# TLS Stream & Socket API

1.3.0i

Generated by Doxygen 1.8.17

# Chapter 1

# TLS Stream and Socket API

Release 1.2.0b
17.05.22

## 1.1 Motivation

Why this API is needed? For the TLS a underlying SSL-library must be used. The used SSL-library depends on the platform specific implementation (e.g. Botan on ICAS1, WolfSSL on ICAS3, ...). In additions the key management depends on the platform specific trust zone implementation, where the secure key operations are performed. The trust zone implementations depends on the used SoC's. All this specific platform implementations must be encapsulated for the application development.

Disclaimer: The sole idea of the TLS-Lib reference implementation is to define the API and show that it could work. It should give the application developer an idea of how to use TLS-Lib using the provided API. This software was written as a proof of concept and is in no way intended to be used in a production environment: It may contain defects & security flaws, and is not fully tested. Be sure to not use the implementation itself for production usage, only the API.

## 1.2 Communication

The diagram shows the example of the viwi based communication for some services. For instance the service distance must be transported in a secure manor, therefore the sSOA with TLS must be used (see the orange flow between the Distance Service Provider and the HMIs).

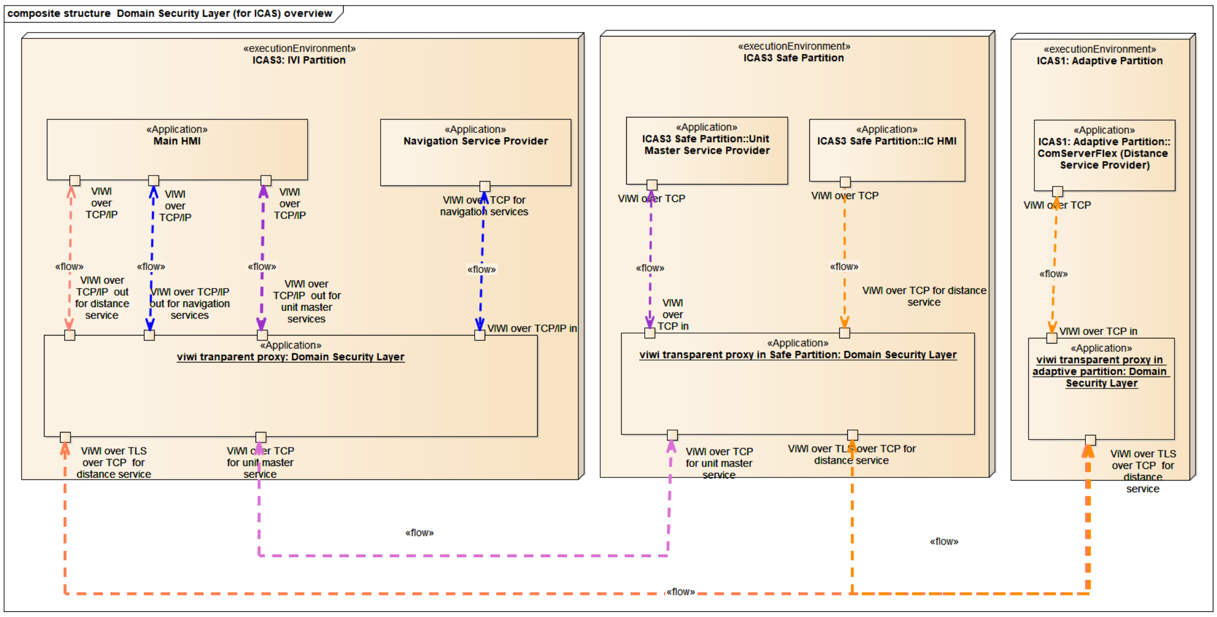
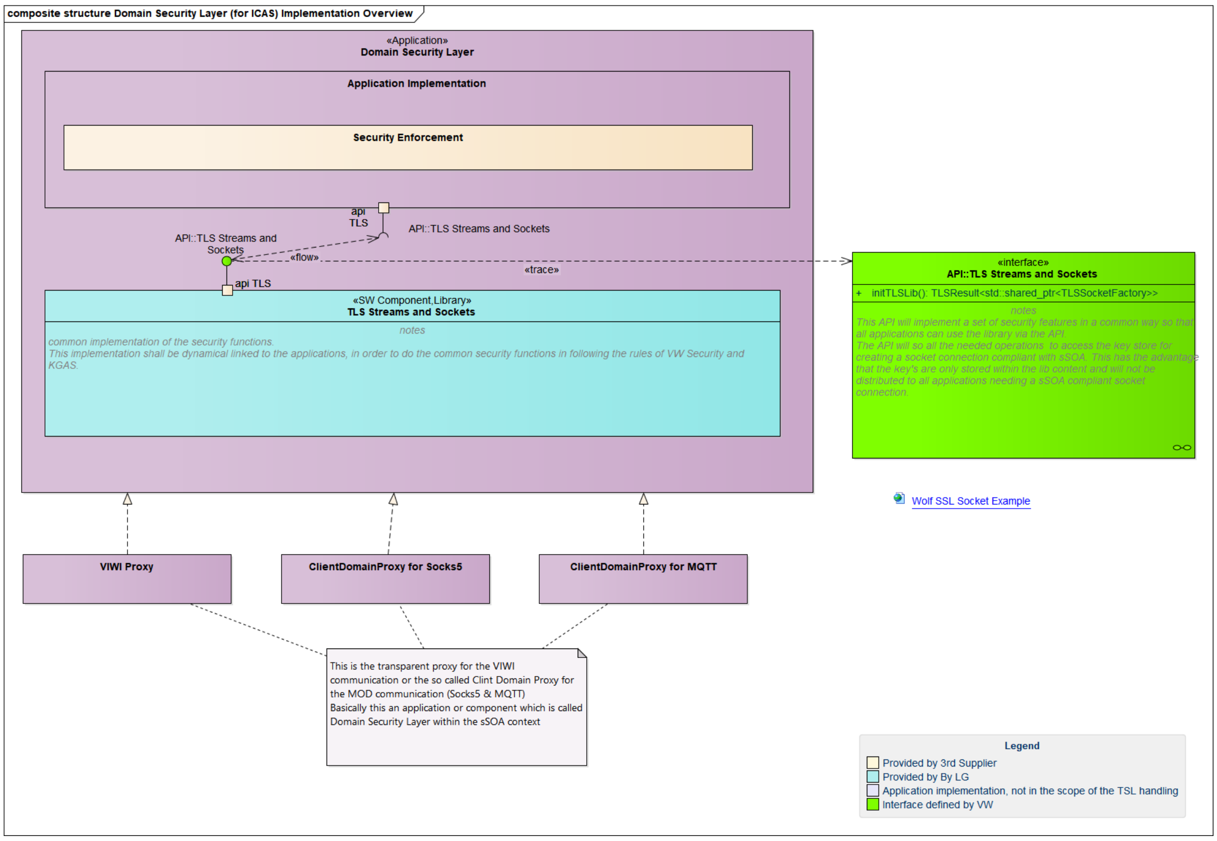

composite structure Domain Security Layer (for ICAS) overview
«executionEnvironment» ICAS3: IVI Partition
«Application» Main HMI
«Application» Navigation Service Provider
VIWI over TCP/IP
VIWI over TCP/IP
VIWI over TCP/IP
VIWI over TCP for navigation services
«flow»
«flow»
«flow»
«flow»
VIWI over TCP/IP out for distance service
VIWI over TCP/IP out for navigation services
VIWI over TCP/IP out for unit master services
VIWI over TCP/IP in
«Application» viwi tranparent proxy: Domain Security Layer
ViWI over TLS over TCP for distance service
ViWI over TCP for unit master service
«executionEnvironment» ICAS3 Safe Partition
«Application» ICAS3 Safe Partition::Unit Master Service Provider
«Application» ICAS3 Safe Partition::IC HMI
ViWI over TCP
ViWI over TCP
«flow»
«flow»
VIWI over TCP in
ViWI over TCP for distance service
«Application» viwi transparent proxy in Safe Partition: Domain Security Layer
ViWI over TCP for unit master service
ViWI over TLS over TCP for distance service
«executionEnvironment» ICAS1: Adaptive Partition
«Application» ICAS1: Adaptive Partition:: ComServerFlex (Distance Service Provider)
ViWI over TCP
«flow»
VIWI over TCP in
«Application» viwi transparent proxy in adaptive partition: Domain Security Layer
ViWI over TLS over TCP for distance service
«flow»
«flow»
«flow»

**Figure 1.1 Domain Security Layer (for ICAS) overview**

## 1.3 Using the API

**Basically the API can be used for the**

- **transparent proxy for the VIWI communication**
- **Clint Domain Proxy for the MOD communication (Socks5 & MQTT)**
- **GateWay for the MOD communication (Socks5 & MQTT)**


composite structure Domain Security Layer (for ICAS) Implementation Overview
«Application»
Domain Security Layer
Application Implementation
Security Enforcement
api TLS
API::TLS Streams and Sockets
API::TLS Streams and Sockets
«flow»
«trace»
api TLS
«SW Component,Library»
TLS Streams and Sockets
notes
«interface»
API::TLS Streams and Sockets
notes
Wolf SSL Socket Example
VIWI Proxy
ClientDomainProxy for Socks5
ClientDomainProxy for MQTT
Legend

+ initTLSLib(): TLSResult<std::shared_ptr<TLSSocketFactory>>

common implementation of the security functions.
This implementation shall be dynamical linked to the applications, in order to do the common security functions in following the rules of VW Security and KGAS.

This API will implement a set of security features in a common way so that all applications can use the library via the API.
The API will so all the needed operations to access the key store for creating a socket connection compliant with sSOA. This has the advantage that the key's are only stored within the lib content and will not be distributed to all applications needing a sSOA compliant socket connection.

This is the transparent proxy for the VIWI communication or the so called Clint Domain Proxy for the MOD communication (Socks5 & MQTT)
Basically this an application or component which is called Domain Security Layer within the sSOA context

| | |
|---|---|
| ▭ | Provided by 3rd Supplier |
| ▭ | Provided by By LG |
| ▭ | Application implementation, not in the scope of the TSL handling |
| 🟩 | Interface defined by VW |

**Figure 1.2 Domain Security Layer Implementation overview**

## 1.4 MOD Special Handling

For the MOD CDP (Client Domain Proxy) a drop TLS is needed, because the stream between the application and the backend is already TLS encrypted and this TLS stream must be tunneled between the CDP and the OCU. For the connection between the CDP and the OCU the TSL-PSK have to be used and must be encrypted as long as the tunneled data steam is stable. >

Below the communication between an application using a MOD service and the backend service is shown. Logically the application will direct talk to the MOD service using TLS (please note this is connection will use the normal certificate based TLS handshake, which is different to the TLS-PSK handshake defined by the sSOA concept). Technically the application will not talk directly the backend service, but it will talk to the client domain proxy (CDP) which have to be located within the same execution environment (e.g. a virtual machine). From the CDP to the Gateway Proxy an tunnel is created where the TLS encrypted data stream is transferred. This tunnel will also be encrypted by TSL-PSK. Since double encryption make no sense and the OCU has a very week CPU the TLS-PSK encryption can be dropped after the connection to the backend is successfully established. >

Therefore we have the basic requirements

- **The TSL-PSK encryption shall be droppable by the client**
- **Dropping of the TSL-PSK encryption shall not lead to a data loss on the data stream.**

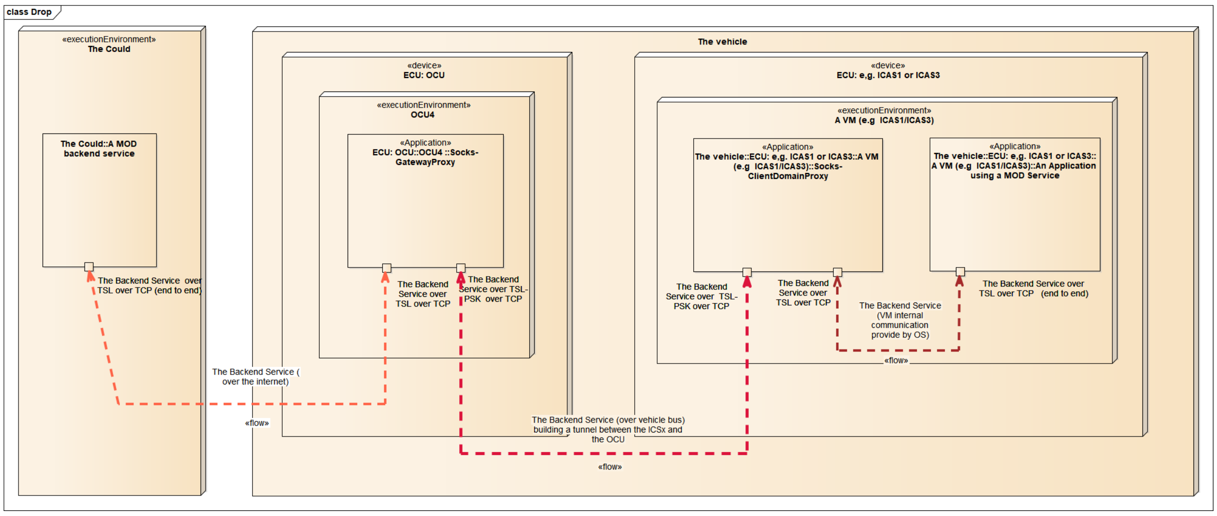

class Drop
«executionEnvironment»
The Could
The Could::A MOD backend service
The Backend Service over TSL over TCP (end to end)
The vehicle
«device»
ECU: OCU
«executionEnvironment»
OCU4
«Application»
ECU: OCU::OCU4 ::Socks-GatewayProxy
The Backend Service over TSL over TCP
The Backend Service over TSL-PSK over TCP
The Backend Service ( over the internet)
«flow»
The Backend Service (over vehicle bus) building a tunnel between the ICSx and the OCU
«flow»
«device»
ECU: e,g. ICAS1 or ICAS3
«executionEnvironment»
A VM (e.g ICAS1/ICAS3)
«Application»
The vehicle::ECU: e,g. ICAS1 or ICAS3::A VM (e.g ICAS1/ICAS3)::Socks-ClientDomainProxy
«Application»
The vehicle::ECU: e,g. ICAS1 or ICAS3::A VM (e.g ICAS1/ICAS3)::An Application using a MOD Service
The Backend Service over TSL-PSK over TCP
The Backend Service over TSL over TCP
The Backend Service (VM internal communication provide by OS)
The Backend Service over TSL over TCP (end to end)
«flow»

**Figure 1.3 MOD Communication with Socks**

## 1.5 Releases

**The API release and the reference implementation can be found at "https://devstack.vwgroup.↩ com/bitbucket/projects/E3THIRD/repos/e3_security_tslapi/"**

