VECTOR

MICROSAR TLS

Technical Reference

Transport Layer Security

Version 5.01.00

Authors	Thorsten Albers
Status	Released

Document Information

History

Author	Date	Version	Remarks
Thorsten Albers	2010-10-28	1.0	Creation of document
Thorsten Albers	2011-01-12	1.1	Tls_Shutdown added
Thorsten Albers	2011-03-29	1.2	More detailed descriptions
Thorsten Albers	2011-08-26	1.3	ECC support included
Thorsten Albers	2011-10-27	1.4	minor updates
Thorsten Albers	2012-08-06	1.5	TLS 1.2 support added, cipher suites with SHA256 added
Fabian Eisele	2015-05-27	2.00.00	Added MSR4R12 parts
Thorsten Albers	2015-11-03	2.00.01	Conversion to new template, update of API descriptions, add some configuration parameter descriptions
Thorsten Albers	2016-03-01	2.01.00	Moved crypto calculation for handshake into separate MainFunction. Layout changed for new Vector logo.
Thorsten Albers	2016-03-07	3.00.00	FEAT-1347: TLS as TcpIp plug-in
Thorsten Albers	2016-04-20	3.01.00	FEAT-1741: Support TLS cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
Thorsten Albers	2016-05-23	3.01.01	FEAT-1741: Support server certificate validation using Certificate Revocation Lists (CRL)
Thorsten Albers	2016-10-27	4.00.00	FEAT-2005: Release of TLS_RSA_WITH_AES_128_CBC_SHA256, also support TLS1.2 with RSA and client certificate FEAT-1996: TLS Heartbeat Extension
Thorsten Albers	2017-02-28	4.00.01	Review integration
Thorsten Albers	2017-03-29	5.00.00	FEAT-2275: Release of TLS
Thorsten Albers	2017-05-16	5.01.00	STORYC-592: TLS Server CertChain Callout in TLS Client STORYC-638: Report Invalid Server CertChain STORYC-1506: Grant access to the TLS master secret

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	4.2.1
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

[4]	AUTOSAR	AUTOSAR_SWS_SocketAdaptor.pdf	4.0.3
[5]	AUTOSAR	AUTOSAR_SWS_TcpIp.pdf	4.2.1
[6]	IETF	RFC 793: TCP	-
[7]	IETF	RFC 768: UDP	-
[8]	IETF	RFC 2246: TLS 1.0	-
[9]	IETF	RFC 5246: TLS 1.2	-
[10]	IETF	RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	-
[11]	IETF	RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions	-
[12]	IETF	RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP	-

Scope of the Document

This technical reference describes the general use of the TLS basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler) 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 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.01.xx	Tool based configuration, ASR and BSD APIs
2.00.xx	Reduction of function local crypto workspaces, support for ECC cipher suites
3.00.xx	Support of Ipv6 and Ipv4 address format
3.01.xx	Support for certificates with ECDSA-with-SHA256 signature
3.02.xx	Support of TLS 1.2 added
3.03.xx	Support Pass-Through Mode
3.04.xx	Support reading the NVM Block ID of the used root certificate
4.00.xx	API change for multi controller support
4.01.xx	Support certificate validation using OCSP stapling V1
5.00.xx	Support the new ASR4.2.1 TcpIp API, but the TLS API is still according to TcpIp ASR4.0.x
6.00.xx	Moved crypto calculation for handshake into separate MainFunction.
7.00.xx	FEAT-1347: TLS as TcpIp plug-in
7.01.xx	FEAT-1741: Support TLS cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 FEAT-1741: Support server certificate validation using Certificate Revocation Lists (CRL)
8.00.xx	FEAT-2005: Release of TLS_RSA_WITH_AES_128_CBC_SHA256, also support TLS1.2 with RSA and client certificate FEAT-1996: TLS Heartbeat Extension
9.00.xx	Release of TLS as a product component
9.01.xx	FEAT-2889: Support TLS 'trusted_ca_keys' extension as specified in RFC 6066
9.02.xx	STORYC-592: TLS Server CertChain Callout in TLS Client STORYC-638: Report Invalid Server CertChain STORYC-1506: Grant access to the TLS master secret

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the BSW module TLS. TLS is not an AUTOSAR defined module.

Supported AUTOSAR Release*:	4.2.1+	
Supported Configuration Variants:	pre-compile	
Vendor ID:	TLS_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	TLS_MODULE_ID	255 decimal (according to ref. [3])
Module Instance:	TLS_INSTANCE_ID	103 decimal (default value, chosen by Vector)

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The TLS provides secure encrypted communication over a connection oriented transport layer (here: TCP). The Tls module implements a TLS client.

