_Xcrypt_IV

Figure 5-11 Crypto driver key types

#### 5.12.6.1.5 Crypto driver keys configuration

Create the `CryptoKeys` show in Figure 5-12 based on the previously create `CryptoKeyTypes`.

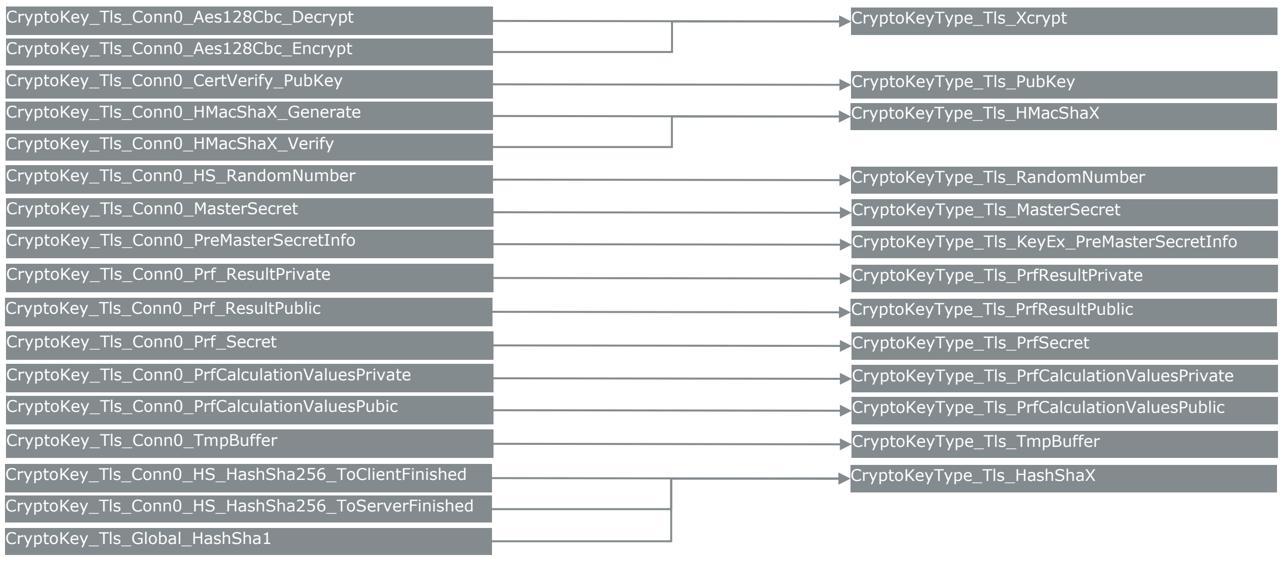

CryptoKey_Tls_Conn0_Aes128Cbc_Decrypt
CryptoKey_Tls_Conn0_Aes128Cbc_Encrypt
CryptoKey_Tls_Conn0_CertVerify_PubKey
CryptoKey_Tls_Conn0_HMacShaX_Generate
CryptoKey_Tls_Conn0_HMacShaX_Verify
CryptoKey_Tls_Conn0_HS_RandomNumber
CryptoKey_Tls_Conn0_MasterSecret
CryptoKey_Tls_Conn0_PreMasterSecretInfo
CryptoKey_Tls_Conn0_Prf_ResultPrivate
CryptoKey_Tls_Conn0_Prf_ResultPublic
CryptoKey_Tls_Conn0_Prf_Secret
CryptoKey_Tls_Conn0_PrfCalculationValuesPrivate
CryptoKey_Tls_Conn0_PrfCalculationValuesPubic
CryptoKey_Tls_Conn0_TmpBuffer
CryptoKey_Tls_Conn0_HS_HashSha256_ToClientFinished
CryptoKey_Tls_Conn0_HS_HashSha256_ToServerFinished
CryptoKey_Tls_Global_HashSha1
CryptoKeyType_Tls_Xcrypt
CryptoKeyType_Tls_PubKey
CryptoKeyType_Tls_HMacShaX
CryptoKeyType_Tls_RandomNumber
CryptoKeyType_Tls_MasterSecret
CryptoKeyType_Tls_KeyEx_PreMasterSecretInfo
CryptoKeyType_Tls_PrfResultPrivate
CryptoKeyType_Tls_PrfResultPublic
CryptoKeyType_Tls_PrfSecret
CryptoKeyType_Tls_PrfCalculationValuesPrivate
CryptoKeyType_Tls_PrfCalculationValuesPublic
CryptoKeyType_Tls_TmpBuffer
CryptoKeyType_Tls_HashShaX

Figure 5-12 Crypto driver keys

### 5.12.6.2 Crypto interface configuration

This chapter describes the CryIf module configuration.

It is required to enable the parameter `CryIfRedirection`.

#### 5.12.6.2.1 CryIf crypto modules configuration

Create a CryIfCryptoModule for the used crypto driver. It is required to enable the parameter `CryIfSupportsKeyElementCopyPartial` for TLS.

#### 5.12.6.2.2 CryIf channels configuration

Create one `CryIfChannel` element for each `CryptoDriverObject` listed in Chapter 5.12.6.1.1 Figure 5-8 so that the mapping shown in 5.12.6.3.1 Figure 5-13 can be created.

#### 5.12.6.2.3 CryIf keys configuration

It is not required to create the `CryIfKeys` manually, instead it is recommended to create the Crypto driver keys and use the 'propagate' solving action of the CSM. Because the elements will just be forwarded (see 5.12.6.3.2 Figure 5-14).

### 5.12.6.3 Crypto Service Manager configuration

This chapter shows the required Csm configuration to run TLS.

#### 5.12.6.3.1 CSM queue configuration

The following `CsmQueue` elements need to be created and mapped to the `CryptoDriverObjects` using `CryIfChannels` as shown in Figure 5-13.