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.3.0i | 04.05.23 | api-1.3 | v1.3.0i | • Removed OpenSSL dependency.<br>• Improved local test cases.<br>• Better usage of WolfSSL constants in the reference implementation. |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.3.0h | 30.03.23 | api-1.3 | v1.3.0h | • Extension of error logs in case of wolfssl failure<br>• Extension of error logs of the Revocation check and the Authentic time<br>• cleanup of build scripts |
| 1.3.0g | 19.02.23 | api-1.3 | v1.3.0g | • upgrade to WolfSSL 5.5.4<br>• the tls shall not require authInfo extension on root cert |
| 1.3.0f | 19.01.22 | api-1.3 | v1.3.0f | • build for ICC DEV_9SCR failed<br>• extension for TLSCipherSuiteUseCase "CSUSDefaultWithSoftFail" with OCSP<br>• |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.3.0e | 12.01.22 | api-1.3 | v1.3.0e | • Extend ref impl, documenatation and test suite for O↩CSP deletion handlingion requests<br>• Decode OCSP Response failed ( pointer is Null)<br>• close shall not block for robustness reasons<br>• update copyright note<br>• clarify GPLv2 license handling (tlsAPI-W↩S/test/tls↩Simple↩Sample/src/wolfssl↩_cert_server.↩cpp) |
| 1.3.0d | 12.12.22 | api-1.3 | v1.3.0d | • upgrade to wolfssl 5.5.3 |
| 1.3.0c | 08.12.22 | api-1.3 | v1.3.0c | • DoCache works, but reading from cache leads to verification error<br>• OCSP insert and remove from cache issue<br>• TLS API 1.↩3.0b reference implementation has a misspelled return code in TLSEngine.cpp |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.3.0b | 20.10.22 | api-1.3 | v1.3.0b | • Unintialized bytes in vwg::tls::impl::InternIOStream::Connect()<br>• Thread Manager need to see the native thread name.<br>• fix in botan engine feed function buffer length check |
| 1.3.0a | 11.09.22 | SOP_ME4_2022 | 1.3.0a | • Extension of the ref-impl. for extension of the OCSP Proxy handling for persistent storage |
| 1.2.0b | 17.05.22 | SOP_ME4_2022 | 1.2.0b | • Merge changes from SOP_ME4_2022 1.0.0k<br>• Move connectionLoggingName to Parent |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.2.0a | 28.04.22 | SOP_ME4_2022 | 1.2.0a | • Add client information string for logging<br>• Register Wolfssl trace callback to TLS-↩Library<br>• Direct TLS-↩Library logs into sys-log<br>• Add makefile cappa dependencies to SYSAPI_CO↩LLECTION and FND_LOG<br>• AAdd TLS↩CipherSuite↩UseCases↩Settings with Softfail Implemention |
| 1.1.0k | 16.05.22 | SOP_ME3_2021 | v1.1.0k | • Fix Botan engines (cert + psk) feed() remove internal buffer size constrain<br>• copy ∗.tsv files for packaging<br>• Handle Cmake error - do not ignore |
| 1.1.0j | 29.03.22 | SOP_ME3_2021 | v1.1.0j | • Added Android build variant (linux_amd64↩_icc_sdk), for arm64-v8a, under Clang |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.1.0i | 10.03.22 | SOP_ME3_2021 | v1.1.0i | • Migrated to wolfssl version 5.2.0<br>• A few changes made in order to switch from wolfssl version 4.8.1 to 5.2.0 |
| 1.1.0h | 06.03.22 | SOP_ME3_2021 | v1.1.0h | • Added TLSA↩PI_ENABLE_↩OE3_SPECIA↩L_CERT_HAN↩LING for special handling for the O3<br>• TrM OCSP Caching does not work due to Cache-I↩Ds not being deterministic |
| 1.1.0g | 24.01.22 | SOP_ME3_2021 | v1.1.0g | • Fixed evaluation of public key pins according to RFC 7469, Sec.2.6.<br>• Fixed hash pinning tests in the components tests. |
| 1.1.0f | 25.11.21 | SOP_ME3_2021 | v1.1.0f | • Updated gcc version 9.3.0.<br>• Cleanup API documentation and fixed clang format.<br>• Fixed CI/CD issues. |
| 1.1.0e | 23.09.21 | SOP_ME3_2021 | v1.1.0e | • Fixed linkage error. |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.1.0d | 13.09.21 | SOP_ME3_2021 | v1.1.0d | • Added workaround to BEs scripts for CI/CD. |
| 1.1.0 | 29.08.21 | SOP_ME3_2021 | v1.1.0c | • Updated to wolfssl-4.8.1.<br>• Fixed hash pinning implementation due to crashing.<br>• Deployment CI/CD scripts. |
| 1.1.0b | 07.07.21 | SOP_ME3_2021 | v1.1.0b | • Disable the OCSP requests in case of hard fail fallback mecahnism by enabling the flag ICAS3_N↩O_OCSP_HA↩RD_FAIL due to ICAS3. |
| 1.1.0 | 31.05.21 | SOP_ME3_2021 | v1.1.0a | • Added O↩CSP proxy client/server callbacks. |
| 1.1.0RC4b | 22.04.21 | SOP_ME3_2021 | v1.1.0RC4b | • Updated to WolfSSL-4.7.0.<br>• Fixed memory leaks and valgrind warnings.<br>• Added more unit tests. |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.1.0RC4a | 01.03.21 | SOP_ME3_2021 | v1.1.0RC4a | • Fixed the key size check in WolfSSL PSK↩Callback to be no bigger than keyMaxLength.<br>• Removed const from "toIANA↩ProtocolName" bool return value. |
| 1.1.0RC3a | 11.02.21 | SOP_ME3_2021 | v1.1.0RC3a | • Extension of use cases for cipher suite selection.<br>• Added OC↩SP fallback mechanism.<br>• Improved Unit Test (85% coverage).<br>• Improved component test.<br>• Improve connection process - success is depend on Hash-Pinning check in Wolf↩SSL. |
| 1.1.0RC2a | 09.12.20 | SOP_ME3_2021 | v1.1.0RC2a | • Added authentic time check. |
| 1.1.0RC1a | 30.11.20 | SOP_ME3_2021 | v1.1.0RC1a | • Added alpn support. |
| 1.0.4i | 18.11.20 | SOP_ME_2020 | v1.0.4i | • Fall Back to no-mutex usage for wolfSSL_↩shutdown. |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.4h | 17.11.20 | SOP_ME_2020 | v1.0.4h | • Improved Unit Test.<br>• Updated to WolfSSL 4.5.0.<br>• TLS 1.3 supp- port in Wolf↩ SSL cert-based engine.<br>• Improved C↩ Makefile and repository structure.<br>• Fixed UserIO↩ Stream bug - re- turn user imple- mentaion in is↩ Open and is↩ Close instead of defualt value.<br>• Removed close server after failed "doSSL↩ Handshake" |
| 1.0.4g | 29.10.20 | SOP_ME_2020 | v1.0.4g | • removed wolf↩ SSL_CTX↩ _set_verify - SSL_VERIF↩ Y_PEER mode is turned on by default |
| 1.0.4f | 26.10.20 | SOP_ME_2020 | v1.0.4f | • wolfSSL_get↩ _peer_chain is used instead of wolfSSL_SE↩ SSION_get_↩ peer_chain |
| 1.0.4e | 19.10.20 | SOP_ME_2020 | v1.0.4e | • Supported El- liptic Curves Extension with wolfSSL |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.4d | 05.08.20 | SOP_ME_2020 | v1.0.4d | • Fixed the stream usage by distinguishing between the user's stream implementation and the library's stream implementation |
| 1.0.4c | 27.07.20 | SOP_ME_2020 | v1.0.4c | • Fixed the stream and the engines implementation to support multi-threaded systems |
| 1.0.4b | 22.06.20 | SOP_ME_2020 | v1.0.4b | • Fixed creation of multiple connections with different security levels & ports in wolfSSL PSK engine |
| 1.0.4a | 26.05.20 | SOP_ME_2020 | v1.0.4a | • Fixed creation of multiple connections with different security levels in wolfSSL PSK engine<br>• Fixed stream closing on error issues<br>• Minor naming, documentation and readability fixes |
| 1.0.4 | 17.02.20 | SOP_ME_2020 | v1.0.4 | • CiphersuitesId is represented by string<br>• New Wolfssl version in use 4.3.0 |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.3 | 15.01.20 | SOP_ME_2020 | v1.0.3 | • Support single-sided authentication<br>• Support multiple ciphersuites for cert-based<br>• Support cert↩ Pinning using EC certificates<br>• Updated documentation |
| 1.0.2 | 01.12.19 | SOP_ME_2020 | v1.0.2 | • Fix IOStream headers<br>• Update Mock↩ TEE |
| 1.0.1 | 03.11.19 | SOP_ME_2020 | v1.0.1 | • Fixed API<br>• Changed signedness of some parameters |
| 1.0.0 | 02.09.19 | SOP_ME_2020 | v1.0.0 | • Added server name indication (SNI) support<br>• Fixed shutdown issues |
| 1.0.0 RC8a | 04.08.19 | SOP_ME_2020 | RC8a | • Replaced TEE mock<br>• Added TEE error codes<br>• Enabled usage of PSK key of size 256 & 512 in addition to 128 bit<br>• Added functionality for creating socket on already accepted connection FD |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.0 RC7b | 01.07.19 | RC7 | | • added certificate pinning |
| 1.0.0 RC7a | 27.06.19 | RC7 | v1.0.0_RC7 | • added OCSP stapling<br>• added cert pinning (Botan only)<br>• added support for TLS alert codes<br>• extended botan for dropTLS support |
| 1.0.0 RC6c | 18.04.19 | RC6c Cert POC | | • Adaptions for the e3 SW-PAC |
| 1.0.0 RC6b | 18.04.19 | RC6b PSK POC | | • Adaptions for the e3 SW-PAC |
| 1.0.0 RC6a | 07.03.19 | RC6_pre | | • adding support for certificate based client<br>• refactor botan engine<br>• refactor wolfssl engine |
| 1.0.0 RC5b | 04.03.19 | master | | • fixed non-blocking send<br>• fix IPv6 bind failure<br>• added new logging mechanism |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.0 RC5a | 18.02.19 | master | | • adding clinet/server hint<br>• update of readme file, to refect the last deliries<br>• cleanup of API<br>• adding session creation using file-descriptor<br>• separating the build process(engine and library) |
| 1.0.0 RC4 Preview | 05.12.18 | rc4_pre | | • Extension for viwi proxy: adding an factory to upgrade a server socket.<br>• Extension for MOD to support certificate based TLS |
| 1.0.0 RC3f | 24.01.19 | master | | • adding test aplication<br>• fixing readme.<br>• adding gcov support |
| 1.0.0 RC3e | 17.01.19 | master | | • fix memory leaks |
| 1.0.0 RC3d | 16.12.18 | master | | • Adding support for non-blocking API calls |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.0 RC3c | 06.12.18 | master | v1.0.0_RC3c | • This version will only contain bug fixes.<br>• FIX of IPv6 issues.<br>• Fix return of send/receive is an enum (TLS←EngineError)<br>• Every accept in the server sockets creates a new engine |
| 1.0.0 RC3b | 15.11.18 | master | v1.0.0_RC3b | • Complete the reference implementation. Adding missing function calls<br>• Providing a verification suite which tests the implementation against the expectations.<br>• changed to cmake for building the reference library and verification suite. |
| 1.0.0 RC3a | 05.11.18 | | v1.0.0_RC3a | • Adding Botan SSL Support to reference implementation. |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.0 RC3 | 30.10.18 | | v1.0.0_RC3 | • ErrorHandler use shared_ptr for inet<br>• ErrorHandler use enum for error code<br>• InetAddress↩ Factory make ctor private.<br>• add c++ style callbacks<br>• improve return code – setters to ctors<br>• using Lamda expression for callback<br>• provide a initial reference implementation |
| Preview for 1.0.0 RC3 | 25.10.18 | preview_1.0.0_RC3 | | • ErrorHandler use shared_ptr for inet<br>• ErrorHandler use enum for error code<br>• InetAddress↩ Factory make ctor private.<br>• add c++ style callbacks<br>• improve return code – setters to ctors |

| Version | Release Date | Branch | Tag | Notes |
|---|---|---|---|---|
| 1.0.0 RC2 | 22.10.18 | master | v1.0.0_RC2 | • update of return codes (new codes added).<br>• adding reference implementation of tlsLibrary.<br>• adding reference project providing server and client samples. |
| 1.0.0 RC1 | 22.10.18 | master | | • Initial Version |

# Chapter 2

# Deprecated List

**Member vwg::tls::ITLSSocketFactory::createTlsClient (const std::shared_ptr< IOStream > stream, const std::string &hostName, const CertStoreID &certStoreId, const ClientCertificateSetID &client↩ CertificateSetID, const CipherSuiteIds &cipherSuiteIds, const TimeCheckTime &checkTime, const std↩ ::vector< HashSha256 > &httpPublicKeyPinningHashs, const bool revocationCheckEnabled=false)=0**

this method becomes deprecated since 1.1.0, please use method with ALPN support.

# Chapter 3

# Namespace Index

## 3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 4

# Hierarchical Index

## 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 5

# Class Index

## 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 6

# File Index

## 6.1 File List

Here is a list of all files with brief descriptions:

# Chapter 7

# Namespace Documentation

## 7.1 vwg Namespace Reference

This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.

### Namespaces

- tls
- types

### 7.1.1 Detailed Description

This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.