2.1 Architecture Overview

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

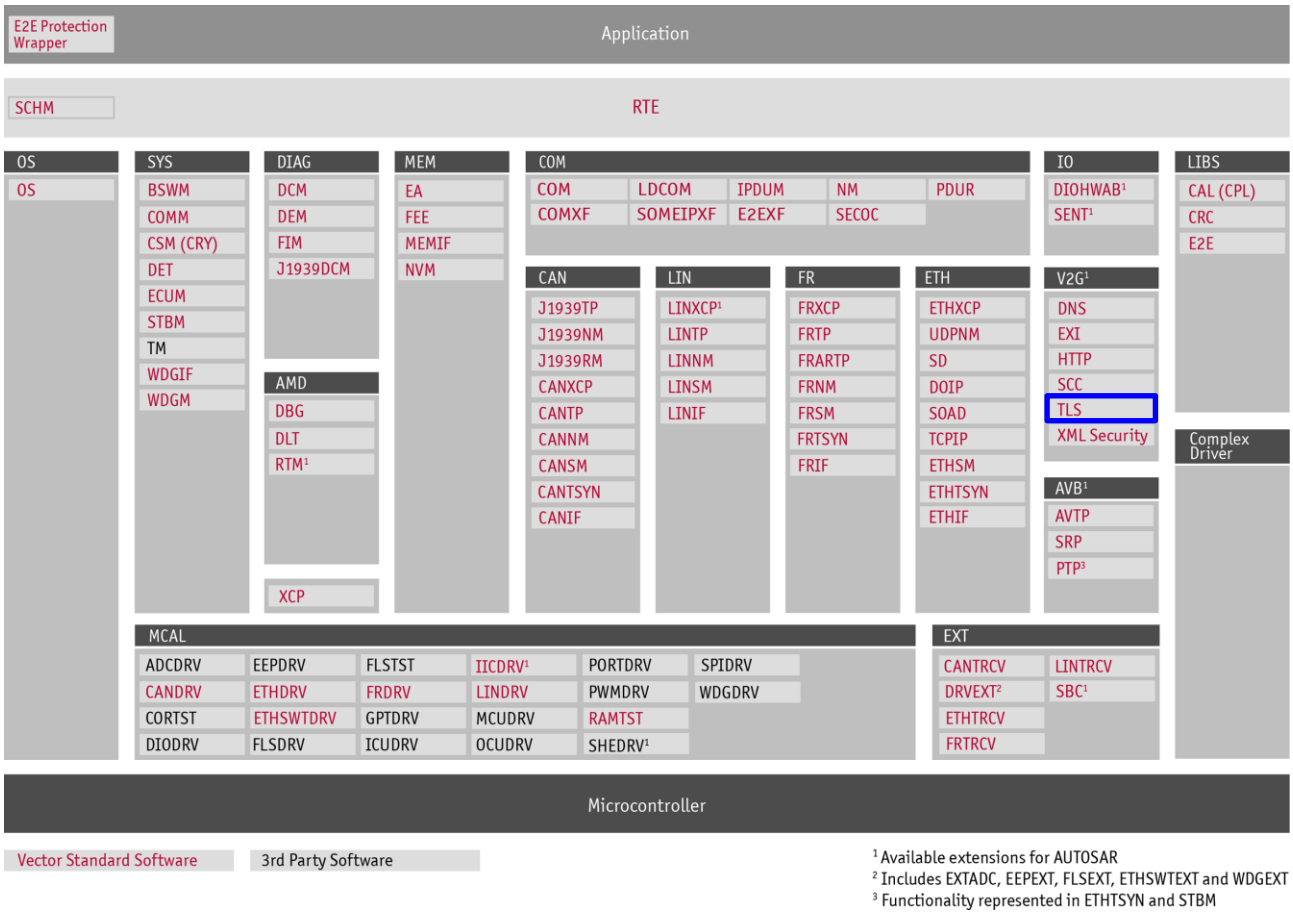

Figure 2-1 AUTOSAR 4.2 Architecture Overview

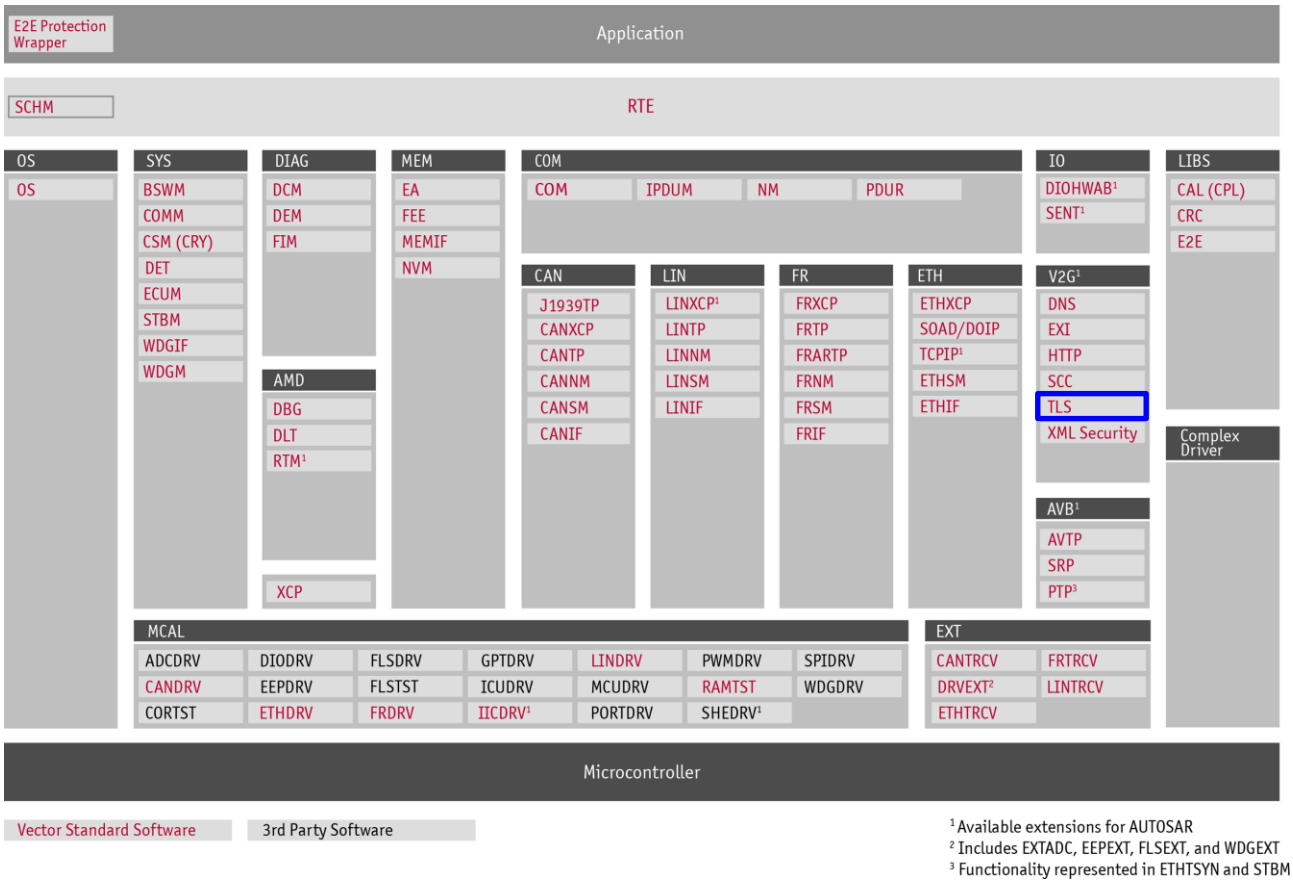

Figure 2-2 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the TLS. These interfaces are described in chapter 4.3.7.