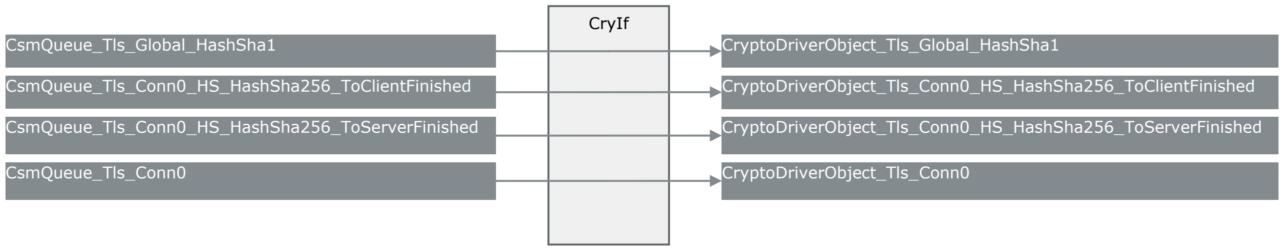

CryIf
CsmQueue_Tls_Global_HashSha1
CsmQueue_Tls_Conn0_HS_HashSha256_ToClientFinished
CsmQueue_Tls_Conn0_HS_HashSha256_ToServerFinished
CsmQueue_Tls_Conn0
CryptoDriverObject_Tls_Global_HashSha1
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToClientFinished
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToServerFinished
CryptoDriverObject_Tls_Conn0

Figure 5-13 Csm to Crypto driver object mapping

#### 5.12.6.3.2 CSM keys configuration

It is not required to create the `CsmKeys` manually, instead it is recommended to create the Crypto driver keys and use the 'propagate' solving action of the CSM.

After creating the `CsmKeys` automatically the configurations should look like shown in Figure 5-14.

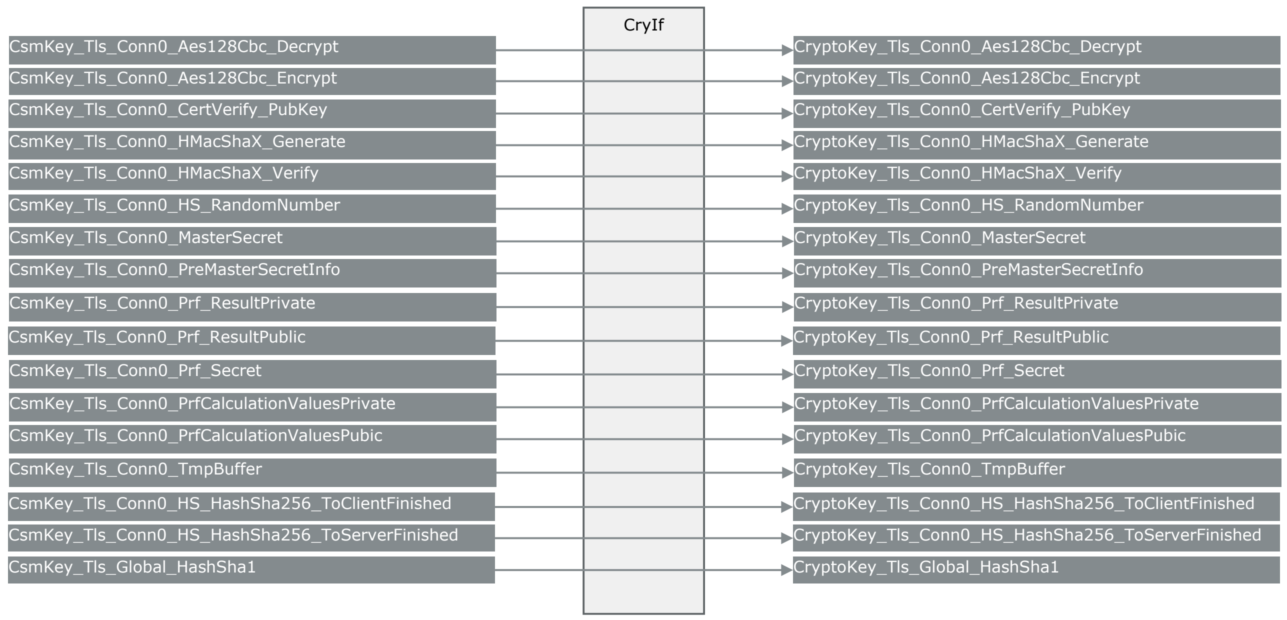

CryIf
CsmKey_Tls_Conn0_Aes128Cbc_Decrypt
CsmKey_Tls_Conn0_Aes128Cbc_Encrypt
CsmKey_Tls_Conn0_CertVerify_PubKey
CsmKey_Tls_Conn0_HMacShaX_Generate
CsmKey_Tls_Conn0_HMacShaX_Verify
CsmKey_Tls_Conn0_HS_RandomNumber
CsmKey_Tls_Conn0_MasterSecret
CsmKey_Tls_Conn0_PreMasterSecretInfo
CsmKey_Tls_Conn0_Prf_ResultPrivate
CsmKey_Tls_Conn0_Prf_ResultPublic
CsmKey_Tls_Conn0_Prf_Secret
CsmKey_Tls_Conn0_PrfCalculationValuesPrivate
CsmKey_Tls_Conn0_PrfCalculationValuesPubic
CsmKey_Tls_Conn0_TmpBuffer
CsmKey_Tls_Conn0_HS_HashSha256_ToClientFinished
CsmKey_Tls_Conn0_HS_HashSha256_ToServerFinished
CsmKey_Tls_Global_HashSha1
CryptoKey_Tls_Conn0_Aes128Cbc_Decrypt
CryptoKey_Tls_Conn0_Aes128Cbc_Encrypt
CryptoKey_Tls_Conn0_CertVerify_PubKey
CryptoKey_Tls_Conn0_HMacShaX_Generate
CryptoKey_Tls_Conn0_HMacShaX_Verify
CryptoKey_Tls_Conn0_HS_RandomNumber
CryptoKey_Tls_Conn0_MasterSecret
CryptoKey_Tls_Conn0_PreMasterSecretInfo
CryptoKey_Tls_Conn0_Prf_ResultPrivate
CryptoKey_Tls_Conn0_Prf_ResultPublic
CryptoKey_Tls_Conn0_Prf_Secret
CryptoKey_Tls_Conn0_PrfCalculationValuesPrivate
CryptoKey_Tls_Conn0_PrfCalculationValuesPubic
CryptoKey_Tls_Conn0_TmpBuffer
CryptoKey_Tls_Conn0_HS_HashSha256_ToClientFinished
CryptoKey_Tls_Conn0_HS_HashSha256_ToServerFinished
CryptoKey_Tls_Global_HashSha1

Figure 5-14 Csm to Crypto driver keys mapping

#### 5.12.6.3.3 CSM primitive configuration

The `CsmPrimitives` listed in Figure 5-15 needs to be created.