## 7.2 vwg::tls Namespace Reference

### Classes

- class AlpnMode

  *A setting container for ALPN supporting. There are basically three modes possible:*
- class IANAProtocolFunction

  *This class contains some helper methods when conversion from the IANAProtocol enum value to Protocol name.*
- class IInetAddress

  *Representation an interface of an IP address. Basically this will give you an immutable IP address interface.*
- class InetAddressFactory

  *This a definition of a the factory to create instances of the IInetAddress. The supplier has to provide the implementation of the static methods by this class. Basically there is no need to create an instance of this class.*
- class IOStream

  *Representation an interface of an I/O stream. Can read, write and close.*
- class ITLSClientSocket

  *Server TLS-PSK aware client socket interface. This interface must be implemented by the supplier.*
- class ITLSErrorListener
- class ITLSOcspHandler

  *This interface defines APIs to process and handle OCSP messages.*
- class ITLSServerSocket

  *Server TLS-PSK aware server socket interface. This interface must be implemented by the supplier.*
- class ITLSSessionEndpoint

  *Represents a communication session between a service provider and a service consumer. This interface must be implemented by the supplier.*
- class ITLSSocketBase

  *This is an interface which defines a set of operation and features have to be available on each socket and session endpoint.*
- class ITLSSocketFactory

  *This is the interface of the socket factory. One need to get an instance of this interface to create a server or a client socket. Use the function initTLSLib to get the instance of the factory. The implementation will have only one instance of the factory.*
- struct TimeCheckTime

  *This is a structure that will be used to pass the authentic time. basically this time will be compared with the system time, as shown below.*
- class TLSConnectionSettings

  *this class is used to define the TLS connection properties for a backend TLS connection. This class contains a set of configuration properties for the TLS connection.*
- class TLSOcspCachedResponse

  *This class represents a cached OCSP response message.*
- class TLSOcspRequest

  *This class represents a wrapper for a raw OCSP request message.*
- class TLSOcspRequestResponse

  *This class represents a wrapper for a raw OCSP response message which used as a result object from the OCSP. Proxy process after requests processing.*
- struct TLSResult

  *This is a struct to return the return code or the value in case the operation is performed successful. Basically it will take a payload or an return code. One can assume that the paylod is empty if the operation failed. One have to use failed or succeeded first to check if the payload is set or not first. Currently it is assumed that the access of a empty payload will fail and an error is raised.*

## Typedefs

- using CipherSuiteIds = std::string
- using SPIInetAddress = std::shared_ptr< IInetAddress >
- using IInetAddressResult = TLSResult< SPIInetAddress >
- using ApiVersionType = std::string
- typedef void(∗ ErrorHandler) (SPIInetAddress inet, const UInt16 port, const TLSReturnCodes errorCode)
- using SPITLSSessionEndpoint = std::shared_ptr< ITLSSessionEndpoint >
- using TLSSessionStatusListener = std::function< void(SPITLSSessionEndpoint endpoint, const TLSSessionStatus status)>
- using TLSDropStatusListener = std::function< void(SPITLSSessionEndpoint endpoint, const TLSDropStatus status)>
- using SPTLSSessionEndpoint = std::shared_ptr< ITLSSessionEndpoint >
- using TLSSessionEndpointResult = TLSResult< SPTLSSessionEndpoint >
- using ClientCertificateSetID = std::string
- using HashSha256 = std::vector< char >
- using CertStoreID = std::string
- using ITLSSocketFactoryResult = TLSResult< std::shared_ptr< ITLSSocketFactory > >
- using SPTLSClientSocket = std::shared_ptr< ITLSClientSocket >
- using SPTLSServerSocket = std::shared_ptr< ITLSServerSocket >
- using TLSClientSocketResult = TLSResult< SPTLSClientSocket >
- using TLSServerSocketResult = TLSResult< SPTLSServerSocket >

## Enumerations

- enum CipherSuiteId : vwg::types::UInt16 {
  TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = 0xCCA9, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_S
  = 0xC02C, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 = 0xC02B, TLS_ECDHE_RSA_WITH_AES_256_GCM_SH
  = 0xC030,
  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 = 0xC02F, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  = 0x009F, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 = 0x009E, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA25
  = 0xC023,
  TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = 0xCCA8, TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SH
  = 0xCCAA, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA = 0xC009, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SH
  = 0xC00A,
  TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 = 0xC027, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
  = 0xC013, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA = 0xC014, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  = 0x0067,
  TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 = 0x006B, TLS_RSA_WITH_AES_128_GCM_SHA256 = 0x009C, TLS_RSA_WITH_AES_256_GCM_SHA384 = 0x009D, TLS_RSA_WITH_AES_128_CBC_SHA256
  = 0x003C,
  TLS_RSA_WITH_AES_256_CBC_SHA256 = 0x003D, TLS_RSA_WITH_AES_128_CBC_SHA = 0x002F, TLS_RSA_WITH_AES_256_CBC_SHA = 0x0035, TLS_RSA_WITH_3DES_EDE_CBC_SHA = 0x000A }

  *This enum defines the list of permitted cipher suits.*
- enum StreamReturnCode { RC_STREAM_WOULD_BLOCK = -1, RC_STREAM_IO_ERROR = -2 }

  *Error values for receiving or sending data.*
- enum IANAProtocol { NONE = 0, HTTP = 1, HTTP2 = 2 }

  *This enum defines the supported protocols which can be used in case ALPN is used. Please see the IANAProtocol definitions in RFC7230* `https://tools.ietf.org/html/rfc7230`.
- enum TLSCipherSuiteUseCasesSettings : UInt32 {
  CSUSDefault = 0, CSUSLegacy = 1, CSUSLongtermSecure = 2, CSUSIanaRecommended = 3,
  CSUSDefaultWithSoftFail = 4, CSUSEndOfEnum }
- enum SecurityLevel : UInt32 { AUTHENTIC_WITHPSK = 0, CONFIDENTIAL_WITHPSK = 1 }

  *Defines the SSOA confidentiality.*
- enum SocketType : UInt32 { SOCKETTYPE_STREAM = 0, SOCKETTYPE_DATAGRAM = 1 }

*Defines the socket type.*

- enum TLSDropSuppot : UInt32 { TLS_NOT_DROPABLE = 0, TLS_DROPABLE = 1 }
- enum TLSReturnCodes : Int32 {
RC_TLS_SUCCESSFUL = 0, RC_TLS_INIT_FAILED = 1, RC_TLS_CONNECT_FAILED, RC_TLS_ACCEPT_FAILED,
RC_TLS_INVALID_DOMAIN, RC_TLS_KEY_MISSING, RC_TLS_KEY_ERROR, RC_TLS_USAGE_AFTER_CLEANUP,
RC_TLS_IO_ERROR, RC_TLS_WOULD_BLOCK_READ, RC_TLS_WOULD_BLOCK_WRITE, RC_TLS_PEER_CLOSED,
RC_TLS_AUTHENTIC_TIMECHECK_FAILED, RC_TLS_MAX_PERMITTED_DEVIATION, RC_TLS_SEND_AFTER_SHUTD
RC_TLS_INVALID_IP = 1000,
RC_TLS_DROPPING_NOTSUPPORTED, RC_TLS_DROPPING_FAILED, RC_TLS_PUBLIC_KEY_PINNING_FAILED,
RC_TLS_UNEXPECTED_MESSAGE = 2010,
RC_TLS_BAD_RECORD_MAC = 2020, RC_TLS_RECORD_OVERFLOW = 2022, RC_TLS_DECOMPRESSION_FAILURE
= 2030, RC_TLS_HANDSHAKE_FAILURE = 2040,
RC_TLS_BAD_CERTIFICATE = 2042, RC_TLS_UNSUPPORTED_CERTIFICATE = 2043, RC_TLS_CERTIFICATE_REVOKE
= 2044, RC_TLS_CERTIFICATE_EXPIRED = 2045,
RC_TLS_CERTIFICATE_UNKNOWN = 2046, RC_TLS_ILLEGAL_PARAMETER = 2047, RC_TLS_UNKOWN_CA
= 2048, RC_TLS_UNKNOWN_CA = 2048,
RC_TLS_ACCESS_DENIED = 2049, RC_TLS_DECODE_ERROR = 2050, RC_TLS_DECRYPT_ERROR =
2051, RC_TLS_PROTOCOL_VERSION = 2070,
RC_TLS_INSUFFICIENT_SECURITY = 2071, RC_TLS_NO_RENEGOTIATION = 2100, RC_TLS_UNSUPPORTED_EXTENS
= 2110, RC_TLS_CERTIFICATE_UNOBTAINABLE = 2111,
RC_TLS_UNRECOGNIZED_NAME = 2112, RC_TLS_BAD_CERTIFICATE_STATUS_RESPONSE = 2113,
RC_TLS_BAD_CERTIFICATE_HASH_VALUE = 2114, RC_TLS_NO_APPLICATION_PROTOCOL = 2120,
RC_TLS_TEE_ACCESS_ERROR = 3000, RC_TLS_CERTSTORE_NOT_FOUND, RC_TLS_UNKNOWN_CLIENT_CERTIFIC
RC_TLS_CLIENT_CERTIFICATE_SET_IDERROR,
RC_TLS_PROGRAMMING_ERROR_RESULT = -1000 }
- enum TLSDropStatus : UInt32 {
TLSDROP_SECURED, TLSDROP_DROPPED, TLSDROP_REQUESTED, TLSDROP_SEND_LOCKED,
TLSDROP_PERFORMED }
- enum TLSSessionStatus : UInt32 { TLSSESSION_SECURED, TLSSESSION_UNSECURED, TLSSESSION_BROKEN,
TLSSESSION_CLOSED }

  *Defines the possible status values of the session.*

## Functions

- const ApiVersionType ApiVersion ("TLS_API_1.3")
- ITLSSocketFactoryResult initTLSLib ()

  *This is the entry point for the library. This will return the Socket factory when all initialization needed are successfully performed. These is basically initialization of:*
- void cleanupTLSLib ()

  *Use this method to cleanup the implementation. This can be used to cleanup the TLS library (e.g. Wolf SSL or Botan SSL). after this the ITLSSocketFactory will not return any socket instance.*

## Variables

- const static unsigned int MAX_PERMITTED_DEVIATION = 86400

  *Defines the maximum permitted deviation of |expectedTime - system_time.now()|. since 1.1.0.*
- const static TimeCheckTime CHECK_TIME_OFF = {0, 0}

  *Defines that time check is not required.*
- const static UInt32 DEFAULT_OCSP_ONLINE_TIMEOUT_MS = 30000

  *Defines a default OCSP timeout in milliseconds.*
- const static AlpnMode ALPN_OFF = AlpnMode(std::vector<IANAProtocol>{NONE})

  *Defines that ALPN is off and the protocol is undecided, this is identical to TLS without any ALPN support.*
- const static AlpnMode ALPN_DEFAULT = AlpnMode(std::vector<IANAProtocol>{HTTP})

*Defines the default ALPN.*

- const static AlpnMode ALPN_HTTP2 = AlpnMode(std::vector<IANAProtocol>{IANAProtocol::HTTP2})

  *Defines HTTP2 ALPN.*
- const static AlpnMode ALPN_ANY = AlpnMode(std::vector<IANAProtocol>{IANAProtocol::HTTP2, IANA↩ Protocol::HTTP})

  *Defines all supported ALPN.*
- const static std::string CSUSDefaultStr = "default"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.*
- const static std::string CSUSDefaulWithSoftFailtStr = "default_with_soft_fail"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.*
- const static std::string CSUSLegacyStr = "legacy"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLegacy for more detail.*
- const static std::string CSUSLongtermSecureStr = "longterm_secure"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLongtermSecure for more detail.*
- const static std::string CSUSIanaRecommendedStr = "iana_recommended"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSIanaRecommended for more detail.*
- const UInt32 MODE_BLOCKING = 0
- const UInt32 MODE_ASYNC = 1
- const int TLS_EOF = 0

  *Defines the EOF value 0 in case that the connection is closed. This can happen if a closed on a socket is made and there are pending receive and send. Please be aware of that EOF is defined as -1.*
- const ClientCertificateSetID CLINET_CERTICATE_SET_BASE = "BASE"

## 7.2.1 Typedef Documentation

### 7.2.1.1 ApiVersionType

```
using vwg::tls::ApiVersionType = typedef std::string
```

Definition at line 48 of file TLSApiTypes.h.

### 7.2.1.2 CertStoreID

```
using vwg::tls::CertStoreID = typedef std::string
```

Definition at line 52 of file TLSSocketFactory.h.

#### 7.2.1.3 CipherSuiteIds

```
using vwg::tls::CipherSuiteIds = typedef std::string
```

Definition at line 71 of file CipherSuitesDefenitions.h.

#### 7.2.1.4 ClientCertificateSetID

```
using vwg::tls::ClientCertificateSetID = typedef std::string
```

Definition at line 49 of file TLSSocketFactory.h.

#### 7.2.1.5 ErrorHandler

```
typedef void(* vwg::tls::ErrorHandler) (SPIInetAddress inet, const UInt16 port, const TLSReturnCodes
errorCode)
```

Definition at line 999 of file TLSApiTypes.h.

#### 7.2.1.6 HashSha256

```
using vwg::tls::HashSha256 = typedef std::vector<char>
```

Definition at line 51 of file TLSSocketFactory.h.

#### 7.2.1.7 IInetAddressResult

```
using vwg::tls::IInetAddressResult = typedef TLSResult<SPIInetAddress>
```

Definition at line 132 of file InetAddress.h.

#### 7.2.1.8 ITLSSocketFactoryResult

```
using vwg::tls::ITLSSocketFactoryResult = typedef TLSResult<std::shared_ptr<ITLSSocketFactory>
>
```

Definition at line 844 of file TLSSocketFactory.h.

#### 7.2.1.9 SPIInetAddress

```
using vwg::tls::SPIInetAddress = typedef std::shared_ptr<IInetAddress>
```

Definition at line 127 of file InetAddress.h.

#### 7.2.1.10 SPITLSSessionEndpoint

```
using vwg::tls::SPITLSSessionEndpoint = typedef std::shared_ptr<ITLSSessionEndpoint>
```

Definition at line 96 of file TLSSession.h.

#### 7.2.1.11 SPTLSClientSocket

```
using vwg::tls::SPTLSClientSocket = typedef std::shared_ptr<ITLSClientSocket>
```

Definition at line 145 of file TLSSockets.h.

#### 7.2.1.12 SPTLSServerSocket

```
using vwg::tls::SPTLSServerSocket = typedef std::shared_ptr<ITLSServerSocket>
```

Definition at line 146 of file TLSSockets.h.

#### 7.2.1.13 SPTLSSessionEndpoint

```
using vwg::tls::SPTLSSessionEndpoint = typedef std::shared_ptr<ITLSSessionEndpoint>
```

Definition at line 302 of file TLSSession.h.

#### 7.2.1.14 TLSClientSocketResult

```
using vwg::tls::TLSClientSocketResult = typedef TLSResult<SPTLSClientSocket>
```

Definition at line 147 of file TLSSockets.h.

#### 7.2.1.15 TLSDropStatusListener

```
using vwg::tls::TLSDropStatusListener = typedef std::function<void(SPITLSSessionEndpoint endpoint,
const TLSDropStatus status)>
```

Definition at line 108 of file TLSSession.h.

#### 7.2.1.16 TLSServerSocketResult

```
using vwg::tls::TLSServerSocketResult = typedef TLSResult<SPTLSServerSocket>
```

Definition at line 148 of file TLSSockets.h.

#### 7.2.1.17 TLSSessionEndpointResult

```
using vwg::tls::TLSSessionEndpointResult = typedef TLSResult<SPTLSSessionEndpoint>
```

Definition at line 303 of file TLSSession.h.

#### 7.2.1.18 TLSSessionStatusListener

```
using vwg::tls::TLSSessionStatusListener = typedef std::function<void(SPITLSSessionEndpoint
endpoint, const TLSSessionStatus status)>
```

Definition at line 102 of file TLSSession.h.

### 7.2.2 Enumeration Type Documentation

**Enumerator**

### 7.2.2.1 CipherSuiteId

```
enum vwg::tls::CipherSuiteId : vwg::types::UInt16
```

This enum defines the list of permitted cipher suits.

**Enumerator**

| | |
|---|---|
| TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 | |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 | |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 | |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | |
| TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 | |
| TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 | |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA | |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA | |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA | |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA | |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 | |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 | |
| TLS_RSA_WITH_AES_128_GCM_SHA256 | |
| TLS_RSA_WITH_AES_256_GCM_SHA384 | |
| TLS_RSA_WITH_AES_128_CBC_SHA256 | |
| TLS_RSA_WITH_AES_256_CBC_SHA256 | |
| TLS_RSA_WITH_AES_128_CBC_SHA | |
| TLS_RSA_WITH_AES_256_CBC_SHA | |
| TLS_RSA_WITH_3DES_EDE_CBC_SHA | |

Definition at line 42 of file CipherSuitesDefenitions.h.