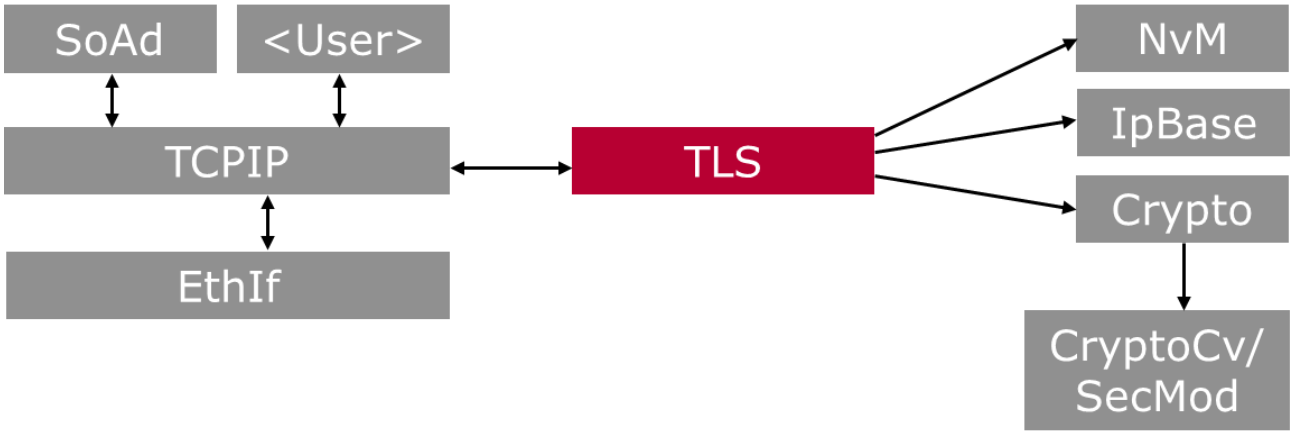

Figure 2-3 Interfaces to adjacent modules of the TLS

Applications access the services of the BSW modules indirectly through the module TcpIp (without using service ports provided by the BSW module via the RTE).

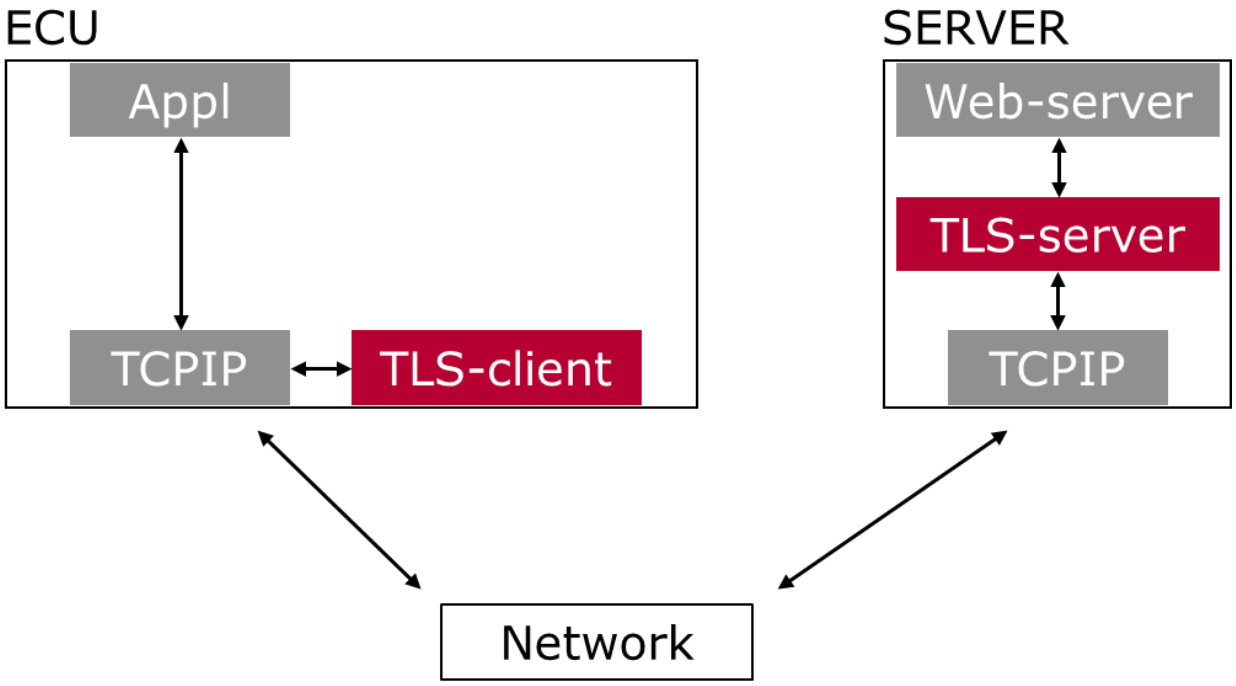

Figure 2-4 global TLS usage architecture

3 Functional Description

3.1 Features

The TLS module supports protocol version 1.0 and 1.2. The module can be configured to support protocol version 1.0, 1.2 or both in parallel.

3.1.1 Supported Cipher Suites

The following TLS cipher suites are supported

- TLS_NULL_WITH_NULL_NULL (only for first handshake)
- TLS_RSA_WITH_NULL_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_NULL_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_ECDSA_WITH_NULL_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

3.1.2 Support certificate validation using OCSP

TLS supports certificate validation using Online Certificate Status Protocol (OCSP) stapling V1 [12].

3.1.3 Support certificate validation using CRL

TLS supports certificate validation using Certificate Revocation Lists (CRL) according to RFC 5280 [10]. There is a global container to specify all stored CRLs (in NvM). The CRLs are parsed after module startup. When receiving a server certificate during the TLS handshake it is validated that the certificate is not listed in the matching CRL.

The current implementation only supports the validation of the server certificate using CRL, not the further received certificate chain.

The current implementation has no mechanism to update the NvM blocks that contain the CRLs but this has to be handled by other means. When the CRLs are updated during runtime, the TLS needs to be informed about this update to update its parsed information about the CRLs.