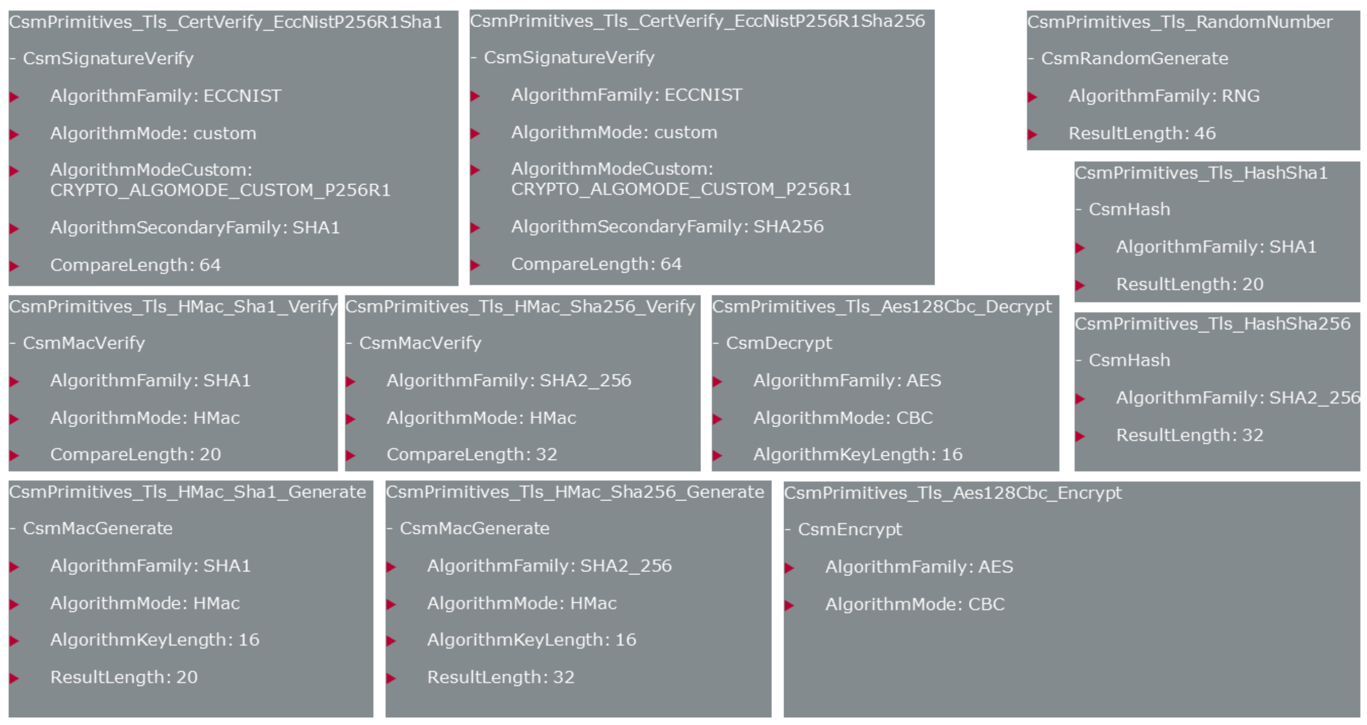


Figure 5-15 Csm primitives

The `CsmPrimitives` in Figure 5-15 require the `CryptoDriverObject` configurations as shown in chapter 5.12.6.1.1 Figure 5-8. There is no direct mapping from the `CsmPrimitives` to the `CryptoDriverObject` primitives. The relation is built via the `CsmJob` and `CsmQueue` elements. For a better overview the relation is shown in Figure 5-16.

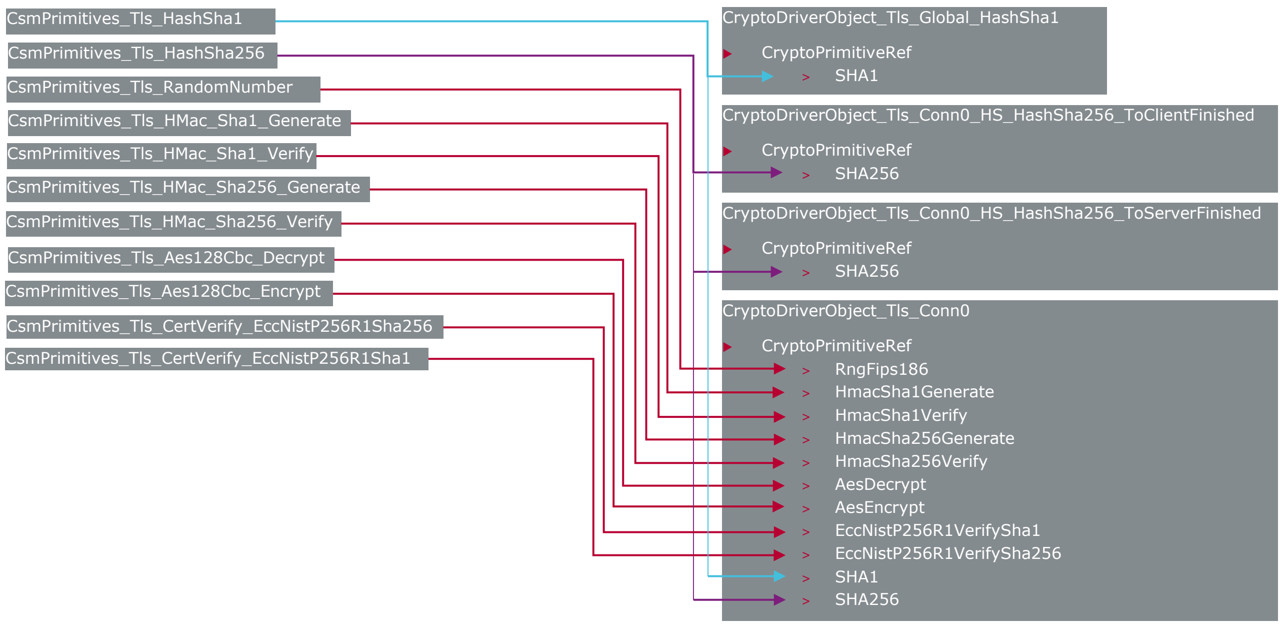

CsmPrimitives_Tls_HashSha1
CsmPrimitives_Tls_HashSha256
CsmPrimitives_Tls_RandomNumber
CsmPrimitives_Tls_HMac_Sha1_Generate
CsmPrimitives_Tls_HMac_Sha1_Verify
CsmPrimitives_Tls_HMac_Sha256_Generate
CsmPrimitives_Tls_HMac_Sha256_Verify
CsmPrimitives_Tls_Aes128Cbc_Decrypt
CsmPrimitives_Tls_Aes128Cbc_Encrypt
CsmPrimitives_Tls_CertVerify_EccNistP256R1Sha256
CsmPrimitives_Tls_CertVerify_EccNistP256R1Sha1
CryptoDriverObject_Tls_Global_HashSha1
CryptoPrimitiveRef
SHA1
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToClientFinished
CryptoPrimitiveRef
SHA256
CryptoDriverObject_Tls_Conn0_HS_HashSha256_ToServerFinished
CryptoPrimitiveRef
SHA256
CryptoDriverObject_Tls_Conn0
CryptoPrimitiveRef
RngFips186
HmacSha1Generate
HmacSha1Verify
HmacSha256Generate
HmacSha256Verify
AesDecrypt
AesEncrypt
EccNistP256R1VerifySha1
EccNistP256R1VerifySha256
SHA1
SHA256

Figure 5-16 Csm primitive requirements for Crpyto driver objects

#### 5.12.6.3.4 CSM redirection configuration

TLS requires the `CsmInOutRedirection` configurations shown in Figure 5-17.