### 7.2.2.2 IANAProtocol

```
enum vwg::tls::IANAProtocol
```

This enum defines the supported protocols which can be used in case ALPN is used. Please see the IANAProtocol definitions in RFC7230 `https://tools.ietf.org/html/rfc7230`.

**Since**

1.1.0

**Enumerator**

| | |
|---|---|
| NONE | |
| HTTP | |
| HTTP2 | |

Definition at line 57 of file TLSApiTypes.h.

#### 7.2.2.3 SecurityLevel

```
enum vwg::tls::SecurityLevel : UInt32
```

Defines the SSOA confidentiality.

AUTHENTIC_WITHPSK defines PSK connection with authentication.

CONFIDENTIAL_WITHPSK defines confidential PSK connection.

**Enumerator**

| | |
|---|---|
| AUTHENTIC_WITHPSK | |
| CONFIDENTIAL_WITHPSK | |

Definition at line 977 of file TLSApiTypes.h.

#### 7.2.2.4 SocketType

```
enum vwg::tls::SocketType : UInt32
```

Defines the socket type.

SOCKETTYPE_STREAM Stream socket.

SOCKETTYPE_DATAGRAM Datagram socket.

**Enumerator**

| | |
|---|---|
| SOCKETTYPE_STREAM | |
| SOCKETTYPE_DATAGRAM | |

Definition at line 986 of file TLSApiTypes.h.

#### 7.2.2.5 StreamReturnCode

```
enum vwg::tls::StreamReturnCode
```

Error values for receiving or sending data.

**Enumerator**

| | |
|---|---|
| RC_STREAM_WOULD_BLOCK | |
| RC_STREAM_IO_ERROR | |

Definition at line 44 of file IOStream.h.

#### 7.2.2.6 TLSCipherSuiteUseCasesSettings

```
enum vwg::tls::TLSCipherSuiteUseCasesSettings : UInt32
```

this enum defines the possible setting cipher suits based on predefined use cases. This will replace the cipher suite list. Especially in case of using TLS1.2 and TLS1.3 in parallel, it may will be more complex. In addition the ECC curves are currently not covered sufficient in the TLS1.0.x. Instead of using the list of cipher suites, a set of use cases can will be defined. Based on the use cases the cipher suites are selected.

Please see https://devstack.vwgroup.com/jira/browse/IMAN-46128 for the cipher suits associted to the use cases.

**CSUSDefault** This defines the default cipher suite set, which is defined for in the according QHAL. This is the default for all MOD functions.

```
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256 (TLS1.3 only)
- TLS_AES_256_GCM_SHA384 (TLS1.3 only)
- TLS_CHACHA20_POLY1305_SHA256 (TLS1.3 only)
```

**CSUSDefaultWithSoftFail** This contains the same cyphier suite set as CSUSDefault. The difference to CSUS↩ Default, is the beaviour of the revocation check. For CSUSDefaultWithSoftFail the revocation check will use the "soft fail" schema.

since 1.2.0

**CSUSLegacy** This defines the set which contains biggest set of cipher suites. This is intended for all use case where the access to the internet is needed. Use cases are online radio, which is using all possible server, which are not under the control of MOD.

```
          - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
          - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
          - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
          - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
          - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
          - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
          - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
          - TLS_AES_128_GCM_SHA256 (TLS1.3 only)
          - TLS_AES_256_GCM_SHA384 (TLS1.3 only)
          - TLS_CHACHA20_POLY1305_SHA256 (TLS1.3 only)
          - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
          - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
          - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
          - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
          - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
          - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
          - TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
          - TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
          - TLS_RSA_WITH_AES_128_GCM_SHA256
          - TLS_RSA_WITH_AES_256_GCM_SHA384
          - TLS_RSA_WITH_AES_128_CBC_SHA256
          - TLS_RSA_WITH_AES_256_CBC_SHA256
          - TLS_RSA_WITH_AES_128_CBC_SHA
          - TLS_RSA_WITH_AES_256_CBC_SHA
          - TLS_RSA_WITH_3DES_EDE_CBC_SHA
          - TLS_AES_128_CCM_SHA256 (TLS1.3 only)
```

**CSUSLongtermSecure** This is most restrictive, this will only contain the cipher suites with high key length. It is expected that these cipher suites are most secured for the next years.

```
  - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
  - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
  - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_AES_256_GCM_SHA384 (TLS1.3 only)
  - TLS_CHACHA20_POLY1305_SHA256 (TLS1.3 only)
```

**CSUSIanaRecommended** This is the list of cipher suites which are recommended by IANA.

```
                    - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
                    - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
                    - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
                    - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
                    - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
                    - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
                    - TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
                    - TLS_AES_128_GCM_SHA256 (TLS1.3 only)
                    - TLS_AES_256_GCM_SHA384 (TLS1.3 only)
                    - TLS_CHACHA20_POLY1305_SHA256 (TLS1.3 only)
                    - TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
                    - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
                    - TLS_AES_128_CCM_SHA256 (TLS1.3 only)
```

**Since**

1.1.0

**Enumerator**

| | |
|---|---|
| CSUSDefault | |
| CSUSLegacy | |
| CSUSLongtermSecure | |
| CSUSIanaRecommended | |
| CSUSDefaultWithSoftFail | |
| CSUSEndOfEnum | |

Definition at line 355 of file TLSApiTypes.h.

#### 7.2.2.7 TLSDropStatus

```
enum vwg::tls::TLSDropStatus : UInt32
```

**Enumerator**

| | |
|---|---|
| TLSDROP_SECURED | |
| TLSDROP_DROPPED | |
| TLSDROP_REQUESTED | |
| TLSDROP_SEND_LOCKED | |
| TLSDROP_PERFORMED | |

Definition at line 50 of file TLSSession.h.

#### 7.2.2.8 TLSDropSuppot

```
enum vwg::tls::TLSDropSuppot : UInt32
```

**Enumerator**

| | |
|---|---|
| TLS_NOT_DROPABLE | |
| TLS_DROPABLE | |

Definition at line 988 of file TLSApiTypes.h.

#### 7.2.2.9 TLSReturnCodes

```
enum vwg::tls::TLSReturnCodes : Int32
```

**Enumerator**

| | |
|---|---|
| RC_TLS_SUCCESSFUL | |
| RC_TLS_INIT_FAILED | |
| RC_TLS_CONNECT_FAILED | |
| RC_TLS_ACCEPT_FAILED | |
| RC_TLS_INVALID_DOMAIN | This shall be returned when the domain name provided by the application is not valid according to the sSOA domain name specification. |
| RC_TLS_KEY_MISSING | this shall be returned in case there is no valid key for the provider consumer connection defined. |

**Enumerator**

| | |
|---|---|
| RC_TLS_KEY_ERROR | This shall be returned in case there will be a error to derive the session key from the PSK key. This error shall cover all the errors due to the trust zone handling. The library shall cover all diagnostic related requirements and created according trace information. |
| RC_TLS_USAGE_AFTER_CLEANUP | This error shall be returned when the library functions/class are used after calling the cleanup method. |
| RC_TLS_IO_ERROR | This shall be returned due to IO/protocol error. |
| RC_TLS_WOULD_BLOCK_READ | This shall be returned in non-blocking mode when the operation would block. The caller is advised to check the error code and repeat the operation when the socket is ready for read/write, according to the error code. |
| RC_TLS_WOULD_BLOCK_WRITE | |
| RC_TLS_PEER_CLOSED | This shall be returned due to peer unexpectedly closing the connection. |
| RC_TLS_AUTHENTIC_TIMECHECK_FAILED | This shall be returned due to authentic time check failed. |
| RC_TLS_MAX_PERMITTED_DEVIATION | This shall be returned if $\vert$permitted deviation (check time member)$\vert$ $>=$ MAX_PERMITTED_DEVIATION. |
| RC_TLS_SEND_AFTER_SHUTDOWN | This shall be returned due to attempting to send after shutdown. |
| RC_TLS_INVALID_IP | this will be returned, an invalid IP address is given by the user and the IP address validation failed. |
| RC_TLS_DROPPING_NOTSUPPORTED | |
| RC_TLS_DROPPING_FAILED | |
| RC_TLS_PUBLIC_KEY_PINNING_FAILED | |
| RC_TLS_UNEXPECTED_MESSAGE | |
| RC_TLS_BAD_RECORD_MAC | |
| RC_TLS_RECORD_OVERFLOW | |
| RC_TLS_DECOMPRESSION_FAILURE | |
| RC_TLS_HANDSHAKE_FAILURE | |
| RC_TLS_BAD_CERTIFICATE | |
| RC_TLS_UNSUPPORTED_CERTIFICATE | |
| RC_TLS_CERTIFICATE_REVOKED | |
| RC_TLS_CERTIFICATE_EXPIRED | |
| RC_TLS_CERTIFICATE_UNKNOWN | |
| RC_TLS_ILLEGAL_PARAMETER | |
| RC_TLS_UNKOWN_CA | |
| RC_TLS_UNKNOWN_CA | |
| RC_TLS_ACCESS_DENIED | |
| RC_TLS_DECODE_ERROR | |
| RC_TLS_DECRYPT_ERROR | |
| RC_TLS_PROTOCOL_VERSION | |
| RC_TLS_INSUFFICIENT_SECURITY | |
| RC_TLS_NO_RENEGOTIATION | |
| RC_TLS_UNSUPPORTED_EXTENSION | |
| RC_TLS_CERTIFICATE_UNOBTAINABLE | |
| RC_TLS_UNRECOGNIZED_NAME | |

**Enumerator**

| | |
|---|---|
| RC_TLS_BAD_CERTIFICATE_STATUS_RESPO↩ NSE | |
| RC_TLS_BAD_CERTIFICATE_HASH_VALUE | |
| RC_TLS_NO_APPLICATION_PROTOCOL | This is used for the ALPN extension, for details please see `https://tools.ietf.↩ org/rfc/rfc7301.txt` chapter 3.2. In the event that the server supports no protocols that the client advertises, than this error is returned. **Since** 1.1.0 |
| RC_TLS_TEE_ACCESS_ERROR | The TEE report an error while performing the operation. This can be either permission problem or other TEE specific problems. |
| RC_TLS_CERTSTORE_NOT_FOUND | The TEE does not contain a certificate store (aka “truststore” aka “root certificate bundle” in other docs) for given certStoreId. Depending on the library implementation and the used SSL implementation the message RC_TLS_UNKOWN_CA can be returned. |
| RC_TLS_UNKNOWN_CLIENT_CERTIFICATE_S↩ ET_ID | The given certificate set id is unknown. it shall be one of the permitted values CLINET_CERTICATE_SET_BASE = "BASE" or CLINET_CERTICATE_SET_VKMS = "VKMS" or the project specific. |
| RC_TLS_CLIENT_CERTIFICATE_SET_IDERROR | The TEE does not contain client certificate set and/or private key for given clientCertificateSetID. Depending on the library implementation and the used SSL implementation the message RC_TLS_NO_CERTIFICATE_RESERVED can be returned. |
| RC_TLS_PROGRAMMING_ERROR_RESULT | This error will be present if an invalid error message is created by the library. This will indicate a programming error of the library. |

Definition at line 41 of file TLSReturnCodes.h.

### 7.2.2.10 TLSSessionStatus

```
enum vwg::tls::TLSSessionStatus : UInt32
```

Defines the possible status values of the session.

**Enumerator**

| | |
|---|---|
| TLSSESSION_SECURED | TLSSESSION_SECURED shall be the default case. This indicates that the connection is active an security is active. |
| TLSSESSION_UNSECURED | TLSSESSION_UNSECURED is only be supported in case the TLS can be dropped. This indicates that the connection is active but security was dropped. |
| TLSSESSION_BROKEN | TLSSESSION_BROKEN indicates that a connection is not working anymore, due to errors. |
| TLSSESSION_CLOSED | TLSSESSION_CLOSED indicates that a connection is closed. |

Definition at line 61 of file TLSSession.h.

## 7.2.3 Function Documentation

### 7.2.3.1 ApiVersion()

```
const ApiVersionType vwg::tls::ApiVersion (
             "TLS_API_1.3"  )
```

### 7.2.3.2 cleanupTLSLib()

```
void vwg::tls::cleanupTLSLib ( )
```

Use this method to cleanup the implementation. This can be used to cleanup the TLS library (e.g. Wolf SSL or Botan SSL). after this the ITLSSocketFactory will not return any socket instance.

### 7.2.3.3 initTLSLib()

```
ITLSSocketFactoryResult vwg::tls::initTLSLib ( )
```

This is the entry point for the library. This will return the Socket factory when all initialization needed are successfully performed. These is basically initialization of:

- the TLS/SSL library
- communication to the trust zone

**Returns**

the TLSSocketFactory or an error code.

## 7.2.4 Variable Documentation

### 7.2.4.1 ALPN_ANY

```
const static AlpnMode vwg::tls::ALPN_ANY = AlpnMode(std::vector<IANAProtocol>{IANAProtocol::↩
HTTP2, IANAProtocol::HTTP})  [static]
```

Defines all supported ALPN.

Definition at line 252 of file TLSApiTypes.h.

#### 7.2.4.2 ALPN_DEFAULT

```
const static AlpnMode vwg::tls::ALPN_DEFAULT = AlpnMode(std::vector<IANAProtocol>{HTTP})
[static]
```

Defines the default ALPN.

Definition at line 242 of file TLSApiTypes.h.

#### 7.2.4.3 ALPN_HTTP2

```
const static AlpnMode vwg::tls::ALPN_HTTP2 = AlpnMode(std::vector<IANAProtocol>{IANAProtocol↩
::HTTP2}) [static]
```

Defines HTTP2 ALPN.

Definition at line 247 of file TLSApiTypes.h.

#### 7.2.4.4 ALPN_OFF

```
const static AlpnMode vwg::tls::ALPN_OFF = AlpnMode(std::vector<IANAProtocol>{NONE}) [static]
```

Defines that ALPN is off and the protocol is undecided, this is identical to TLS without any ALPN support.

Definition at line 237 of file TLSApiTypes.h.

#### 7.2.4.5 CHECK_TIME_OFF

```
const static TimeCheckTime vwg::tls::CHECK_TIME_OFF = {0, 0} [static]
```

Defines that time check is not required.

Definition at line 142 of file TLSApiTypes.h.

#### 7.2.4.6 CLINET_CERTICATE_SET_BASE

```
const ClientCertificateSetID vwg::tls::CLINET_CERTICATE_SET_BASE = "BASE"
```

Definition at line 50 of file TLSSocketFactory.h.

#### 7.2.4.7 CSUSDefaultStr

```
const static std::string vwg::tls::CSUSDefaultStr = "default"  [static]
```

Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.

**Since**

1.1.0

Definition at line 372 of file TLSApiTypes.h.