3.1.4 Support TLS 'trusted_ca_keys' extension as specified in RFC 6066

Tls supports the extension "trusted_ca_indication" as specified by RFC 6066 [11]. The indication is sent as an extension in the client_hello message. In this extension the client sends a hash of each root certificate known to the client.

There is no need for any configuration or user interaction.

3.1.5 TLS Server CertChain Callout in TLS Client

The Tls client can indicate the certificate chain received in the 'certificate' message to the socket owner, combined with the Tls validation result. This is done in an optional callback configured in the TcpIp module for a socket owner.

The indication forwards two elements:

- The complete certificate as received from the server. Each certificate is preceded by a three byte length information.
- The validation result for the certificate chain.

The socket owner then decides what to do with the certificate chain (e.g. do some further evaluations, or store a received certificate as a root certificate to the NvM). Then the socket owner can decide if Tls shall accept the certificate chain and proceed with the handshake, or if the handshake shall be canceled. Making its decision the socket owner can overrule the Tls validation result.

3.1.6 Limitations

3.1.6.1 L001: Client support only

The current version of the TLS implements only a TLS client. So the module cannot act as a server and accept any incoming connection requests.

3.1.6.2 L002: Message size

The current implementation limits the maximum size of incoming messages. This is needed since messages have to be buffered completely to check the message authentication. Forwarding parts of the message without knowing its validity is too dangerous.

3.1.6.3 L005: TLS fragmentation

The current implementation of TLS does not support message fragmentation (for transmission) on TLS level. Encrypted messages have to fit into TLS internal buffers.

3.1.6.4 L007: Maximum number of certificates in the CRL

The maximum supported number of revoked certificate listed in a CRL is limited to 254 certificates. This limitation is caused by the usage of uint8 for chapter indices in the module IpBase (used for BER decoding).

3.2 Initialization

The TLS is initialized by calling the `Tls_InitMemory` service and the `Tls_Init` service with the configuration as parameter. Since the TLS module currently does not support PostBuild functionality the configuration parameter used for Tls_Init can be NULL.

The Crypto module has to be initialized before the TLS module.

3.3 States

The TLS is operational after initialization.

3.4 Main Functions

The TLS has two main functions. The function Tls_MainFunction processes state changes, transmission and reception of messages and the function Tls_MainFunctionLowPrio handles the crypto calculations during the TLS handshake.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [1], if development error reporting is enabled (i.e. pre-compile parameter `TLS_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 TLS ID is 255, the reported module Instance ID is 103.

The reported service IDs identify the services which are described in 5.5. For a list of the used API IDs see the section '`/* TLS ApiIds */`' in file Tls.h in the embedded implementation.

For the IDs of the errors reported to DET see section '`/* TLS DET errors */`' in file Tls.h in the embedded implementation.

3.5.2 Production Code Error Reporting

The current implementation does not support any production error reporting according to the AUTOSAR standard (using the DEM module as specified in [2]).

4 Integration

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

To use the Tls module, the modules ‘TcpIp’, ‘IpBase’, ‘Crypto’ and ‘CryptoCv/SecMod’ are required, too. Therefore the headers TcpIp.h, IpBase.h and Crypto.h will be included.

4.1 Scope of Delivery

The delivery of the TLS contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
`Tls.c`	Implementation
`Tls.h`	API declaration
`Tls_Cbk.h`	API call-back declaration
`Tls_Priv.h`	Component local macro and variable declaration
`Tls_Lcfg.h`	Link-time parameter configuration declaration
`Tls_Types.h`	Types header for API of Tls

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool [config tool].

File Name	Description
`Tls_Cfg.h`	Pre-compile time parameter configuration
`Tls_Lcfg.c`	Link-time parameter configuration

Table 4-2 Generated files

4.2 Critical Sections

The TLS calls service functions of upper layers in order to prevent interruption of critical sections.

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

TLS_EXCLUSIVE_AREA_0: This is the only critical section used in TLS. Currently this critical section is only used in Tls_RxIndication.

4.3 General integration notes

TLS offers a secure communication between two nodes. This communication has to be established using the TLS internal handshake protocol. During the handshake a lot of cryptographic calculations have to be done that consume a significant amount of time and therefore cannot be performed in a single Tls_MainFunction cycle. Certificate and message signatures have to be checked, and asymmetric en-/de-cryptions have to be done.

These calculations are executed in the context of the Tls_MainFunctionLowPrio. This function is called in a BackgroundTask. Using a background task enables the rest of the ECU to operate quite normal and have further communication on different busses. On a 32-bit CPU with 80MHz the calculations for a Handshake using an ECDSA cipher suite could take approximately 1s pure calculation time, and being interrupted all the time makes it even much slower.

4.3.1 External access to memory (e.g. XCP or Diagnostics) has to be limited

For series production ECUs memory access has to be limited so that no RAM or ROM used by TLS is accessible via read or write. Therefore calibration access (e.g. XCP), diagnostics access (e.g. UDS) and any other access has to be switched off entirely or the access to memory and variables has to be switched off. The memory used by TLS can be mapped via AUTOSAR memory mapping.

Caution
This limitation is required to ensure the security of TLS connections and private keys of certificates.

4.3.2 Access to real global time and data information

For the validation of received certificates not only the signature has to be checked, but also the valid life time of a certificate has to be checked. To be able to check times noted in the certificates, TLS needs access to an absolute time information. Therefore TLS configures a callback to get the real time and date information from the application.

4.3.3 Access to real random values

TLS needs good random values. For derivation of random values during runtime TLS uses a random function from the Crypto module. But before the Crypto random functions can be used an initial random value is needed as a seed, since the random functions in Crypto can only produce pseudo random values based on the initial seed. Therefore TLS configures a callback to get an initial seed from the application.

4.3.4 CRL configuration

To use CRLs for server certificate validation, the matching CRLs have to be configured. For each CRL a block in NVM is needed. For the configuration of the parameter ‘/MICROSAR/Tls/TlsGeneral/TlsCrlNvmExchData’ the size of the biggest block used for the CRLs is needed.