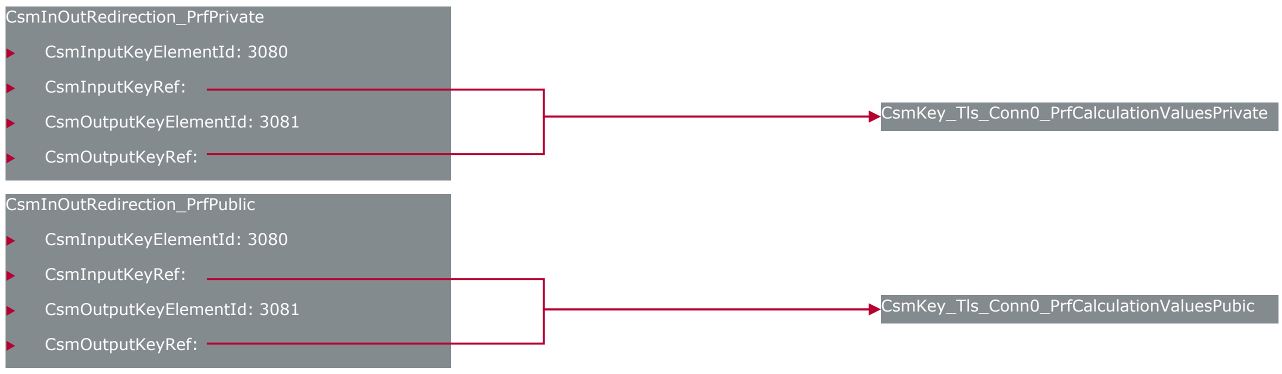

CsmInOutRedirection_PrfPrivate
CsmInputKeyElementId: 3080
CsmInputKeyRef:
CsmOutputKeyElementId: 3081
CsmOutputKeyRef:
CsmKey_Tls_Conn0_PrfCalculationValuesPrivate
CsmInOutRedirection_PrfPublic
CsmInputKeyElementId: 3080
CsmInputKeyRef:
CsmOutputKeyElementId: 3081
CsmOutputKeyRef:
CsmKey_Tls_Conn0_PrfCalculationValuesPubic

Figure 5-17 Csm redirection

#### 5.12.6.3.5 CSM job configuration

Create the `CsmJobs` and mappings as shown in Figure 5-18.