#### 7.2.4.8 CSUSDefaulWithSoftFailtStr

```
const static std::string vwg::tls::CSUSDefaulWithSoftFailtStr = "default_with_soft_fail"  [static]
```

Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.

**Since**

1.2.0

Definition at line 382 of file TLSApiTypes.h.

Referenced by vwg::tls::TLSConnectionSettings::TLSConnectionSettings().

#### 7.2.4.9 CSUSIanaRecommendedStr

```
const static std::string vwg::tls::CSUSIanaRecommendedStr = "iana_recommended"  [static]
```

Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSIanaRecommended for more detail.

**Since**

1.1.0

Definition at line 410 of file TLSApiTypes.h.

Referenced by vwg::tls::TLSConnectionSettings::TLSConnectionSettings().

#### 7.2.4.10 CSUSLegacyStr

```
const static std::string vwg::tls::CSUSLegacyStr = "legacy"  [static]
```

Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLegacy for more detail.

**Since**

1.1.0

Definition at line 392 of file TLSApiTypes.h.

Referenced by vwg::tls::TLSConnectionSettings::TLSConnectionSettings().

#### 7.2.4.11 CSUSLongtermSecureStr

```
const static std::string vwg::tls::CSUSLongtermSecureStr = "longterm_secure"  [static]
```

Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLongtermSecure for more detail.

**Since**

1.1.0

Definition at line 401 of file TLSApiTypes.h.

Referenced by vwg::tls::TLSConnectionSettings::TLSConnectionSettings().

#### 7.2.4.12 DEFAULT_OCSP_ONLINE_TIMEOUT_MS

```
const static UInt32 vwg::tls::DEFAULT_OCSP_ONLINE_TIMEOUT_MS = 30000  [static]
```

Defines a default OCSP timeout in milliseconds.

Definition at line 147 of file TLSApiTypes.h.

#### 7.2.4.13 MAX_PERMITTED_DEVIATION

```
const static unsigned int vwg::tls::MAX_PERMITTED_DEVIATION = 86400  [static]
```

Defines the maximum permitted deviation of |expectedTime - system_time.now()|. since 1.1.0.

Definition at line 110 of file TLSApiTypes.h.

#### 7.2.4.14 MODE_ASYNC

```
const UInt32 vwg::tls::MODE_ASYNC = 1
```

Definition at line 968 of file TLSApiTypes.h.

#### 7.2.4.15 MODE_BLOCKING

```
const UInt32 vwg::tls::MODE_BLOCKING = 0
```

Definition at line 967 of file TLSApiTypes.h.

#### 7.2.4.16 TLS_EOF

```
const int vwg::tls::TLS_EOF = 0
```

Defines the EOF value 0 in case that the connection is closed. This can happen if a closed on a socket is made and there are pending receive and send. Please be aware of that EOF is defined as -1.

Definition at line 91 of file TLSSession.h.

## 7.3 vwg::types Namespace Reference

### Typedefs

- using Boolean = bool
- typedef std::uint8_t UInt8
- typedef std::uint16_t UInt16
- typedef std::uint32_t UInt32
- typedef std::uint64_t UInt64
- typedef std::int8_t Int8
- typedef std::int16_t Int16
- typedef std::int32_t Int32
- typedef std::int64_t Int64
- using Byte = UInt8
- using UUID = std::array< UInt8, 16 >

### 7.3.1 Typedef Documentation

#### 7.3.1.1 Boolean

```
using vwg::types::Boolean = typedef bool
```

Definition at line 39 of file vwgtypes.h.

#### 7.3.1.2 Byte

```
using vwg::types::Byte = typedef UInt8
```

Definition at line 51 of file vwgtypes.h.

#### 7.3.1.3 Int16

```
typedef std::int16_t vwg::types::Int16
```

Definition at line 47 of file vwgtypes.h.

#### 7.3.1.4 Int32

```
typedef std::int32_t vwg::types::Int32
```

Definition at line 48 of file vwgtypes.h.

#### 7.3.1.5 Int64

```
typedef std::int64_t vwg::types::Int64
```

Definition at line 49 of file vwgtypes.h.

#### 7.3.1.6 Int8

```
typedef std::int8_t vwg::types::Int8
```

Definition at line 46 of file vwgtypes.h.

#### 7.3.1.7 UInt16

```
typedef std::uint16_t vwg::types::UInt16
```

Definition at line 41 of file vwgtypes.h.

#### 7.3.1.8 UInt32

```
typedef std::uint32_t vwg::types::UInt32
```

Definition at line 42 of file vwgtypes.h.

#### 7.3.1.9 UInt64

```
typedef std::uint64_t vwg::types::UInt64
```

Definition at line 43 of file vwgtypes.h.

#### 7.3.1.10 UInt8

```
typedef std::uint8_t vwg::types::UInt8
```

Definition at line 40 of file vwgtypes.h.

#### 7.3.1.11 UUID

```
using vwg::types::UUID = typedef std::array<UInt8, 16>
```

Definition at line 54 of file vwgtypes.h.

# Chapter 8

# Class Documentation

## 8.1 vwg::tls::AlpnMode Class Reference

A setting container for ALPN supporting. There are basically three modes possible:

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- AlpnMode (const std::vector< std::string > &userDefinedAlpnSetting)

  *Constructor.*
- AlpnMode (const std::vector< IANAProtocol > &supportedProtocols)

  *Constructor.*
- virtual ~AlpnMode ()=default
- bool userDefinedALPNisUsed () const

  *Gets a boolean that tells if the ALPN setting is defined.*
- const std::vector< IANAProtocol > & getSupportedProtocols () const

  *Gets Supported IANA protocols.*
- const std::vector< std::string > & getUserDefinedAlpnSetting () const

  *Gets an ALPN setting.*

### Private Attributes

- bool m_userDefinedALPNisUsed
- std::vector< std::string > m_userDefinedAlpnSetting
- std::vector< IANAProtocol > m_supportedProtocols

### 8.1.1 Detailed Description

A setting container for ALPN supporting. There are basically three modes possible:

a) ALPN can be provided as a user defined string list. In this case the protocol list is passed to the TLS library without no additional check. This means that an invalid value can cause unexpected errors, if an invalid string is used. The given string must be complaint to chapter "3.1. The Application-Layer Protocol Negotiation Extension" of RFC 7301.

b) ALPN parameter can be provided by a vector of pre defined enum's and constant of the ALPN mode type.

c) If an empty list vector is used, then ALPN is unused in the client hello. Basically this shall be identical like the the usage of HTTP protocol, but it can be different if the server is not supporting ALPN.

**Since**

1.1.0

Definition at line 166 of file TLSApiTypes.h.

### 8.1.2 Constructor & Destructor Documentation

#### 8.1.2.1 AlpnMode() [1/2]

```
vwg::tls::AlpnMode::AlpnMode (
            const std::vector< std::string > & userDefinedAlpnSetting )  [inline], [explicit]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *userDefinedAlpnSetting* | ALPN setting. |

Definition at line 174 of file TLSApiTypes.h.

#### 8.1.2.2 AlpnMode() [2/2]

```
vwg::tls::AlpnMode::AlpnMode (
            const std::vector< IANAProtocol > & supportedProtocols )  [inline], [explicit]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *supportedProtocols* | Supported IANA protocols. |

Definition at line 185 of file TLSApiTypes.h.

#### 8.1.2.3 ∼AlpnMode()

```
virtual vwg::tls::AlpnMode::∼AlpnMode ( )  [virtual], [default]
```

### 8.1.3 Member Function Documentation

#### 8.1.3.1 getSupportedProtocols()

```
const std::vector<IANAProtocol>& vwg::tls::AlpnMode::getSupportedProtocols ( ) const  [inline]
```

Gets Supported IANA protocols.

**Returns**

Supported IANA protocols.

Definition at line 211 of file TLSApiTypes.h.

#### 8.1.3.2 getUserDefinedAlpnSetting()

```
const std::vector<std::string>& vwg::tls::AlpnMode::getUserDefinedAlpnSetting ( ) const  [inline]
```

Gets an ALPN setting.

**Returns**

ALPN setting.

Definition at line 222 of file TLSApiTypes.h.

#### 8.1.3.3 userDefinedALPNisUsed()

```
bool vwg::tls::AlpnMode::userDefinedALPNisUsed ( ) const  [inline]
```

Gets a boolean that tells if the ALPN setting is defined.

**Returns**

true if ALPN setting is defined, otherwise false.

Definition at line 200 of file TLSApiTypes.h.

### 8.1.4 Member Data Documentation

#### 8.1.4.1 m_supportedProtocols

```
std::vector<IANAProtocol> vwg::tls::AlpnMode::m_supportedProtocols [private]
```

Definition at line 230 of file TLSApiTypes.h.

#### 8.1.4.2 m_userDefinedALPNisUsed

```
bool vwg::tls::AlpnMode::m_userDefinedALPNisUsed [private]
```

Definition at line 228 of file TLSApiTypes.h.

#### 8.1.4.3 m_userDefinedAlpnSetting

```
std::vector<std::string> vwg::tls::AlpnMode::m_userDefinedAlpnSetting [private]
```

Definition at line 229 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.2 vwg::tls::IANAProtocolFunction Class Reference

This class contains some helper methods when conversion from the IANAProtocol enum value to Protocol name.

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- IANAProtocolFunction ()=default
- ∼IANAProtocolFunction ()=default
- bool toIANAProtocolName (const IANAProtocol &protocol, std::string &oProtocolName)

  *Converts IANAProtocol enum value to Protocol name.*

### Public Attributes

- const std::string ProtocolNameHTTP = "http/1.1"
- const std::string ProtocolNameHTTP2 = "h2"

### 8.2.1 Detailed Description

This class contains some helper methods when conversion from the IANAProtocol enum value to Protocol name.

**Since**

1.1.0

Definition at line 71 of file TLSApiTypes.h.

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 IANAProtocolFunction()

```
vwg::tls::IANAProtocolFunction::IANAProtocolFunction ( )  [default]
```

#### 8.2.2.2 ∼IANAProtocolFunction()

```
vwg::tls::IANAProtocolFunction::∼IANAProtocolFunction ( )  [default]
```

### 8.2.3 Member Function Documentation

#### 8.2.3.1 toIANAProtocolName()

```
bool vwg::tls::IANAProtocolFunction::toIANAProtocolName (
            const IANAProtocol & protocol,
            std::string & oProtocolName )  [inline]
```

Converts IANAProtocol enum value to Protocol name.

**Parameters**

| | | |
|---|---|---|
| in | *protocol* | IANA protocol enum value to be converted. |
| out | *oProtocolName* | should be contained the protocol name if converted successfully. |

**Returns**

true if converted successfully, false otherwise.

Definition at line 89 of file TLSApiTypes.h.

References vwg::tls::HTTP, and vwg::tls::HTTP2.

### 8.2.4 Member Data Documentation

#### 8.2.4.1 ProtocolNameHTTP

```
const std::string vwg::tls::IANAProtocolFunction::ProtocolNameHTTP = "http/1.1"
```

Definition at line 77 of file TLSApiTypes.h.

#### 8.2.4.2 ProtocolNameHTTP2

```
const std::string vwg::tls::IANAProtocolFunction::ProtocolNameHTTP2 = "h2"
```

Definition at line 78 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.3 vwg::tls::IInetAddress Class Reference

Representation an interface of an IP address. Basically this will give you an immutable IP address interface.

```
#include <InetAddress.h>
```

### Public Member Functions

- IInetAddress ()
- virtual ∼IInetAddress ()=default
- virtual Boolean isIPv6 ()=0

  *Checks if this a valid IPv6 address.*
- virtual Boolean isIPv4 ()=0

  *Checks if this is a valid IPv6 address.*
- virtual std::string toString ()=0

  *Makes a sting representation of the IP address.*
- virtual Boolean isValid ()=0

  *Checks if this is a valid IP address. basically this will always be true, because the factory InetAddressFactory will only return valid IInetAddress objects.*
- virtual UInt32 validate ()=0

  *Starts the IP address validation. this is maybe not needed by the application.*
- virtual sa_family_t getSaFamily ()=0

  *This gives the sa_family_t of the IP address. this belongs to the socket API, and will be used by the implementation of the library when creating the network socket. see also* `http://man7.↩ org/linux/man-pages/man2/bind.2.html` *for the SaFamily.*
- virtual uint8_t $*$ getAddr ()

  *get the IP address.*

## Protected Attributes

- uint8_t m_addr [16]

## 8.3.1 Detailed Description

Representation an interface of an IP address. Basically this will give you an immutable IP address interface.

Definition at line 54 of file InetAddress.h.

## 8.3.2 Constructor & Destructor Documentation

### 8.3.2.1 IInetAddress()

```
vwg::tls::IInetAddress::IInetAddress ( )  [inline]
```

Definition at line 57 of file InetAddress.h.

### 8.3.2.2 ∼IInetAddress()

```
virtual vwg::tls::IInetAddress::∼IInetAddress ( )  [virtual], [default]
```

## 8.3.3 Member Function Documentation

### 8.3.3.1 getAddr()

```
virtual uint8_t* vwg::tls::IInetAddress::getAddr ( )  [inline], [virtual]
```

get the IP address.

**Returns**

IP address

Definition at line 114 of file InetAddress.h.

#### 8.3.3.2 getSaFamily()

```
virtual sa_family_t vwg::tls::IInetAddress::getSaFamily ( ) [pure virtual]
```

This gives the sa_family_t of the IP address. this belongs to the socket API, and will be used by the implementation of the library when creating the network socket. see also `http://man7.org/linux/man-pages/man2/bind.2.html` for the SaFamily.

**Returns**

SaFamily of the IP address.

#### 8.3.3.3 isIPv4()

```
virtual Boolean vwg::tls::IInetAddress::isIPv4 ( ) [pure virtual]
```

Checks if this is a valid IPv6 address.

**Returns**

true if this is a valid IPv6 address

#### 8.3.3.4 isIPv6()

```
virtual Boolean vwg::tls::IInetAddress::isIPv6 ( ) [pure virtual]
```

Checks if this a valid IPv6 address.

**Returns**

true if this is a valid IPv6 address.

#### 8.3.3.5 isValid()

```
virtual Boolean vwg::tls::IInetAddress::isValid ( ) [pure virtual]
```

Checks if this is a valid IP address. basically this will always be true, because the factory InetAddressFactory will only return valid IInetAddress objects.

**Returns**

string representation of the IP address.

#### 8.3.3.6 toString()

```
virtual std::string vwg::tls::IInetAddress::toString ( )  [pure virtual]
```

Makes a sting representation of the IP address.

**Returns**

string representation of the IP address

#### 8.3.3.7 validate()

```
virtual UInt32 vwg::tls::IInetAddress::validate ( )  [pure virtual]
```

Starts the IP address validation. this is maybe not needed by the application.

**Returns**

an underlying error code.