4.3.5 Heartbeat configuration

To use the TLS heartbeat this function first has to be enabled by configuring a default heartbeat period. During runtime the usage of the heartbeat has to be enabled for each socket by using the TcpIp_ChangeParameter API with ParameterId TCPIP_PARAMID_V_TLS_HEARTBEAT_MODE and parameter value TLS_HB_PEER_NOT_ALLOWED_TO_SEND (uint8). Additionally the heartbeat period can be set by using the TcpIp_ChangeParameter API with ParameterId TCPIP_PARAMID_V_TLS_HEARTBEAT_PERIOD_S and parameter value <seconds> (uint16). TLS will then cyclically send heartbeat requests to the server and evaluate the received responses. If there is no response to the heartbeat request in the configured period, the upper layer is called using the TcpIp Event callbacks, the EventType is extended to have a value ‘TCPIP_TLS_HEARTBEAT_NO_RESPONSE’.

4.3.6 TLS Connection Setup

Before communicate over an encrypted TLS connection is possible, the connection has to be established. The following steps are required to setup a TLS connection:

- Get a socket by calling TcpIp_<Up>_GetSocket
- Request the usage of TLS by calling TcpIp_ChangeParameter with ParameterId TCPIP_PARAMID_V_USE_TLS and ParamterValue ‘TRUE’ (Boolean)
- Set buffer sizes for TCP by calling TcpIp_ChangeParameter with ParameterIds TCPIP_PARAMID_TCP_RXWND_MAX and TCPIP_PARAMID_V_TCP_TXBUFSIZE (uint32), followed by ParameterIds TCPIP_PARAMID_V_TLS_TXBUFSIZE and TCPIP_PARAMID_V_TLS_RXBUFSIZE (uint32)
- If client authentication shall be used, information about the client certificate and key have to be provided by a call of TcpIp_Tls_SetClientCertInfo
- Optionally bind the socket by calling TcpIp_Bind
- Trigger the connection establishment by calling TcpIp_TcpConnect. As soon as the connection is established the upper layer will be informed by a call of its callback <Up_TcpConnected>