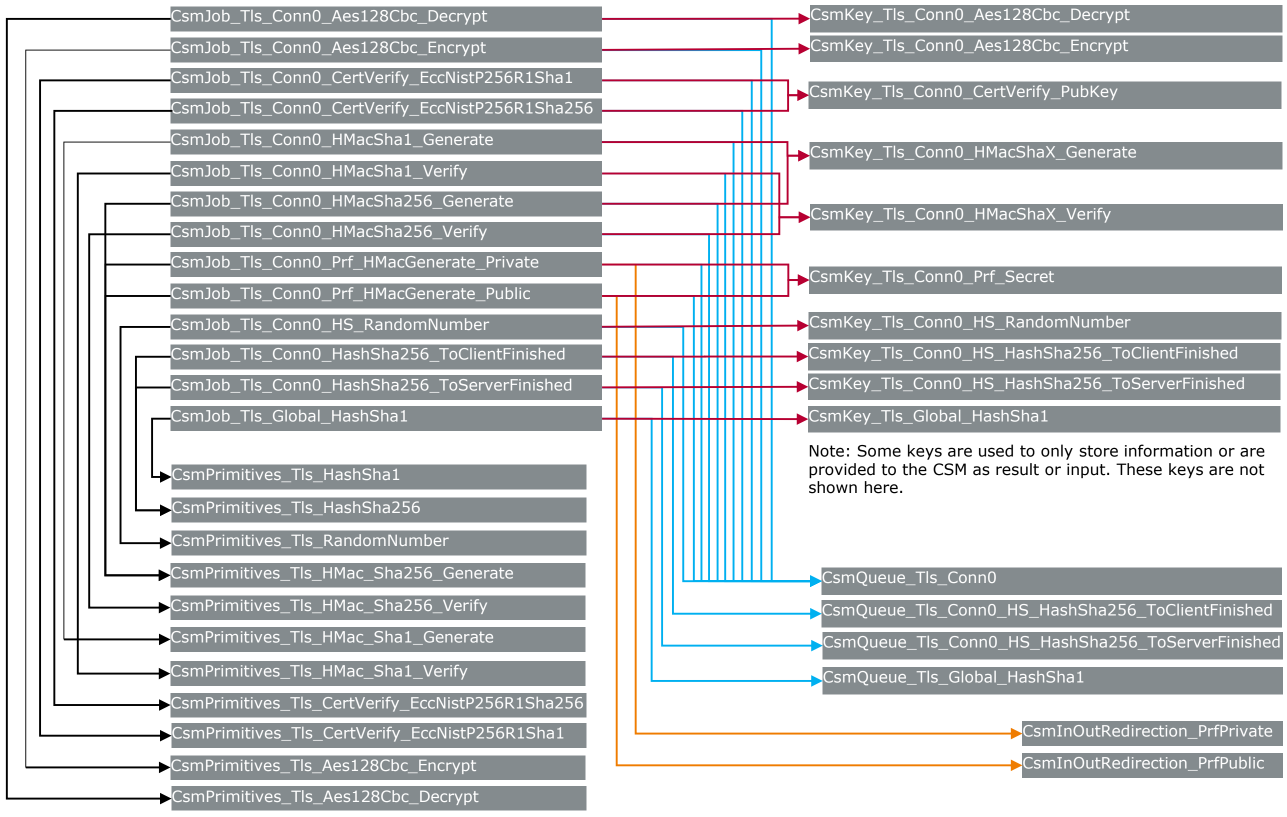

CsmJob_Tls_Conn0_Aes128Cbc_Decrypt
CsmJob_Tls_Conn0_Aes128Cbc_Encrypt
CsmJob_Tls_Conn0_CertVerify_EccNistP256R1Sha1
CsmJob_Tls_Conn0_CertVerify_EccNistP256R1Sha256
CsmJob_Tls_Conn0_HMacSha1_Generate
CsmJob_Tls_Conn0_HMacSha1_Verify
CsmJob_Tls_Conn0_HMacSha256_Generate
CsmJob_Tls_Conn0_HMacSha256_Verify
CsmJob_Tls_Conn0_Prf_HMacGenerate_Private
CsmJob_Tls_Conn0_Prf_HMacGenerate_Public
CsmJob_Tls_Conn0_HS_RandomNumber
CsmJob_Tls_Conn0_HashSha256_ToClientFinished
CsmJob_Tls_Conn0_HashSha256_ToServerFinished
CsmJob_Tls_Global_HashSha1
CsmPrimitives_Tls_HashSha1
CsmPrimitives_Tls_HashSha256
CsmPrimitives_Tls_RandomNumber
CsmPrimitives_Tls_HMac_Sha256_Generate
CsmPrimitives_Tls_HMac_Sha256_Verify
CsmPrimitives_Tls_HMac_Sha1_Generate
CsmPrimitives_Tls_HMac_Sha1_Verify
CsmPrimitives_Tls_CertVerify_EccNistP256R1Sha256
CsmPrimitives_Tls_CertVerify_EccNistP256R1Sha1
CsmPrimitives_Tls_Aes128Cbc_Encrypt
CsmPrimitives_Tls_Aes128Cbc_Decrypt
CsmKey_Tls_Conn0_Aes128Cbc_Decrypt
CsmKey_Tls_Conn0_Aes128Cbc_Encrypt
CsmKey_Tls_Conn0_CertVerify_PubKey
CsmKey_Tls_Conn0_HMacShaX_Generate
CsmKey_Tls_Conn0_HMacShaX_Verify
CsmKey_Tls_Conn0_Prf_Secret
CsmKey_Tls_Conn0_HS_RandomNumber
CsmKey_Tls_Conn0_HS_HashSha256_ToClientFinished
CsmKey_Tls_Conn0_HS_HashSha256_ToServerFinished
CsmKey_Tls_Global_HashSha1
Note: Some keys are used to only store information or are provided to the CSM as result or input. These keys are not shown here.
CsmQueue_Tls_Conn0
CsmQueue_Tls_Conn0_HS_HashSha256_ToClientFinished
CsmQueue_Tls_Conn0_HS_HashSha256_ToServerFinished
CsmQueue_Tls_Global_HashSha1
CsmInOutRedirection_PrfPrivate
CsmInOutRedirection_PrfPublic

Figure 5-18 Csm job mappings

### 5.12.7 TLS Startup

For the proper operation of the TLS component, additional initializations are required from the user. This includes seeding the random number generator, loading the certificate and private keys (only necessary for ECC cipher suites) and loading the PSK keys into the corresponding CSM keys.

#### 5.12.7.1 Additional initializations – Random Generator

The random generator, which is used during the TLS handshake for generating the random values contained inside the ‘*Hello_Messages*’, and also during the exchange of application data to generate the ‘IV’ of the AES operation, must be seeded and initialized before the first TLS connection will be established. The following source code example shows the necessary steps to do so. The parameter `RandKeyId` therefore must match with the configured TLS random generator job, which is represented by the ECUC parameter: `TcpIpTlsCsmRandomGenerateJobRef`.

```
/**********************************************************************************************
 * Tls_RandomSeedInit()
 *********************************************************************************************/
/*! \brief       Initialization function for random generator
 *  \details     Service to initialize the modules random generator used by the TLS.
 *  \param[in]   RandKeyId           CSM KeyId for the configured random generator.
 *  \return      E_OK                if initialization is successful.
 *               E_NOT_OK            error during initialization
 *********************************************************************************************/
Std_ReturnType Tls_RandomSeedInit(uint32  RandKeyId) {

  /* ----- Local Variables ---------------------------------------------- */
  uint8 rndSeed[TLS_CRYPTO_SEED_LEN];
  Std_ReturnType retVal = E_NOT_OK;
  /* ----- Implementation ----------------------------------------------- */
  /* Fill the seed array with randomness */
  Appl_Crypto_GetRandArray(&rndSeed[0], TLS_CRYPTO_SEED_LEN);

  /* Set the key element for the random function generator valid */
  if ((Csm_KeyElementSet(RandKeyId, CRYPTO_KE_RANDOM_SEED_STATE, rndSeed, TLS_CRYPTO_SEED_LEN) ==
       E_OK)
    && (Csm_KeySetValid(RandKeyId) == E_OK))
  {
    retVal = E_OK;
  }

  return retVal;
} /* Tls_RandomSeedInit() */
```

**Caution**
If the random generator is not properly loaded, no TLS connection can be established, and the TLS handshake will fail.

### 5.12.7.2 Additional initializations – Loading the Certificate

To successfully perform an TLS handshake, the used Server / Client certificates must be stored or loaded in the corresponding KeyM Certificate Element before the connection is getting used. There are several ways to archive this, depending on the configuration of the KeyM. For loading the certificates during runtime of the ECU, the TLS provides the TcpIp_Tls_ServiceChainCertificate() API. See chapter 4.2.60 TcpIp_Tls_ServiceChainCertificate for further information.

Refer to the KeyM Technical Reference, if the certificate should be stored statically via configuration and 'KeyMCertInit' value.

The User must also ensure, that the corresponding private key of the installed Server / Client Leaf certificate is loaded and available before the connection is getting used. The TLS-Server Leaf certificate key is used during runtime to calculate the signature (depending on the selected cipher suite) of the ephemeral public key. The TLS-Client Leaf certificate key is used during runtime to calculate the signature of the CertificateVerify message. Loading the keys can be archived by storing the private key in the key element 'CRYPTO_KE_SIGNATURE_KEY' of the corresponding CSM key, via the provided CSM Csm_KeyElementSet() API.

### 5.12.7.3 Additional initializations – Loading the PSK keys

If the TLS handshake should be performed with an PSK cipher suite, the MICROSAR Classic TCPIP uses the configured CSM key to calculate the master secret and all necessary cryptographical keys during the handshake. To successfully perform the TLS handshake with a PSK cipher suite, the user must load the corresponding PSK key into the CSM. Therefore, each individual configured PSK Identity of each configured TLS connection must be loaded. The following source code example shows the necessary steps to do so. The parameter `KeyElementId` refers to the matching crypto key element id to the configured CSM Key, which is represented by the ECUC parameter `CryptoKeyElementId`. The parameter '*KeyId*' refers to the configured CSM key for this specific PSK identity, which is represented by the ECUC parameter `TcpIpTlsPresharedKeyCsmKeyRef`. The parameter `KeyDataPtr` contains the actual PSK Key as byte array. The length of the given PSK key is presented by the parameter `KeySize`.