### 8.3.4 Member Data Documentation

#### 8.3.4.1 m_addr

```
uint8_t vwg::tls::IInetAddress::m_addr[16]  [protected]
```

Definition at line 120 of file InetAddress.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/InetAddress.h

## 8.4 vwg::tls::InetAddressFactory Class Reference

This a definition of a the factory to create instances of the IInetAddress. The supplier has to provide the implementation of the static methods by this class. Basically there is no need to create an instance of this class.

```
#include <InetAddress.h>
```

## Static Public Member Functions

- static IInetAddressResult makeIPAddress (const std::string inetAddr)

  *Factory method to create a valid IP IPv4 / IPv6 Address object. The given string will be validated and an IInetAddress is returned if valid.*
- static IInetAddressResult makeIPAddress (const char ∗inetAdd)

  *Factory method to create a valid IP IPv4 / IPv6 Address object. The given string will be validated and an IInetAddress is returned if valid.*

## Private Member Functions

- InetAddressFactory ()=default

### 8.4.1 Detailed Description

This a definition of a the factory to create instances of the IInetAddress. The supplier has to provide the implementation of the static methods by this class. Basically there is no need to create an instance of this class.

Definition at line 139 of file InetAddress.h.

### 8.4.2 Constructor & Destructor Documentation

#### 8.4.2.1 InetAddressFactory()

```
vwg::tls::InetAddressFactory::InetAddressFactory ( )  [private], [default]
```

### 8.4.3 Member Function Documentation

#### 8.4.3.1 makeIPAddress() [1/2]

```
static IInetAddressResult vwg::tls::InetAddressFactory::makeIPAddress (
            const char * inetAdd )  [static]
```

Factory method to create a valid IP IPv4 / IPv6 Address object. The given string will be validated and an IInetAddress is returned if valid.

**Parameters**

| | | |
|---|---|---|
| in | *inetAddr* | a string which defines a IP address. e.g "127.0.0.1" |

**Returns**

a valid IInetAddress or an error if not valid.

### 8.4.3.2 makeIPAddress() [2/2]

```
static IInetAddressResult vwg::tls::InetAddressFactory::makeIPAddress (
            const std::string inetAddr )  [static]
```

Factory method to create a valid IP IPv4 / IPv6 Address object. The given string will be validated and an IInetAddress is returned if valid.

**Parameters**

| | | |
|---|---|---|
| in | *inetAddr* | a string which defines an IP address. e.g "::2" or "4:6:7...". |

**Returns**

a valid IInetAddress or an error if not valid.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/InetAddress.h

## 8.5 vwg::tls::IOStream Class Reference

Representation an interface of an I/O stream. Can read, write and close.

```
#include <IOStream.h>
```

### Public Member Functions

- IOStream ()=default
- virtual ∼IOStream ()=default
- virtual int32_t receive (void ∗buf, uint32_t len)=0

  *Reads from the stream, up to len bytes. The method blocks until data are available, unless in non-blocking mode.*
- virtual int32_t send (const void ∗buf, uint32_t len)=0

  *Writes into the stream. The method blocks until data are sent, unless in non-blocking mode.*
- virtual void close ()=0

  *Closes the stream.*
- virtual bool isOpen ()=0

  *Check whether the stream is open or not.*
- virtual bool isClosed ()=0

  *Check whether the stream is open or not.*

### 8.5.1 Detailed Description

Representation an interface of an I/O stream. Can read, write and close.

Definition at line 52 of file IOStream.h.

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 IOStream()

```
vwg::tls::IOStream::IOStream ( )  [default]
```

#### 8.5.2.2 ∼IOStream()

```
virtual vwg::tls::IOStream::∼IOStream ( )  [virtual], [default]
```

### 8.5.3 Member Function Documentation

#### 8.5.3.1 close()

```
virtual void vwg::tls::IOStream::close ( )  [pure virtual]
```

Closes the stream.

#### 8.5.3.2 isClosed()

```
virtual bool vwg::tls::IOStream::isClosed ( )  [pure virtual]
```

Check whether the stream is open or not.

**Returns**

true if the stream is closed, false otherwise

#### 8.5.3.3 isOpen()

```
virtual bool vwg::tls::IOStream::isOpen ( )  [pure virtual]
```

Check whether the stream is open or not.

**Returns**

true if the stream is open, false otherwise

#### 8.5.3.4 receive()

```
virtual int32_t vwg::tls::IOStream::receive (
            void * buf,
            uint32_t len )  [pure virtual]
```

Reads from the stream, up to len bytes. The method blocks until data are available, unless in non-blocking mode.

**Parameters**

| | | |
|---|---|---|
| in | *buf* | the buffer to read into |
| in | *len* | length of the buffer, in bytes |

**Returns**

the number of bytes received or the relevant StreamReturnCode error code

#### 8.5.3.5 send()

```
virtual int32_t vwg::tls::IOStream::send (
            const void * buf,
            uint32_t len )  [pure virtual]
```

Writes into the stream. The method blocks until data are sent, unless in non-blocking mode.

**Parameters**

| | | |
|---|---|---|
| in | *buf* | the buffer to write |
| in | *len* | length of the buffer, in bytes |

**Returns**

the number of bytes sent or the relevant StreamReturnCode error code

The documentation for this class was generated from the following file:

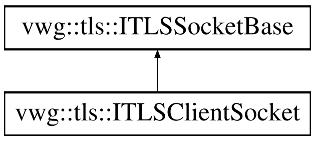
- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/IOStream.h

## 8.6 vwg::tls::ITLSClientSocket Class Reference

Server TLS-PSK aware client socket interface. This interface must be implemented by the supplier.

```
#include <TLSSockets.h>
```

Inheritance diagram for vwg::tls::ITLSClientSocket:



### Public Member Functions

- ITLSClientSocket ()=default
- virtual ∼ITLSClientSocket ()=default
- virtual TLSResult< std::shared_ptr< ITLSSessionEndpoint > > connect ()=0

    *a client shall call this method in to get connected to the server. This will do all underling operations like*
- virtual void setSoTimeout (Int32 timeout)=0

    *Changes the default socket timeout, SO_RCVTIMEO and SO_SNDTIMEO options, according to `https://linux.die.net/man/3/setsockopt`.*
- virtual int getSocketFD ()=0

    *Gets the network socket file descriptor.*

### Additional Inherited Members

### 8.6.1 Detailed Description

Server TLS-PSK aware client socket interface. This interface must be implemented by the supplier.

For TCP based communication make a connect call to retrieve a connection to the server. The server connection is represented by a TLSSession where one can read and write the data. Within the connect call all needed operations are performed. This includes:

- make the TLS or TLS-PSK handshake (see `https://tools.ietf.org/html/rfc4279`).
- derive the pre shared key from the SSOA domain name.
- derive the session key from the pre shared key stored within the trust zone.

Definition at line 108 of file TLSSockets.h.

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 ITLSClientSocket()

```
vwg::tls::ITLSClientSocket::ITLSClientSocket ( )  [default]
```

#### 8.6.2.2 ∼ITLSClientSocket()

```
virtual vwg::tls::ITLSClientSocket::∼ITLSClientSocket ( )  [virtual], [default]
```

### 8.6.3 Member Function Documentation

#### 8.6.3.1 connect()

```
virtual TLSResult<std::shared_ptr<ITLSSessionEndpoint> > vwg::tls::ITLSClientSocket::connect
( )  [pure virtual]
```

a client shall call this method in to get connected to the server. This will do all underling operations like

- make the TLS or TLS-PSK handshake (see `https://tools.ietf.org/html/rfc4279`)
- derive the pre shared key from the SSOA domain name
- derive the session key from the pre shared key stored within the trust zone.

**Returns**

an ITLSSessionEndpoint instance when operation was successful, otherwise an error code is delivered.

#### 8.6.3.2 getSocketFD()

```
virtual int vwg::tls::ITLSClientSocket::getSocketFD ( )  [pure virtual]
```

Gets the network socket file descriptor.

**Returns**

the network socket file descriptor.

#### 8.6.3.3 setSoTimeout()

```
virtual void vwg::tls::ITLSClientSocket::setSoTimeout (
            Int32 timeout )  [pure virtual]
```

Changes the default socket timeout, SO_RCVTIMEO and SO_SNDTIMEO options, according to `https↩://linux.die.net/man/3/setsockopt`.

**Parameters**

| | | |
|---|---|---|
| `in` | *timeout* | The new socket timeout value in milliseconds. |

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSSockets.h

## 8.7 vwg::tls::ITLSErrorListener Class Reference

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- ITLSErrorListener ()=default
- virtual ∼ITLSErrorListener ()=default
- virtual void errorListener (SPIInetAddress inet, const UInt16 port, const TLSReturnCodes errorCode)=0

### 8.7.1 Detailed Description

Definition at line 1001 of file TLSApiTypes.h.

### 8.7.2 Constructor & Destructor Documentation

#### 8.7.2.1 ITLSErrorListener()

```
vwg::tls::ITLSErrorListener::ITLSErrorListener ( )  [default]
```

#### 8.7.2.2 ∼ITLSErrorListener()

```
virtual vwg::tls::ITLSErrorListener::∼ITLSErrorListener ( )  [virtual], [default]
```

### 8.7.3 Member Function Documentation

#### 8.7.3.1 errorListener()

```
virtual void vwg::tls::ITLSErrorListener::errorListener (
             SPIInetAddress inet,
             const UInt16 port,
             const TLSReturnCodes errorCode )  [pure virtual]
```

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.8 vwg::tls::ITLSOcspHandler Class Reference

This interface defines APIs to process and handle OCSP messages.

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- ITLSOcspHandler ()=default
- virtual ∼ITLSOcspHandler ()=default
- virtual void cacheResponses (const std::vector< TLSOcspCachedResponse > &responses) noexcept=0

  *Cache the OCSP responses.*
- virtual std::future< std::vector< TLSOcspRequestResponse > > processRequests (const std::vector< TLSOcspRequest > &requests) noexcept=0

  *Process the OCSP requests and send them to OCSP Proxy process for further processing.*

### 8.8.1 Detailed Description

This interface defines APIs to process and handle OCSP messages.

Definition at line 735 of file TLSApiTypes.h.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 ITLSOcspHandler()

```
vwg::tls::ITLSOcspHandler::ITLSOcspHandler ( )  [default]
```

#### 8.8.2.2 ∼ITLSOcspHandler()

```
virtual vwg::tls::ITLSOcspHandler::∼ITLSOcspHandler ( )  [virtual], [default]
```

## 8.8.3 Member Function Documentation

### 8.8.3.1 cacheResponses()

```
virtual void vwg::tls::ITLSOcspHandler::cacheResponses (
            const std::vector< TLSOcspCachedResponse > & responses )  [pure virtual], [noexcept]
```

Cache the OCSP responses.

**Note**

This method shall be executed in a new thread context.

This method serialize each OCSP response, send it over to OCSP Proxy process via IPC mechanism to save it in cache. This method shall be called after:

- "processRequest" execution.
- full validation and verification of the OCSP responses.

**Parameters**

| | | |
|---|---|---|
| in | *responses* | Vector of OCSP responses to cache. |

### 8.8.3.2 processRequests()

```
virtual std::future<std::vector<TLSOcspRequestResponse> > vwg::tls::ITLSOcspHandler::process←
Requests (
            const std::vector< TLSOcspRequest > & requests )  [pure virtual], [noexcept]
```

Process the OCSP requests and send them to OCSP Proxy process for further processing.

**Note**

This method shall be executed in a new thread context The returned vector shall contain an OCSP request response object FOR EACH ocsp request that was in the requests vector. In case of an error for specific OCSP request handling you shall create an OCSP request response object with the second constructor that builds object by the unique ID only. The order of the responses vector shall be the same as the order in the requests vector.

This method serialize each OCSP requests, send it over to OCSP Proxy process via IPC mechanism to decide whether to send the requests to OCSP responder or to use the responses that already cached.

**Parameters**

| | | |
|---|---|---|
| in | *requests* | Vector of OCSP requests. |

**Returns**

A future that contains a vector of OCSP responses for each OCSP request.

The documentation for this class was generated from the following file:

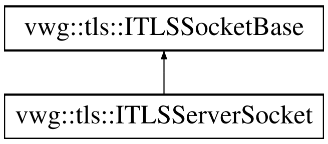
- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.9 vwg::tls::ITLSServerSocket Class Reference

Server TLS-PSK aware server socket interface. This interface must be implemented by the supplier.

`#include <TLSSockets.h>`

Inheritance diagram for vwg::tls::ITLSServerSocket:



### Public Member Functions

- ITLSServerSocket ()=default
- virtual ∼ITLSServerSocket ()=default
- virtual TLSSessionEndpointResult accept ()=0

  *This is a blocking call for the server implementation to wait until the client will get a connection. The server may fork several thread to handle each client in an individual thread. This accept covers all needed operations like.*
- virtual void setSoTimeout (Int32 timeout)=0

  *Sets the socket timeout.*
- virtual int getSocketFD ()=0

  *Gets the network socket file descriptor.*

### Additional Inherited Members

### 8.9.1 Detailed Description

Server TLS-PSK aware server socket interface. This interface must be implemented by the supplier.

For TCP based communication make an accept call to retrieve a connection to the client. The client connection is represented by a TLSSession where one can read and write the data. Within the accept call all needed operations are performed. This includes:

- accept the network connection
- make the TLS or TLS-PSK handshake (see `https://tools.ietf.org/html/rfc4279`)
- derive the pre shared key from the SSOA domain name
- derive the session key from the pre shared key stored within the trust zone.

Definition at line 59 of file TLSSockets.h.

## 8.9.2 Constructor & Destructor Documentation

### 8.9.2.1 ITLSServerSocket()

```
vwg::tls::ITLSServerSocket::ITLSServerSocket ( )  [default]
```

### 8.9.2.2 ∼ITLSServerSocket()

```
virtual vwg::tls::ITLSServerSocket::∼ITLSServerSocket ( )  [virtual], [default]
```

## 8.9.3 Member Function Documentation

### 8.9.3.1 accept()

```
virtual TLSSessionEndpointResult vwg::tls::ITLSServerSocket::accept ( )  [pure virtual]
```

This is a blocking call for the server implementation to wait until the client will get a connection. The server may fork several thread to handle each client in an individual thread. This accept covers all needed operations like.

- accept the network connection
- make the TLS or TLS-PSK handshake (see `https://tools.ietf.org/html/rfc4279`)
- derive the pre shared key from the SSOA domain name
- derive the session key from the pre shared key stored within the trust zone.

**Returns**

a ITLSSessionEndpoint instance when operation was successful, otherwise an error code is delivered.

### 8.9.3.2 getSocketFD()

```
virtual int vwg::tls::ITLSServerSocket::getSocketFD ( )  [pure virtual]
```

Gets the network socket file descriptor.

**Returns**

the network socket file descriptor.

### 8.9.3.3 setSoTimeout()

```
virtual void vwg::tls::ITLSServerSocket::setSoTimeout (
            Int32 timeout )  [pure virtual]
```

Sets the socket timeout.