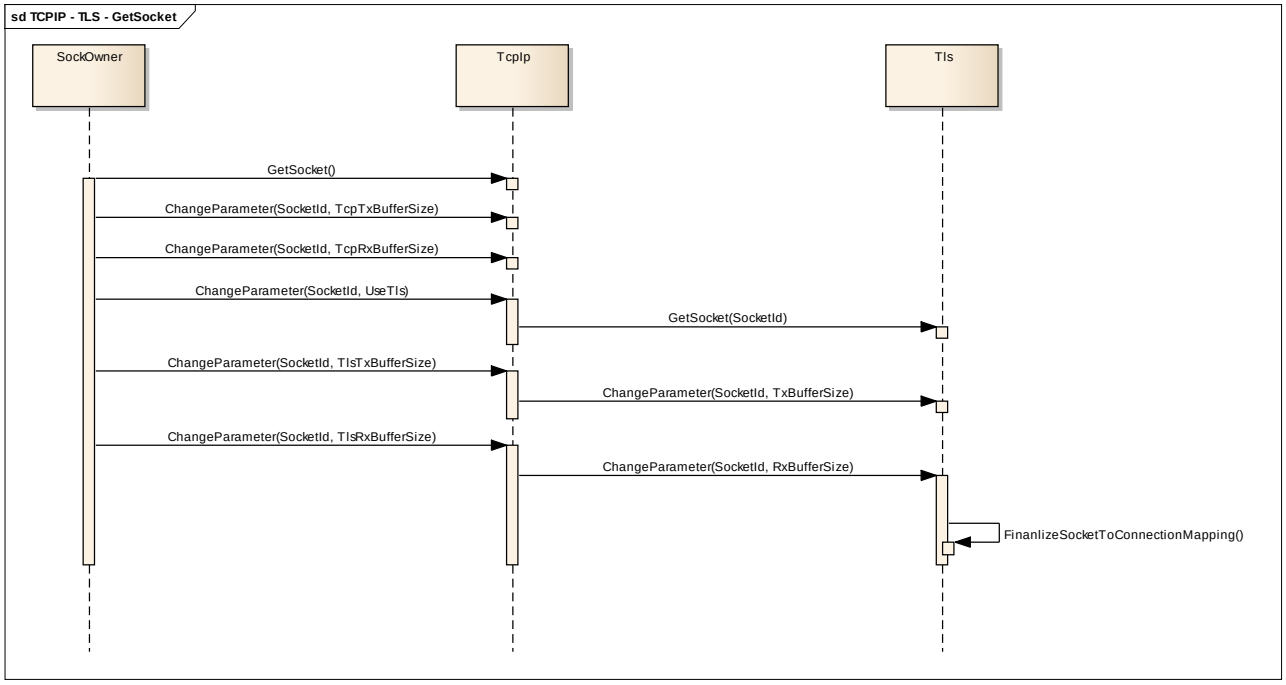

Figure 4-1 Get a socket for TLS and set buffer sizes

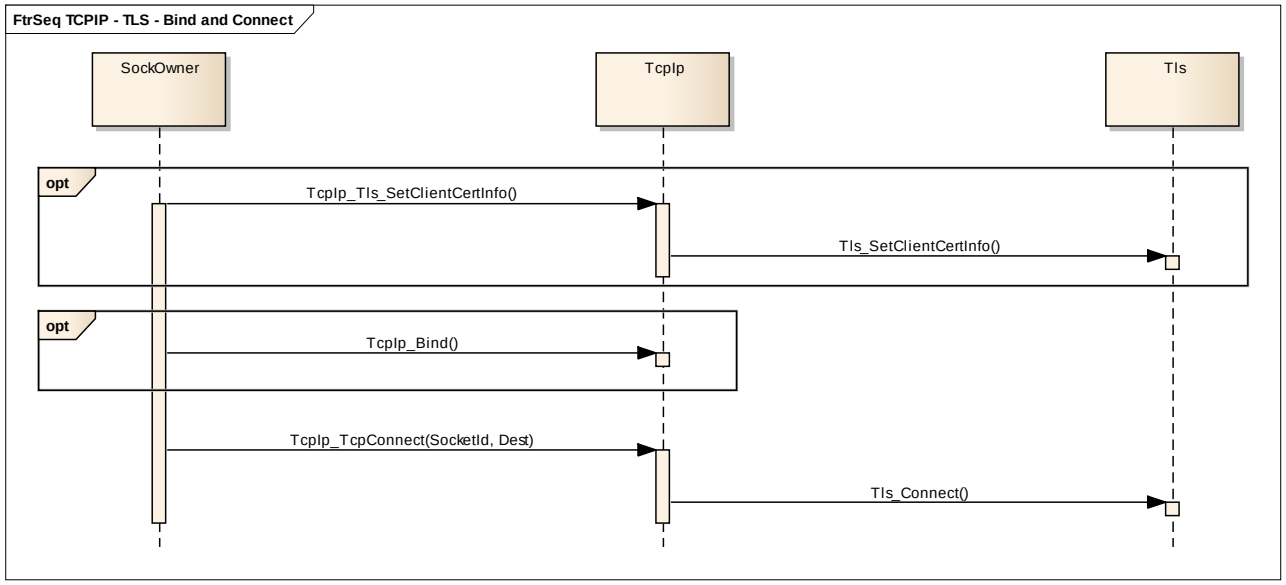

Figure 4-2 Bind and connect a socket

4.3.7 Server CertChain Callout

Tls implements a callback (via TcpIp) that is used to indicate the result of the certificate chain validation. This callback is called in the following cases:

- When a certificate chain could be validated successfully against one of the stored root certificates
- When a certificate chain could not be validated because no matching root certificate was found

> When a certificate chain could not be validated because the signature validation of the chain certificate against the stored root certificate failed

When the callback is issued, the socket owner can do further investigations on the certificates. Then the socket owner can overrule the validation result and determine whether the TLS handshake shall be continued or aborted, or it can just accept the validation result and leave its value as set by Tls.

The validation result can have the following values:

> TCPIP_TLS_VALIDATION_OK

> TCPIP_TLS_VALIDATION_UNKNOWN_CA

> TCPIP_TLS_VALIDATION_LAST_SIGN_INVALID

> TCPIP_TLS_VALIDATION_REFUSED_BY_OWNER

4.3.8 Access to the TLS master secret

By using the API Tls_GetMasterSecret(), access to the used master secret is granted. This feature can be enabled in the configuration. For security reasons the API is disabled by default. If it is enabled, a warning will be displayed in the configurator.

Caution
The API Tls_GetMasterSecret() may never be enabled for production code.

The API Tls_GetMasterSecret() is not wrapped by a corresponding TcpIp API but has to be called directly by the socket owner. To have access to the function declaration the socket owner has to include the header file ‘Tls.h’.

5 API Description

The API of the TLS module is nearly identical to the API of the TCPIP module defined in [4]. This is to make the TLS as transparent as possible, so that the user does not need to behave in different ways depending on the used transport layer.

5.1 TLS usage via the TcpIp module

There are different kinds of functions in TLS.

5.1.1 General functions

Functions that exist for most BSW modules (Init, InitMemory, MainFunction, MainFunctionLowPrio, GetVersionInfo). These functions are called directly by other modules (e.g. by the RTE or the BswM).

5.1.2 Standard communication functions

There are Functions that are similar to the TcpIp communication APIs (GetSocket, Bind, Connect, Transmit, Received, Close, ChangeParameter). These functions are called by TcpIp after the socket owner called the corresponding TcpIp APIs.

5.1.3 TLS specific functions

There are Functions only for TLS (Tls_RootCertWasModified, Tls_GetNvmBlockIdForUsedRootCert, Tls_SetClientCertInfo). These functions are called via TcpIp by prepending a ‘TcpIp_’ to the API name (e.g. TcpIp_Tls_RootCertWasModified). TcpIp will forward these functions directly to TLS.

5.2 Interfaces Overview

The TLS provides the following services:

- Tls_InitMemory()
- Tls_Init()
- Tls_MainFunction()
- Tls_MainFunctionLowPrio()
- Tls_GetVersionInfo()
-
- Tls_SetClientCertInfo()
- Tls_Close()
- Tls_ProvideTxBuffer()
- Tls_TransmitTo()
- Tls_Received()

-
- Tls_RxIndication()
- Tls_TxConfirmation()
- Tls_TcpAccepted()
- Tls_TcpConnected()
- Tls_TcpIpEvent()
- Tls_Cbk_LocalIpAssignmentChg()
-
- Tls_GetNvmBlockIdForUsedRootCert()
- Tls_RootCertWasModified()

5.3 Type Definitions

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

Type Name	C-Type	Description	Value Range
`Tls_ConfigType`	void	Config type used for post-build configuration	
`Tls_StateType`	uint8	Data type for internal module state	
`Tls_EncryptionStateType`	uint8	Data type for internal encryption state	
`TcpIp_ParamIdType (extension of the TCPIP type)`	uint8	There are some parameters that can be changed via the ChangeParameter API	`TCPIP_PARAMID_V_USE_TLS` Configure TcpIp to use TLS for this socket
			`TCPIP_PARAMID_V_TLS_TXBUFSIZE` Configure the transmit buffer size for TLS
			`TCPIP_PARAMID_V_TLS_RXBUFSIZE` Configure the receive buffer size for TLS
			`TCPIP_PARAMID_V_TLS_SELECT_OCSP_REQEST` Specify OCSP requests for the connection
			`TCPIP_PARAMID_V_TLS_HEARTBEAT_MODE` Specify heartbeat mode for the connection

Type Name	C-Type	Description	Value Range
			TCPIP_PARAMID_V_TLS_HEARTBEAT_PERIOD_S Specify heartbeat period for the connection
TcpIp_TcpIpEventType (extension of the TCPIP type)	uint8	No heartbeat response has been received in time after sending a heartbeat request	TCPIP_TLS_HEARTBEAT_NO_RESPONSE

Table 5-1 Type definitions

5.4 Interrupt Service Routines provided by TLS

None

5.5 Services provided by TLS

TLS does not offer any API to the user directly but only through TcpIp.