**Note**
The Length of the PSK Key MUST match with the configured size of the Crypto Key Element Size. Otherwise, the TLS handshake will fail.

```
/**********************************************************************************************
 * Tls_LoadPresharedKey()
**********************************************************************************************/
/*! \brief      Initialization function for random generator
 *  \details    Service to initialize the modules random generator used by the TLS.
 *  \param[in]   KeyElementId        Crypto Key ElelmentId matching to the configured CSM Key
 *  \param[in]   KeyId               CSM Key Id for this configured PSK Identity
 *  \param[in]   KeyDataPtr          Pointer to the PSK Key
 *  \param[in]   KeySize             Length of the PSK Key
 *  \return      E_OK                if initialization is successful.
 *               E_NOT_OK            error during initialization
*****************************************************************************************************/
FUNC(Std_ReturnType, TCPIP_CODE) Tls_LoadPresharedKey(
  uint32                  KeyElementId,
  uint32                  KeyId,
  TCPIP_P2C(uint8)        KeyDataPtr,
  CONST(uint8, AUTOMATIC) KeySize)
{
  /* ----- Local Variables ------------------------------------------- */
  Std_ReturnType retVal = E_NOT_OK;

  /* -----Implementation--------------------------------------------- */
  if ((Csm_KeyElementSet(KeyId, KeyElementId, KeyDataPtr, KeySize) == E_OK)
    && (Csm_KeySetValid(KeyId) == E_OK))
  {
    retVal = E_OK;
  }
  return retVal;
} /* Tls_LoadPresharedKey() */
```

**Caution**
If the PSK key is not properly loaded, no TLS connection can be established, and the TLS handshake will fail.

### 5.12.8 TLS Usage

#### 5.12.8.1 TLS Connection Callout Notification

The successful TLS connection establishment is indicated by the callout event notification with the event type 'TCPIP_TLS_HANDSHAKE_SUCCEEDED. This callout is triggered additionally after the TLS connections is established and ready to communicate. This callout is identical to the TcpIp callouts TcpIp_TcpConnected() and TcpIp_TcpAccepted().

#### 5.12.8.2 TLS Connection Setup – Client

Figure 5-19 gives an overview of the connection setup of an TLS client. For detailed information on the connection setup see the Specification of TCP/IP Stack [1].

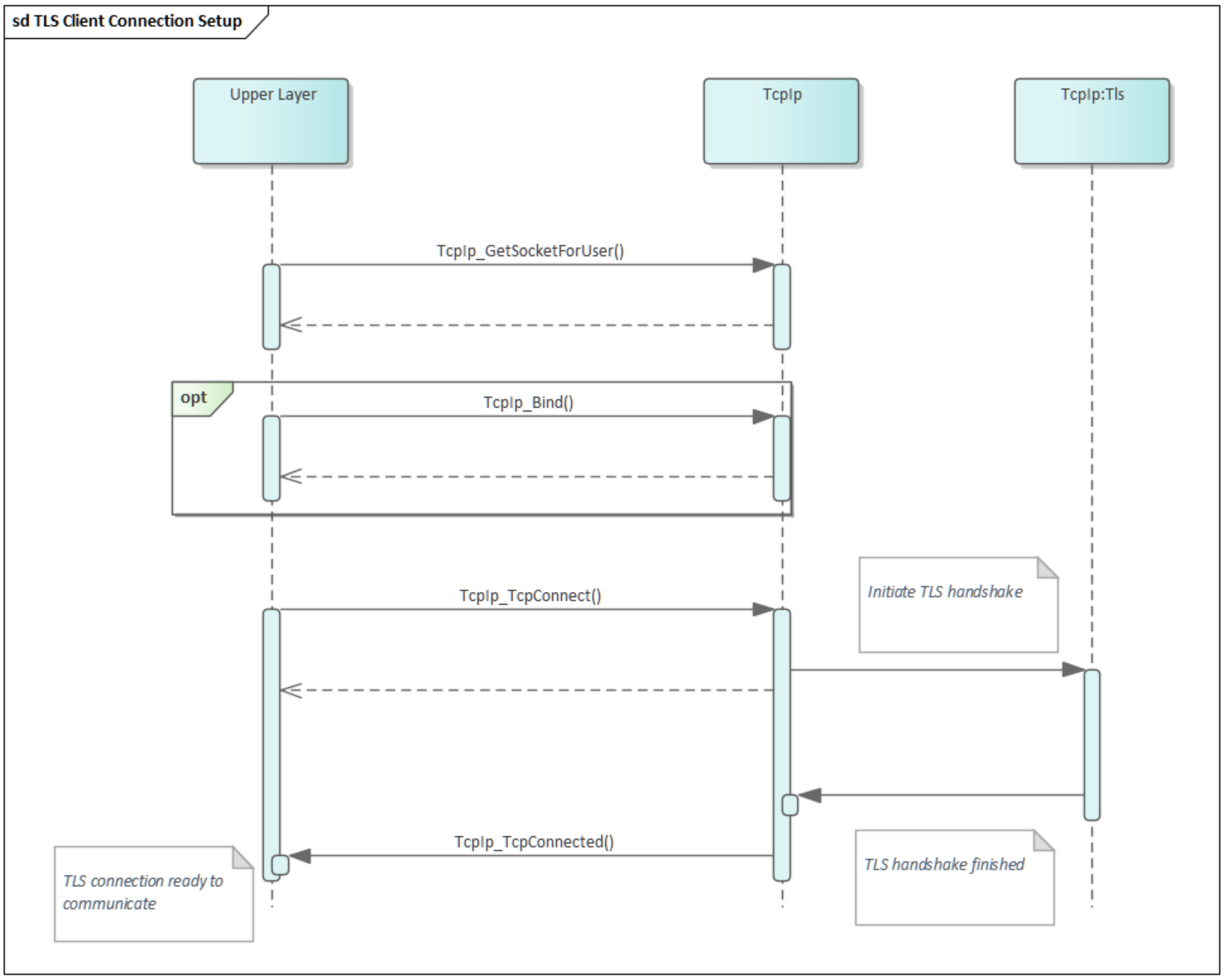


Figure 5-19 TLS Connection Setup – Client overview.

### 5.12.8.3 TLS Validation Result Callout

The TLS Validation Result callout is invoked from the TLS Client during the TLS handshake to inform the UpperLayer about the validation status of the received certificate chain. The UpperLayer can then decide if the TLS handshake should be aborted (e.g., when the certificate status is not sufficient) or the TLS connection should be successfully established. The Validation result also contains information about the received OCSP status and the updated validation status of the OCSP result. The definition of the callout function is described in chapter 4.5.1.13 <Up_TlsValidationResultCallout>.