**Parameters**

| | | |
|---|---|---|
| in | *timeout* | the new socket timeout value in milliseconds. |

The documentation for this class was generated from the following file:

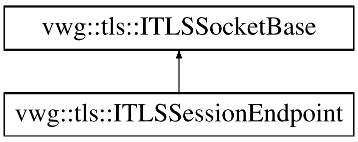
- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSSockets.h

## 8.10 vwg::tls::ITLSSessionEndpoint Class Reference

Represents a communication session between a service provider and a service consumer. This interface must be implemented by the supplier.

```
#include <TLSSession.h>
```

Inheritance diagram for vwg::tls::ITLSSessionEndpoint:



### Public Member Functions

- ITLSSessionEndpoint ()=default
- virtual ~ITLSSessionEndpoint ()=default
- virtual Int32 send (const Byte b[ ], const Int32 len)=0

  *Sends a number of bytes from b[0] to b[len-1].*
- virtual Int32 send (const Byte b[ ], const UInt32 offset, const Int32 len)=0

  *send a number of bytes from b[0+offset] to b[len-1] starting at b at given offset.*
- virtual Int32 flush ()=0

  *Forces to send the bytes. Depending on the underlying socket implementation, it can happen that bytes are still within the send buffer.*
- virtual Int32 available ()=0

  *Checks if bytes are available. The method blocks until data are available.*
- virtual Int32 receive (Byte b[ ], const Int32 len)=0

  *Receive up to len bytes from stream into the buffer starting at b.*
- virtual Int32 receive (Byte b[ ], const UInt32 offset, const Int32 len)=0

  *Receive up to len bytes from stream into the buffer starting at b at given offset.*
- virtual TLSReturnCodes setBlocking (bool blocking)=0

  *Sets blocking/non-blocking mode for the session. Blocking by default.*
- virtual int getSocketFD ()=0

  *Gets the network socket file descriptor.*
- virtual TLSReturnCodes shutdown ()=0

  *Sends a "close notify" alert to the peer. The method blocks, unless in non-blocking mode.*
- virtual std::string getLocalDomainName ()=0

  *Gets the sSOA domain name of the session endpoint.*
- virtual std::string getRemoteDomainName ()=0

*Gets the sSOA domain name of the remote session endpoint.*

- virtual UInt16 getRemotePort ()=0

  *Gets the port of the remote session endpoint .*
- virtual SPIInetAddress getRemoteInetAddress ()=0

  *Gets the inet address of the remote session endpoint .*
- virtual TLSDropStatus getDropState ()=0

  *Gets the current TLS drop status.*
- virtual void setSessionStatusListener (TLSSessionStatusListener listener)=0

  *Sets the listener function (C++-style) for status changes of the session. This overwrites the listener when already set.*
- virtual void setDropStatusListener (TLSDropStatusListener listener)=0

  *Sets the listener function (C++ -style) for drop changes of the session. this overwrites the listener when already set.*

## Additional Inherited Members

## 8.10.1 Detailed Description

Represents a communication session between a service provider and a service consumer. This interface must be implemented by the supplier.

Herewith one user can make send and receive data between the service provider and a service consumer The calls are basically blocking and will return until the operations is performed. This includes:

- network operations.
- Encrypting or decrypting data.

Definition at line 121 of file TLSSession.h.

## 8.10.2 Constructor & Destructor Documentation

### 8.10.2.1 ITLSSessionEndpoint()

```
vwg::tls::ITLSSessionEndpoint::ITLSSessionEndpoint ( )  [default]
```

### 8.10.2.2 ∼ITLSSessionEndpoint()

```
virtual vwg::tls::ITLSSessionEndpoint::∼ITLSSessionEndpoint ( )  [virtual], [default]
```

## 8.10.3 Member Function Documentation

#### 8.10.3.1 available()

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::available ( )  [pure virtual]
```

Checks if bytes are available. The method blocks until data are available.

**Returns**

the number of available bytes.

#### 8.10.3.2 flush()

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::flush ( )  [pure virtual]
```

Forces to send the bytes. Depending on the underlying socket implementation, it can happen that bytes are still within the send buffer.

**Returns**

0 if no error had occurred, or a negative value will indicate an error. The value 0 will indicated that the stream is closed (see TLS_EOF) Use getPendingErrors to retrieve the pending error.

#### 8.10.3.3 getDropState()

```
virtual TLSDropStatus vwg::tls::ITLSSessionEndpoint::getDropState ( )  [pure virtual]
```

Gets the current TLS drop status.

**Returns**

the current TLS drop status of the connection.

#### 8.10.3.4 getLocalDomainName()

```
virtual std::string vwg::tls::ITLSSessionEndpoint::getLocalDomainName ( )  [pure virtual]
```

Gets the sSOA domain name of the session endpoint.

**Returns**

the sSOA domain name of the session endpoint.

#### 8.10.3.5 getRemoteDomainName()

```
virtual std::string vwg::tls::ITLSSessionEndpoint::getRemoteDomainName ( )  [pure virtual]
```

Gets the sSOA domain name of the remote session endpoint.

**Returns**

the sSOA domain name of the remote session endpoint.

#### 8.10.3.6 getRemoteInetAddress()

```
virtual SPIInetAddress vwg::tls::ITLSSessionEndpoint::getRemoteInetAddress ( )  [pure virtual]
```

Gets the inet address of the remote session endpoint .

**Returns**

Gets the inet address of the remote session endpoint .

#### 8.10.3.7 getRemotePort()

```
virtual UInt16 vwg::tls::ITLSSessionEndpoint::getRemotePort ( )  [pure virtual]
```

Gets the port of the remote session endpoint .

**Returns**

Gets the port of the remote session endpoint .

#### 8.10.3.8 getSocketFD()

```
virtual int vwg::tls::ITLSSessionEndpoint::getSocketFD ( )  [pure virtual]
```

Gets the network socket file descriptor.

**Returns**

the network socket file descriptor.

#### 8.10.3.9 receive() [1/2]

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::receive (
            Byte b[],
            const Int32 len )  [pure virtual]
```

Receive up to len bytes from stream into the buffer starting at b.

**Note**

The method blocks until data are available, unless in non-blocking mode. In case of error use getPending←Errors to retrieve the pending error.

**Parameters**

| | | |
|---|---|---|
| in | *b* | buffer to be set with received date. |
| in | *len* | buffer's length, in bytes. |

**Returns**

the number of received bytes, or a negative value will indicate an error. The value 0 will indicated that the stream is closed (see TLS_EOF).

### 8.10.3.10 receive() [2/2]

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::receive (
            Byte b[ ],
            const UInt32 offset,
            const Int32 len )  [pure virtual]
```

Receive up to len bytes from stream into the buffer starting at b at given offset.

**Note**

The method blocks until data are available, unless in non-blocking mode.

**Parameters**

| | | |
|---|---|---|
| in | *b* | buffer to be set with received date. |
| in | *offset* | offset from beginning of the buffer to set data from it. |
| in | *len* | buffer's length, in bytes. |

**Returns**

the number of number of received, or a negative value will indicate an error. The value 0 will indicated that the stream is closed (see TLS_EOF) Use getPendingErrors to retrieve the pending error.

### 8.10.3.11 send() [1/2]

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::send (
            const Byte b[ ],
            const Int32 len )  [pure virtual]
```

Sends a number of bytes from b[0] to b[len-1].

**Note**

The method blocks, unless in non-blocking mode. When an operation is repeated in non-blocking mode, it must be repeated with the same arguments.

**Parameters**

| | | |
|---|---|---|
| `in` | *b* | data buffer for sending data from it. |
| `in` | *len* | buffer's length, in bytes |

**Returns**

the number of send bytes, or a negative value will indicate an error. The value 0 will indicated that the stream is closed (see TLS_EOF) Use getPendingErrors to retrieve the pending error.

#### 8.10.3.12 send() [2/2]

```
virtual Int32 vwg::tls::ITLSSessionEndpoint::send (
             const Byte b[],
             const UInt32 offset,
             const Int32 len )  [pure virtual]
```

send a number of bytes from b[0+offset] to b[len-1] starting at b at given offset.

**Note**

The method blocks, unless in non-blocking mode. When an operation is repeated in non-blocking mode, it must be repeated with the same arguments.

**Parameters**

| | | |
|---|---|---|
| `in` | *b* | data buffer for sending data from it. |
| `in` | *offset* | offset from the beginning of the buffer to send data from it. |
| `in` | *len* | buffer's length, in bytes. |

**Returns**

the number send bytes, or a negative value will indicate an error. The value 0 will indicated that the stream is closed (see TLS_EOF) Use getPendingErrors to retrieve the pending error.

#### 8.10.3.13 setBlocking()

```
virtual TLSReturnCodes vwg::tls::ITLSSessionEndpoint::setBlocking (
             bool blocking )  [pure virtual]
```

Sets blocking/non-blocking mode for the session. Blocking by default.

**Returns**

success indication.

#### 8.10.3.14 setDropStatusListener()

```
virtual void vwg::tls::ITLSSessionEndpoint::setDropStatusListener (
            TLSDropStatusListener listener )  [pure virtual]
```

Sets the listener function (C++ -style) for drop changes of the session. this overwrites the listener when already set.

#### 8.10.3.15 setSessionStatusListener()

```
virtual void vwg::tls::ITLSSessionEndpoint::setSessionStatusListener (
            TLSSessionStatusListener listener )  [pure virtual]
```

Sets the listener function (C++-style) for status changes of the session. This overwrites the listener when already set.

**Parameters**

| | | |
|---|---|---|
| in | *listener* | listener function to be set. |

#### 8.10.3.16 shutdown()

```
virtual TLSReturnCodes vwg::tls::ITLSSessionEndpoint::shutdown ( )  [pure virtual]
```

Sends a "close notify" alert to the peer. The method blocks, unless in non-blocking mode.

**Returns**

success indication.

The documentation for this class was generated from the following file:

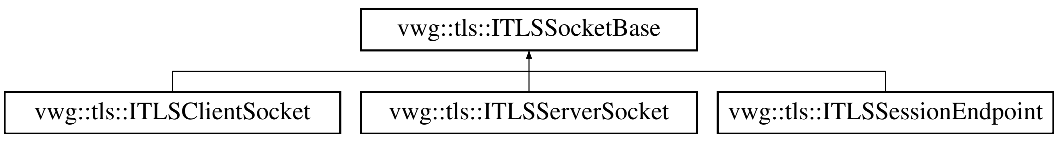
- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSSession.h

## 8.11 vwg::tls::ITLSSocketBase Class Reference

This is an interface which defines a set of operation and features have to be available on each socket and session endpoint.

```
#include <TLSApiTypes.h>
```

Inheritance diagram for vwg::tls::ITLSSocketBase:


vwg::tls::ITLSSocketBase
vwg::tls::ITLSClientSocket
vwg::tls::ITLSServerSocket
vwg::tls::ITLSSessionEndpoint

## Public Member Functions

- ITLSSocketBase ()=default
- virtual ∼ITLSSocketBase ()=default
- Boolean isDatagramSocket ()

  *Gets a boolean that tells if the socket is a Datagram socket.*
- virtual Boolean isConnectionSocket ()=0

  *Gets a boolean that tells if the socket is a stream socket.*
- virtual void close ()=0

  *Closes the underlying socket connection. This will immediately close the connection, all pending data may be lost, therefore one user shall call flush before closing.*
- virtual Boolean isClosed ()=0

  *Checks if the endpoint/connection is closed or not.*
- virtual Boolean isOpen ()=0

  *Checks if the endpoint/connection is closed or not.*
- virtual Boolean isErrorState ()

  *Checks if the endpoint/connection is in some error state.*
- virtual UInt16 getLocalPort ()=0

  *Gets the port of the local session endpoint/socket.*
- virtual SPIInetAddress getLocalInetAddress ()=0

  *gets the inet address of the local session endpoint/socket.*
- virtual Int32 getPendingErrors ()

  *Reads the pending error related to the underlying socket and TLS library. One may call several times until all errors are read.*
- virtual const AlpnMode & getUsedAlpnMode () const =0

  *Gets the used AlpnMode.*
- virtual IANAProtocol getUsedProtocol () const =0

  *Gets the used INANAProtocol.*

## Protected Member Functions

- virtual void addPendingError (Int32 err)

  *Adds a pending error to the queue.*

## Protected Attributes

- std::queue< Int32 > m_errors

## 8.11.1 Detailed Description

This is an interface which defines a set of operation and features have to be available on each socket and session endpoint.

Definition at line 1015 of file TLSApiTypes.h.

## 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 ITLSSocketBase()

```
vwg::tls::ITLSSocketBase::ITLSSocketBase ( )  [default]
```

#### 8.11.2.2 ∼ITLSSocketBase()

```
virtual vwg::tls::ITLSSocketBase::∼ITLSSocketBase ( )  [virtual], [default]
```

### 8.11.3 Member Function Documentation

#### 8.11.3.1 addPendingError()

```
virtual void vwg::tls::ITLSSocketBase::addPendingError (
            Int32 err )  [inline], [protected], [virtual]
```

Adds a pending error to the queue.

**Since**

1.1.0

Definition at line 1132 of file TLSApiTypes.h.

#### 8.11.3.2 close()

```
virtual void vwg::tls::ITLSSocketBase::close ( )  [pure virtual]
```

Closes the underlying socket connection. This will immediately close the connection, all pending data may be lost, therefore one user shall call flush before closing.

**Note**

TLS lib will close only file descriptors that are created by the library and is not responsible for closing file descriptors created by the user. externally created file descriptors should be closed by the user.

#### 8.11.3.3 getLocalInetAddress()

```
virtual SPIInetAddress vwg::tls::ITLSSocketBase::getLocalInetAddress ( )  [pure virtual]
```

gets the inet address of the local session endpoint/socket.

**Returns**

gets the inet address of the session endpoint/socket.

#### 8.11.3.4 getLocalPort()

```
virtual UInt16 vwg::tls::ITLSSocketBase::getLocalPort ( )  [pure virtual]
```

Gets the port of the local session endpoint/socket.

**Returns**

Gets the port of the session endpoint/socket.

#### 8.11.3.5 getPendingErrors()

```
virtual Int32 vwg::tls::ITLSSocketBase::getPendingErrors ( )  [inline], [virtual]
```

Reads the pending error related to the underlying socket and TLS library. One may call several times until all errors are read.

**Returns**

The pending error code (see TLSReturnCodes) or a negative value if there are no pending errors anymore.

Definition at line 1096 of file TLSApiTypes.h.

#### 8.11.3.6 getUsedAlpnMode()

```
virtual const AlpnMode& vwg::tls::ITLSSocketBase::getUsedAlpnMode ( ) const  [pure virtual]
```

Gets the used AlpnMode.

**Returns**

The provided ALPN mode, if no AlpnMode is specified then the const AlpnMode::ALPN_OFF is returned.

**Since**

1.1.0

#### 8.11.3.7 getUsedProtocol()

```
virtual IANAProtocol vwg::tls::ITLSSocketBase::getUsedProtocol ( ) const  [pure virtual]
```

Gets the used INANAProtocol.

**Returns**

The used IANA protocol, In case ALPN is unused then the const IANAProtocol::NONE is returned.

**Since**

1.1.0

#### 8.11.3.8 isClosed()

```
virtual Boolean vwg::tls::ITLSSocketBase::isClosed ( )  [pure virtual]
```

Checks if the endpoint/connection is closed or not.

**Returns**

true if endpoint/connection is closed.

#### 8.11.3.9 isConnectionSocket()

```
virtual Boolean vwg::tls::ITLSSocketBase::isConnectionSocket ( )  [pure virtual]
```

Gets a boolean that tells if the socket is a stream socket.

**Returns**

true if the socket is a stream socket, otherwise false.

#### 8.11.3.10 isDatagramSocket()

```
Boolean vwg::tls::ITLSSocketBase::isDatagramSocket ( )  [inline]
```

Gets a boolean that tells if the socket is a Datagram socket.

**Returns**

true if the socket is a Datagram socket, otherwise false.

Definition at line 1028 of file TLSApiTypes.h.

#### 8.11.3.11 isErrorState()

```
virtual Boolean vwg::tls::ITLSSocketBase::isErrorState ( )  [inline], [virtual]
```

Checks if the endpoint/connection is in some error state.

**Returns**

true if endpoint/connection is in error state. One use getPendingErrors to read the errors. Depending on the error state the connection is closed already.

Definition at line 1071 of file TLSApiTypes.h.

#### 8.11.3.12 isOpen()

```
virtual Boolean vwg::tls::ITLSSocketBase::isOpen ( )  [pure virtual]
```

Checks if the endpoint/connection is closed or not.

**Returns**

true if endpoint/connection is closed.

### 8.11.4 Member Data Documentation

#### 8.11.4.1 m_errors

```
std::queue<Int32> vwg::tls::ITLSSocketBase::m_errors  [protected]
```

Definition at line 1137 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.12 vwg::tls::ITLSSocketFactory Class Reference

This is the interface of the socket factory. One need to get an instance of this interface to create a server or a client socket. Use the function initTLSLib to get the instance of the factory. The implementation will have only one instance of the factory.