5.5.1 Tls_InitMemory

Prototype	
`void Tls_InitMemory (void)`	
Parameter	
void [in]	none
Return code	
void	void
Functional Description	
Initializes global variables of the TLS module.	
Particularities and Limitations	
This function is an extension to AUTOSAR. It has to be called before any other calls to the module.	
Call context	
> task level	

Table 5-2 Tls_InitMemory

5.5.2 Tls_Init

Prototype	
`void Tls_Init (const Tls_ConfigType *CfgPtr)`	
Parameter	
CfgPtr [in]	pointer to module configuration
Return code	
void	void
Functional Description	
Initializes the TLS module. Stores the start address of the post build time configuration of the module and may be used to initialize the data structures.	
Particularities and Limitations	
Has to be called before usage of the module.	
Call context	
> task level	

Table 5-3 Tls_Init

5.5.3 Tls_GetVersionInfo

Prototype	
`void Tls_GetVersionInfo (Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
VersionInfoPtr [in]	pointer for version info
Return code	
void	void
Functional Description	
Get the TLS module version.	
Particularities and Limitations	
Call context	
> task level	

Table 5-4 Tls_GetVersionInfo

5.5.4 Tls_GetNvmBlockIdForUsedRootCert

Prototype
`Std_ReturnType Tls_GetNvmBlockIdForUsedRootCert (const Tls_SocketType SockHnd, NvM_BlockIdType *RootCertBlockIdPtr)`

Parameter	
SockHnd [in]	Socket handle
RootCertBlockIdPtr [out]	Pointer to the block ID
Return code	
Std_ReturnType	> E_OK Block ID can be provided > E_NOT_OK request failed
Functional Description	
Get the NVM block ID of the root certificate used for a TLS connection.	
Particularities and Limitations	
Call context	
> task level	

Table 5-5 Tls_GetNvmBlockIdForUsedRootCert

5.5.5 Tls_RootCertWasModified

Prototype	
`Std_ReturnType Tls_RootCertWasModified (const NvM_BlockIdType NvmBlockId)`	
Parameter	
NvmBlockId [in]	BlockId of the NVM block that has changed
Return code	
Std_ReturnType	> E_OK Block is used as a root certificate > E_NOT_OK request failed, block is not used as a root certificate
Functional Description	
Inform the TLS that a root cert has been updated in NVM.	
Particularities and Limitations	
Call context	
> task level	

Table 5-6 Tls_RootCertWasModified

5.5.6 Tls_MainFunction

Prototype	
`void Tls_MainFunction (void)`	
Parameter	
void [in]	none

Return code	
void	void
Functional Description	
Re-initialize closed connections and transmit pending data.	
Particularities and Limitations	
Call context	
> task level	

Table 5-7 Tls_MainFunction

5.5.7 Tls_MainFunctionLowPrio

Prototype	
`void Tls_MainFunctionLowPrio (void)`	
Parameter	
void [in]	none
Return code	
void	void
Functional Description	
Main function to do all crypto computations. This function shall be called from a lower priority task. The execution time can be very long in case of complex crypto calculations.	
Particularities and Limitations	
Call context	
> task level	

Table 5-8 Tls_MainFunction

5.5.8 Tls_SetClientCertInfo

Prototype	
`Std_ReturnType Tls_SetClientCertInfo (Tls_SocketType SockHnd, uint8 *CertPtr, uint8 *KeyPtr, uint16 CertLen, uint16 KeyLen)`	
Parameter	
SockHnd [in]	socket handle
CertPtr [in]	pointer to client certificate (RAM block)
KeyPtr [in]	pointer to client private key (RAM block)
CertLen [in]	length of client certificate

KeyLen [in]	length of client private key
Return code	
Std_ReturnType	> E_OK cert info could be set > E_NOT_OK cert info could NOT be set
Functional Description	
Set the client certificate and key block ids (necessary for client authentication).	
Particularities and Limitations	
Call context	
> task level	

Table 5-9 Tls_SetClientCertInfo

5.5.9 Tls_Close

Prototype
`Std_ReturnType` **`Tls_Close`** `(Tls_SocketType SockHnd)`
Parameter
SockHnd [in] — socket handle
Return code
Std_ReturnType — > E_OK Close trigger could be accepted > E_NOT_OK Close trigger could NOT be accepted
Functional Description
Close TLS connection / socket.
Particularities and Limitations
Call context
> task level

Table 5-10 Tls_Close

5.5.10 Tls_ProvideTxBuffer

Prototype
`BufReq_ReturnType` **`Tls_ProvideTxBuffer`** `(Tls_SocketType SockHnd, const Tls_SockAddrType *DestinationPtr, Tls_PbufType **PbufPtr, uint32 Length)`
Parameter
SockHnd [in] — Socket handle
DestinationPtr [in] — Destination network address and port

PbufPtr [out]	Pointer for the data
Length [in]	Data length in bytes
Return code	
BufReq_ReturnType	> BUFREQ_OK Buffer could be provided > BUFREQ_E_NOT_OK Provide buffer failed
Functional Description	
Request tx buffer for sending data.	
Particularities and Limitations	
DestinationPtr may be null since it is not used.	
Call context	
> task level	

Table 5-11 Tls_ProvideTxBuffer

5.5.11 Tls_TransmitTo

Prototype	
`Std_ReturnType` **`Tls_TransmitTo`** `(Tls_SocketType SockHnd, const Tls_PbufType *PbufPtr)`	
Parameter	
SockHnd [in]	Socket handle
PbufPtr [out]	Pointer of the data
Return code	
Std_ReturnType	> E_OK Data transmission request could be accepted > E_NOT_OK Data transmission request could NOT be accepted
Functional Description	
Trigger sending of the data.	
Particularities and Limitations	
Call context	
> task level	

Table 5-12 Tls_TransmitTo

5.5.12 Tls_Received

Prototype
`Std_ReturnType` **`Tls_Received`** `(Tls_SocketType SockHnd, uint32 Length)`

Parameter	
SockHnd [in]	Socket handle
Length [in]	Length in bytes
Return code	
Std_ReturnType	> E_OK Data reception acknowledged > E_NOT_OK Data reception acknowledge failed
Functional Description	
Acknowledge received data (to the stack).	
Particularities and Limitations	
Call context	
> interrupt or task level	