### 5.12.8.4 TLS Connection Setup – Server

Figure 5-20 gives an overview of the connection setup of an TLS client. For detailed information on the connection setup see the Specification of TCP/IP Stack [1].

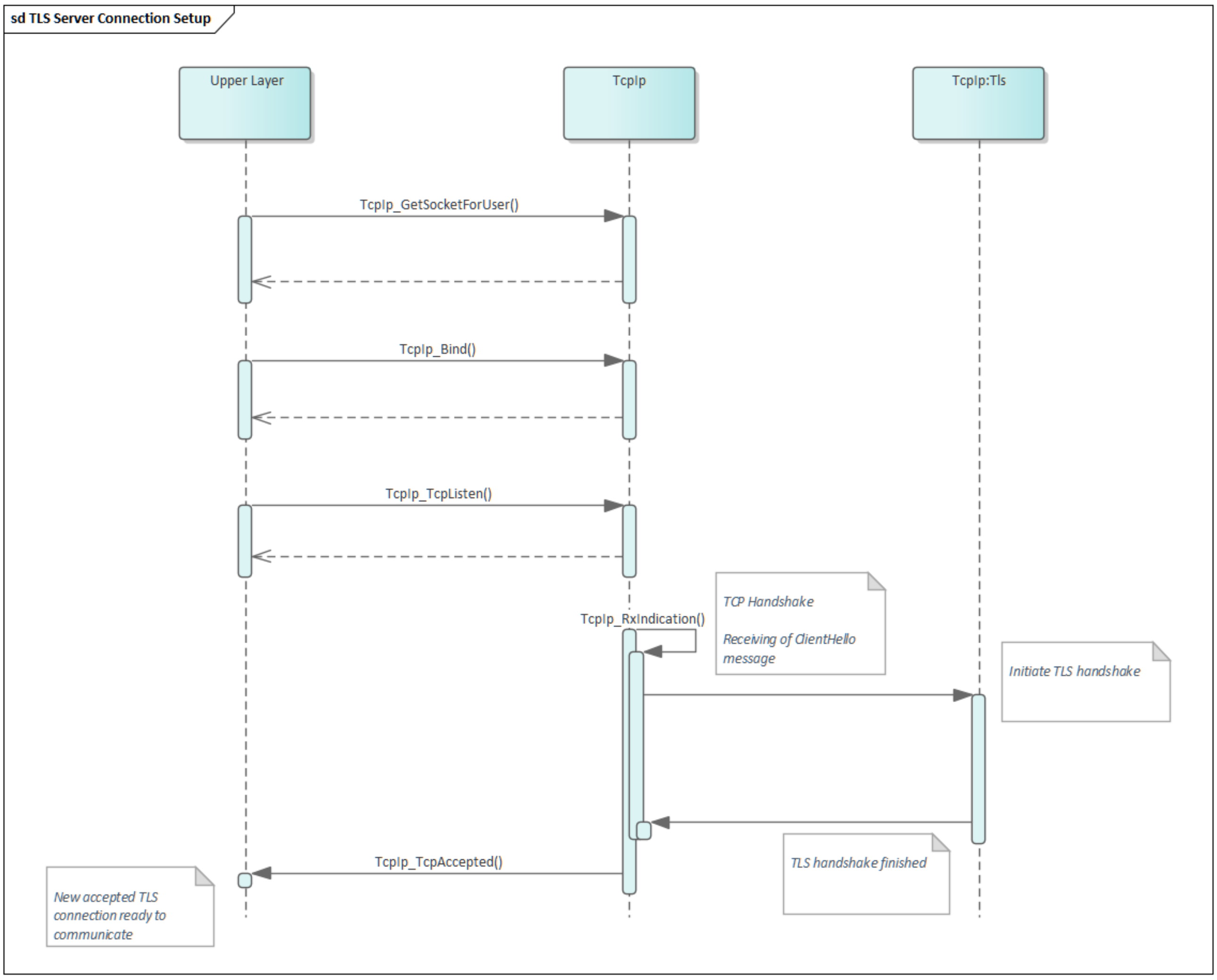


Figure 5-20 TLS Connection Setup – Server overview.

### 5.12.8.5 Generation customizations

The TcpIp module uses the ComStackLib to generate the configuration dependent data of the module. The ComStackLib generation behavior can be customized with the parameters of container `/TcpIp/TcpIpGeneral/TcpIpGeneration`. Please refer to the parameter description and to the TechnicalReference CommonAsr_ComStackLib [7] for more information. Please note that there are customizations which are only valid for a specific quality level (e.g. during the development phase).

### 5.12.9 Dependencies to Other Modules

#### 5.12.9.1 TLS and HSM Configuration Dependencies

If an HSM is used, the HSM driver object buffer size depends upon the used TLS buffer size and (if configured) the TLS record size limit (see chapter TLS Max Fragment Length). The driver object buffer size shared by the HSM does not need to be bigger than the record size limit. If any problems regarding encryption or decryption occur and an HSM is used, please check if the HSM driver object buffer size is configured correctly.

## 5.13 Multicore Support Configuration

TcpIp can be configured to function on multicore systems. To enable multicore support, `/TcpIp/TcpIpGeneral/TcpIpMultiCoreSupportEnabled` must be set. Additionally, each `/TcpIp/TcpIpConfig/TcpIpCtrl` must be mapped to an `EcucPartition` that has been configured for a specific core. This is done by configuring the `TcpIp/TcpIpConfig/TcpIpCtrl/TcpIpCtrlEcuPartition`.

**Note**
The mapping of `TcpIpCtrl` to `EcuPartition` is n:1, implying one or more controllers can be configured to execute on a single core.

# 6 Security Information

The intention of this chapter is to identify possible configuration-dependent security issues which can arise from using the IP-based protocols provided by the TcpIp module. All mentioned vulnerabilities and possible hardening measures refer to the TcpIp module. However, it is recommended to look at the whole system or network architecture to develop a good security solution. Please note that this list does not thrive for completeness. The user of MICROSAR Classic TCPIP is responsible for the customer-specific configuration of this module.

## 6.1 ARP

| **Vulnerability** |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpIpV4General/TcpIpArpEnabled` is set to `TRUE` and a dynamic ARP table is configured an attacker could inject forged table entries (known as ARP spoofing). |
| **Attack class** |
| An attacker could perform a man in the middle attack. |
| **Hardening mechanism** |
| Use a static ARP table or a combination of dynamic and static ARP tables instead of only using a dynamic ARP table. |

Table 6-1 TcpIpArpEnabled, ARP spoofing

| **Vulnerability** |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpIpV4General/TcpIpArpEnabled` is set to `TRUE` and a dynamic ARP table is used an attacker could flood the table with forged table entries (known as ARP flooding). |
| **Attack class** |
| This could lead to time lags on the communication channel. |
| **Hardening mechanism** |
| Use a static ARP table or a combination of dynamic and static ARP tables instead of only using a dynamic ARP table. |

Table 6-2 TcpIpArpEnabled, ARP flooding

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpIpV4General/TcpIpArpDiscardedEntryHandling` is set to `TRUE` new ARP responses are discarded if the ARP table is full. |
| **Attack class** |
| An attacker could fill the ARP table with forged table entries and block the table for valid incoming ARP responses, which leads to a denial of service. |
| **Hardening mechanism** |
| Use a static ARP table or a combination of dynamic static ARP tables instead of only using a dynamic ARP table. Configure a reasonable timeout for the ARP entries in a dynamic ARP table. |

Table 6-3 TcpIpArpDiscardedEntryHandling

## 6.2 ICMPv4

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpIpV4General/TcpIpIcmpEnabled` is set to `TRUE` the TcpIp stack can process and send ICMP messages (e.g. echo response). |
| **Attack class** |
| ICMP messages could reveal useful information to an attacker e.g. with an ICMP echo request an attacker could enumerate which hosts are available in a network. |
| **Hardening mechanism** |
| Use an additional system like a firewall to block certain packets or an intrusion detection system to detect anomalies. In addition, it is possible to deactivate ICMPv4 if not necessary. |

Table 6-4 TcpIpIcmpEnabled

## 6.3 IPv4

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpIpV4General/TcpIpIpV4Enabled` is set to `TRUE` IP spoofing is possible. That means an attacker could pretend to be someone else. |
| **Attack class** |
| The authenticity of IP packets could be threatened. |
| **Hardening mechanism** |
| An additional security protocol (e.g. Internet Protocol Security (IPsec)) can ensure the authenticity of IP packets. Furthermore, a well-grounded network architecture (e.g. use virtual LANs) and the use of a firewall or an intrusion detection system can harden the network. |

Table 6-5 TcpIpIpV4Enabled

## 6.4 IPv6

| Vulnerability |
|---|
| In case `TcpIp/TcpIpGeneral/TcpIpIpV6General/TcpIpIpV6Enabled` is set to `TRUE` IP spoofing is possible. That means an attacker could pretend to be someone else. |

| Attack class |
|---|
| The authenticity of IP packets could be threatened. |
| **Hardening mechanism** |
| An additional security protocol (e.g. Internet Protocol Security (IPsec)) can ensure the authenticity of IP packets. Furthermore, a well-grounded network architecture (e.g. use virtual LANs) and the use of a firewall or an intrusion detection system can harden the network. |

Table 6-6 TcpIpIpV6Enabled

## 6.5 TCP

| Vulnerability |
|---|
| In case `TcpIp/TcpIpGeneral/TcpIpTcpEnabled` is set to `TRUE` an attacker could use port scan techniques based on the TCP flags. |
| **Attack class** |
| This port scanning techniques could reveal useful information about open ports on a system. |
| **Hardening mechanism** |
| The use of a firewall<br>and / or<br>setting the configuration parameter `TcpIp/TcpIpConfig/TcpIpTcpConfig/TcpIpTcpNoRstInvSocket` to `TRUE`<br>makes port scanning more difficult. |

Table 6-7 TcpIpTcpEnabled, port scan

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpTcpEnabled` is set to `TRUE` an attacker could read and manipulate the communication between two hosts. |
| **Attack class** |
| An attacker could perform a man in the middle attack. |
| **Hardening mechanism** |
| Use an additional security protocol (e.g. Transport Layer Security (TLS)) for authentication and data encryption. |

Table 6-8 TcpIpTcpEnabled, man in the middle

## 6.6 UDP

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpUdpEnabled` is set to `TRUE` an attacker could read and manipulate the communication between two hosts. |
| **Attack class** |
| An attacker could perform a man in the middle attack. |
| **Hardening mechanism** |
| Use an additional security protocol (e.g. Internet Protocol Security (IPsec)) for authentication and data encryption. |

Table 6-9 TcpIpUdpEnabled

## 6.7 DHCPv4 Server

| Vulnerability |
|---|
| If `TcpIp/TcpIpGeneral/TcpIpDhcpServerEnabled` is set to `TRUE` an attacker could consume all available IP addresses in the configured IP range (known as DHCP starvation attack). |
| **Attack class** |
| If all IP addresses were consumed by an attacker a new client is not able to receive an IP address, which leads to a denial of service. |
| **Hardening mechanism** |
| An intrusion detection system could be used to detect anomalies. |

Table 6-10 TcpIpDhcpServerEnabled

## 6.8 Reporting of detected Security Events

The TcpIp module supports reporting of security events to the intrusion detection system manager (IdsM) as specified by AUTOSAR.

The reporting can be enabled and configured by the following parameters:

> `/TcpIp/TcpIpGeneral/TcpIpEnableSecurityEventReporting`

> `/TcpIp/TcpIpGeneral/TcpIpSecurityEventRefs`

And supports the following specified events:

| IdsM Event | Short Description |
|---|---|
| `TCPIP_SEV_ARP_IP_ADDR_CONFLICT` | Received local IP address in ARP reply for different MAC. |
| `TCPIP_SEV_DROP_INV_IPV4_ADDR` | Dropped datagram because of invalid IPV4 address. |
| `TCPIP_SEV_DROP_INV_IPV6_ADDR` | Dropped datagram because of invalid IPV6 address. |
| `TCPIP_SEV_DROP_INV_PORT_TCP` | Dropped TCP packet because of invalid destination TCP-Port. |
| `TCPIP_SEV_DROP_INV_PORT_UDP` | Dropped UDP packet because of invalid destination UDP-Port. |

Table 6-11 Supported security events for reporting

# 7 Glossary and Abbreviations

## 7.1 Glossary

| Term | Description |
|---|---|
| | |

Table 7-1 Glossary

## 7.2 Abbreviations

| Abbreviation | Description |
|---|---|
| AH | Authentication Header |
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BSW | Basis Software |
| CSM | Crypto Services Module |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| EAD | Embedded Architecture Designer |
| ECU | Electronic Control Unit |
| ESP | Encapsulated Security Payload |
| ISR | Interrupt Service Routine |
| IPsec | Internet Protocol Security |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| SA | Security Association |
| SAD | Security Association Database |
| SPD | Security Policy Database |
| SRS | Software Requirement Specification |
| SWS | Software Specification |

Table 7-2 Abbreviations

# 8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com