Table 5-13 Tls_Received

5.5.13 Tls_GetSocket

Prototype
`Std_ReturnType Tls_GetSocket (TcpIp_DomainType Domain, TcpIp_ProtocolType Protocol, TcpIp_SocketIdType *SocketIdPtr)`
Parameter
Domain [in] — IP address family (IPv4 or IPv6)
Protocol [in] — Socket protocol as sub-family of parameter type (TCP or UDP)
out [in] — SocketIdPtr Pointer to socket identifier representing the requested socket. This socket identifier must be provided for all further API calls which requires a SocketId.
UserIndex [in] — Identifier for the socket user (e.g. TLS or HTTP)
Return code
Std_ReturnType — > E_OK The request has been accepted > 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. Each accepted incoming TCP connection also allocates a socket resource. 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 5-14 Tls_GetSocket

5.5.14 Tls_ChangeParameter

Prototype	
`Std_ReturnType` **`Tls_ChangeParameter`** `(Tls_SocketType 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 > E_NOT_OK parameter change failed
Functional Description	
change socket parameter configuration	
Particularities and Limitations	
Call context	
> task level	

Table 5-15 Tls_ChangeParameter

5.5.15 Tls_GetMasterSecret

Prototype	
`Std_ReturnType, TLS_CODE)` **`Tls_GetMasterSecret`**`(Tls_SocketIdType SocketId, uint8 **MsDataPtr, uint8 *MsLenPtr)`	
Parameter	
SocketId [in]	Socket id
MsDataPtr [out]	Pointer to the master secret
MsLenPtr [out]	Pointer where the length of the master secret shall be written to
Return code	
Std_ReturnType	> E_OK Access to the master secret was successful > E_NOT_OK Access to the master secret failed
Functional Description	
Get the master secret for a TLS connection.	
Particularities and Limitations	
This function may not be used in production code since this would be a security issue.	

Call context
> TASK

Table 5-16 Tls_ GetMasterSecret

5.6 Services used by TLS

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

Component	API
DET (optional)	`Det_ReportError`
IpBase	`IpBase_Copy`

Table 5-17 Services used by the TLS

5.7 Callback Functions

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

5.7.1 Tls_TcpConnected

Prototype	
`void` **`Tls_TcpConnected`** `(TcpIp_SockHndType SockHnd)`	
Parameter	
SockHnd [in]	Socket on which a TLS connection is established
Return code	
void	void
Functional Description	
Tls callback, remote side of this socket accepted a connection request.	
Particularities and Limitations	
Call context	
> task level	

Table 5-18 Tls_TcpConnected

5.7.2 Tls_TcpIpEvent

Prototype	
`void Tls_TcpIpEvent (TcpIp_SockHndType SockHnd, IpBase_TcpIpEventType Event)`	
Parameter	
SockHnd [in]	Socket on which the indicated event occurred
Event [in]	Event that occurred in TcpIp
Return code	
void	void
Functional Description	
TcpIp event callback.	
Particularities and Limitations	
Call context	
> -	

Table 5-19 Tls_TcpIpEvent

5.7.3 Tls_RxIndication

Prototype	
`void Tls_RxIndication (Tls_SocketType SockHnd, const Tls_SockAddrType *SourcePtr, uint8 *DataPtr, uint16 DataLen)`	
Parameter	
SockHnd [in]	Socket handle
SourcePtr [in]	Pointer to source network address and port
DataPtr [in]	Pointer to the received data
DataLen [in]	Length of the received data
Return code	
void	void
Functional Description	
Receive Indication Callback.	
Particularities and Limitations	
Call context	
> interrupt or task level	

Table 5-20 Tls_RxIndication

5.8 Call-out Functions

5.8.1 <User>_RxIndication

Prototype	
`void <User>_RxIndication(TcpIp_SocketIdType SockHnd, const TcpIp_SockAddrType SourcePtr, uint8 *DataPtr, uint16 DataLen)`	
Parameter	
SockHnd	socket handle
SourcePtr	source network address and port
DataPtr	pointer to the received data
DataLen	length of the received data
Return code	
void	none
Functional Description	
receive indication	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 5-21 <User>_RxIndication

5.8.2 <User>_TxConfirmation

Prototype	
`void <User>_TxConfirmation(TcpIp_SocketIdType SockHnd, uint16 LenByte)`	
Parameter	
SockHnd	socket handle
LenByte	length in bytes
Return code	
void	none
Functional Description	
transmission confirmation callback	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 5-22 <User>_TxConfirmation

5.8.3 <User>_TcpAccepted

Prototype	
`void <User>_TcpAccepted(TcpIp_SocketIdType SockHnd, TcpIp_SocketIdType SocketIdConnected, TcpIp_SockAddrType RemoteAddrPtr)`	
Parameter	
SockHnd	socket handle
SocketIdConnected	socket handle of the newly created socket
RemoteAddrPtr	remote address of the TCP client that connected
Return code	
void	none
Functional Description	
TCP connection accepted.	
Particularities and Limitations	
Only relevant for TCP servers.	
Call Context	
interrupt or task level	

Table 5-23 <User>_TcpAccepted

5.8.4 <User>_TcpConnected

Prototype	
`void <User>_TcpConnected(TcpIp_SocketIdType SockHnd)`	
Parameter	
SockHnd	socket handle
Return code	
void	none
Functional Description	
TCP connected.	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 5-24 <User>_TcpConnected

5.8.5 <User>_TcpIpEvent

Prototype
`void <User>_TcpIpEvent(TcpIp_SocketIdType SockHnd, IpBase_TcpIpEventType Event)`

Parameter	
SockHnd	socket handle
Event	socket event
Return code	
void	none
Functional Description	
TCP event handling.	
Particularities and Limitations	
none	
Call Context	
interrupt or task level	

Table 5-25 <User>_TcpIpEvent

6 Configuration

6.1 Configuration Variants

The TLS supports the configuration variants

- VARIANT-PRE-COMPILE

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

6.2 Configuration with DaVinci Configurator Pro

For a detailed description of the parameters and possible values see Help in DaVinci Configurator Pro.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator Pro	Generation tool for CANbedded and MICROSAR components (MICROSAR 4)
GENy	Generation tool for CANbedded and MICROSAR components (MICROSAR 3)

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BER	Basic Encoding Rules
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
ETHIF	Ethernet Interface
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
Platform	Hardware including Host and Communication Controller (might also be integrated in Host) on which the communication stack is implemented.
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com