```
#include <TLSSocketFactory.h>
```

## Public Member Functions

- ITLSSocketFactory ()=default
- virtual ∼ITLSSocketFactory ()=default
- virtual ApiVersionType getApiVersion ()=0

  *Gets the api version which is implemented.*
- virtual TLSServerSocketResult createServerSocket (SPIInetAddress inet, const UInt16 port, const std::string localDomainName, const SecurityLevel securityLevel, const SocketType socketType=SOCKETTYPE_STREAM)=0

  *Factory for creation of TLS secured server socket.*
- virtual TLSSessionEndpointResult createPskServerSession (int connectionFd, const std::string local↩ DomainName, const SecurityLevel confidentiality)=0

  *Factory for creation of TLS secured server socket.*
- virtual TLSServerSocketResult createServerSocket (int fd, const std::string localDomainName, const SecurityLevel confidentiality)=0

  *Factory for creation of TLS secured server socket.*
- virtual TLSClientSocketResult createClientSocket (SPIInetAddress inet, const UInt16 port, const std::string localDomainName, const SecurityLevel confidentiality, const SocketType socketType=SOCKETTYPE_STREAM)=0

  *Factory for creation of TLS secured client socket.*
- virtual TLSClientSocketResult createClientSocket (int fd, const std::string localDomainName, const SecurityLevel confidentiality)=0

  *Factory for creation of TLS secured client socket.*
- virtual TLSClientSocketResult createTlsClient (const std::shared_ptr< IOStream > stream, const std↩ ::string &hostName, const CertStoreID &certStoreId, const ClientCertificateSetID &clientCertificateSetID, const CipherSuiteIds &cipherSuiteIds, const TimeCheckTime &checkTime, const std::vector< HashSha256 > &httpPublicKeyPinningHashs, const bool revocationCheckEnabled=false)=0

  *factory for creation of TLS secured client end point on top of a given socket using certificates, using a stream instead of a socket.*
- virtual TLSClientSocketResult createTlsClient (const TLSConnectionSettings &connectionSettings, const std::shared_ptr< IOStream > stream, const std::string &hostName, const CertStoreID &certStoreId, const ClientCertificateSetID &clientCertificateSetID, const TimeCheckTime &checkTime, const std::vector< HashSha256 > &httpPublicKeyPinningHashs, const bool revocationCheckEnabled=false) noexcept=0

  *Factory for creation of TLS secured client end point on top of a given socket using certificates, using a stream instead of a socket.*

## 8.12.1 Detailed Description

This is the interface of the socket factory. One need to get an instance of this interface to create a server or a client socket. Use the function initTLSLib to get the instance of the factory. The implementation will have only one instance of the factory.

Definition at line 61 of file TLSSocketFactory.h.

## 8.12.2 Constructor & Destructor Documentation

#### 8.12.2.1 ITLSSocketFactory()

```
vwg::tls::ITLSSocketFactory::ITLSSocketFactory ( )  [default]
```

#### 8.12.2.2 ∼ITLSSocketFactory()

```
virtual vwg::tls::ITLSSocketFactory::∼ITLSSocketFactory ( )  [virtual], [default]
```

### 8.12.3 Member Function Documentation

#### 8.12.3.1 createClientSocket() [1/2]

```
virtual TLSClientSocketResult vwg::tls::ITLSSocketFactory::createClientSocket (
            int fd,
            const std::string localDomainName,
            const SecurityLevel confidentiality )  [pure virtual]
```

Factory for creation of TLS secured client socket.

This factory method will create underlying server socket and will use an SSL library. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

The PSK Key Mapping must be also defined as an external dependency.

**Parameters**

| | | |
|---|---|---|
| in | *fd* | the fd of the socket. Must be connected before creating. responsibilty is solely by the user of the api. |
| in | *localDomainName* | the SSOA defined domain name. Depending on the domain name the PSK key have to be used. (see Secure service communication Secure service-oriented architecture (sSOA) Technische Entwicklung, Querschnittslastenheft: LAH.000.036). |
| in | *confidentiality* | the SSOA confidentiality (see Secure service communication LHA) This call will accept only the security levels AUTHENTIC_WITHPSK or CONFIDENTIAL_WITHPSK. |

**Note**

TLS lib will close only file descriptors that are created by the library and is not responsible for closing file descriptors created by the user. externally created file descriptors should be closed by the user.

**Returns**

TLSClientSocketResult with socket or an error code the expected error code: can be

- RC_TLS_SUCCESSFUL

- RC_TLS_INIT_FAILED
- RC_TLS_CONNECT_FAILED
- RC_TLS_IO_ERROR
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION

#### 8.12.3.2 createClientSocket() [2/2]

```
virtual TLSClientSocketResult vwg::tls::ITLSSocketFactory::createClientSocket (
            SPIInetAddress inet,
            const UInt16 port,
            const std::string localDomainName,
            const SecurityLevel confidentiality,
            const SocketType socketType = SOCKETTYPE_STREAM )  [pure virtual]
```

Factory for creation of TLS secured client socket.

This factory method will create underlying server socket and will use an SSL library. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

The PSK Key Mapping must be also defined as an external dependency.

**Parameters**

| | | |
|---|---|---|
| in | *inet* | the given Inet address for the server to connect. |
| in | *port* | the port number of the socket. |

**Parameters**

| | | |
|---|---|---|
| in | *localDomainName* | the SSOA defined domain name. Depending on the domain name the PSK key have to be used. (see Secure service communication Secure service-oriented architecture (sSOA) Technische Entwicklung, Querschnittslastenheft: LAH.000.036). |
| in | *confidentiality* | the SSOA confidentiality (see Secure service communication LHA) This call will accept only the security levels AUTHENTIC_WITHPSK or CONFIDENTIAL_WITHPSK. |

**Returns**

TLSClientSocketResult with socket or an error code, the expected error code can be:

- RC_TLS_SUCCESSFUL
- RC_TLS_INIT_FAILED
- RC_TLS_CONNECT_FAILED
- RC_TLS_IO_ERROR
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION
- RC_TLS_INVALID_IP

#### 8.12.3.3 createPskServerSession()

```
virtual TLSSessionEndpointResult vwg::tls::ITLSSocketFactory::createPskServerSession (
            int connectionFd,
            const std::string localDomainName,
            const SecurityLevel confidentiality )  [pure virtual]
```

Factory for creation of TLS secured server socket.

This factory method will create underlying server socket and will use a SSL library. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

The PSK Key Mapping must be also defined as an external dependency.

**Parameters**

| | | |
|---|---|---|
| in | *connectionFd* | the FD is an already open and accepted connection. |
| in | *localDomainName* | the SSOA defined domain name. Depending on the domain name the PSK key have to be used. (see Secure service communication Secure service-oriented architecture (sSOA) Technische Entwicklung, Querschnittslastenheft: LAH.000.036). |
| in | *confidentiality* | the SSOA confidentiality (see Secure service communication LHA). This call will accept only the security levels AUTHENTIC_WITHPSK, CONFIDENTIAL_WITHPSK. |

**Note**

TLS lib will close only file descriptors that are created by the library and is not responsible for closing file descriptors created by the user. externally created file descriptors should be closed by the user.

**Returns**

TLSSessionEndpointResult with socket after handshake or error code the expected error code can be:

- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_IO_ERROR
- RC_TLS_SUCCESSFUL
- RC_TLS_INIT_FAILED
- RC_TLS_PROGRAMMING_ERROR_RESULT
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_PEER_CLOSED
- RC_TLS_BAD_RECORD_MAC

- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION

#### 8.12.3.4 createServerSocket() [1/2]

```
virtual TLSServerSocketResult vwg::tls::ITLSSocketFactory::createServerSocket (
            int fd,
            const std::string localDomainName,
            const SecurityLevel confidentiality )  [pure virtual]
```

Factory for creation of TLS secured server socket.

This factory method will create underlying server socket and will use a SSL library. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

The PSK Key Mapping must be also defined as an external dependency

**Parameters**

| | | |
|---|---|---|
| in | *fd* | the fd of the socket. Responsibility is solely by the user of the api, the lib assumes the fd is already initiated. |
| in | *localDomainName* | the SSOA defined domain name. Depending on the domain name the PSK key have to be used. (see Secure service communication Secure service-oriented architecture (sSOA) Technische Entwicklung, Querschnittslastenheft: LAH.000.036). |
| in | *confidentiality* | the SSOA confidentiality (see Secure service communication LHA). This call will accept only the security levels AUTHENTIC_WITHPSK, CONFIDENTIAL_WITHPSK. |

**Note**

TLS lib will close only file descriptors that are created by the library and is not responsible for closing file descriptors created by the user. externally created file descriptors should be closed by the user.

**Returns**

TLSServerSocketResult with socket or error code the expected error code can be :

- RC_TLS_SUCCESSFUL
- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_INIT_FAILED
- RC_TLS_IO_ERROR
- RC_TLS_PROGRAMMING_ERROR_RESULT
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_PEER_CLOSED
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION

#### 8.12.3.5 createServerSocket() [2/2]

```
virtual TLSServerSocketResult vwg::tls::ITLSSocketFactory::createServerSocket (
            SPIInetAddress inet,
            const UInt16 port,
            const std::string localDomainName,
            const SecurityLevel securityLevel,
            const SocketType socketType = SOCKETTYPE_STREAM )  [pure virtual]
```

Factory for creation of TLS secured server socket.

This factory method will create underlying server socket and will use a SSL library. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

The PSK Key Mapping must be also defined as an external dependency.

**Parameters**

| | | |
|---|---|---|
| in | *inet* | the given Inet address for the socket, where the server network socket is opened. see `http://man7.org/linux/man-pages/man2/socket.2.html` keep in mind the a system can have more than one inet address, therefore one need to provide the IP address. |
| in | *port* | the port number of the socket. |
| in | *localDomainName* | the SSOA defined domain name. Depending on the domain name the PSK key have to be used. (see Secure service communication Secure service-oriented architecture (sSOA) Technische Entwicklung, Querschnittslastenheft: LAH.000.036). |
| in | *securityLevel* | the SSOA confidentiality (see Secure service communication LHA). This call will accept only the security levels AUTHENTIC_WITHPSK, CONFIDENTIAL_WITHPSK. |
| in | *socketType* | defines the socket to be stream socket (TCP). |

**Returns**

TLSServerSocketResult with socket or error code, the expected error code can be:

- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_IO_ERROR
- RC_TLS_SUCCESSFUL
- RC_TLS_INIT_FAILED
- RC_TLS_PROGRAMMING_ERROR_RESULT
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_PEER_CLOSED
- RC_TLS_INVALID_IP
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION

#### 8.12.3.6 createTlsClient() [1/2]

```
virtual TLSClientSocketResult vwg::tls::ITLSSocketFactory::createTlsClient (
            const std::shared_ptr< IOStream > stream,
            const std::string & hostName,
            const CertStoreID & certStoreId,
            const ClientCertificateSetID & clientCertificateSetID,
            const CipherSuiteIds & cipherSuiteIds,
            const TimeCheckTime & checkTime,
            const std::vector< HashSha256 > & httpPublicKeyPinningHashs,
            const bool revocationCheckEnabled = false )  [pure virtual]
```

factory for creation of TLS secured client end point on top of a given socket using certificates, using a stream instead of a socket.

This connection will use the common TLS certificate based handshake according to the RFC 5246 for mutual authorization ( `https://www.ietf.org/rfc/rfc5246.txt` ). this factory method will a session endpoint on top of a given OS client socket (see `http://pubs.opengroup.↩ org/onlinepubs/7908799/xns/socket.html`). It assumes the the socket is already bounded and accepted, by the user of the method. In general it is within the method user responsibility to manage the socket. Especially it is important the the method user will not manipulate the socket in parallel nor call the creatTlsClient↩ Endpoint multiple times on the same socket. Any derivation may cause unexpected behavior. The method will try to make the TLS handshake on the given connection, which may fail to the undefined state of the socket connection. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are: a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

Security aspects.

1. The TLS connect will be always use "Service Name Indication". The "Service Name Indication" will be implemented according to **RFC 6066** (see `https://tools.ietf.org/html/rfc6066`). The "Service Name Indication" check will using the given domain name, which have to to be compliant to **RFC 5890**.
2. Certficates....

**Parameters**

| | | |
|---|---|---|
| in | *stream* | this is stream implementation playing the role of the socket where the encrypted data are written to or read from. The stream must be connected before the creating. If a multi-threaded system is used, make sure that the stream implementation includes a timeout value in the send and receive operations, without compromising the server's ability to listen and accept overtime. |
| in | *hostName* | : a) use the name to ensure the backend server will be authentic (server ID verification) b) this must be valid host(domain) name for performing "Service Name Indication" (SNI) (see also ps://de.wikipedia.org/wiki/Server_Name_Indication) the domainName must not be empty, it is mandatory to perform the "Service Name Indication" and "server ID verification" in any case. |
| in | *certStoreId* | the ID of the certificate store. This certificate store shall be located in the trust zone and contain all relevant certificates. predefined "VMKS": for VKMS Root Certificate(s), other for Trust Stores as contained in VI Trust Store Container |

**Parameters**

| | | |
|---|---|---|
| in | *clientCertificateSetID* | this defines the usage of the client key. This will define the if the key is used, if yes the location where the key is located and the key ID within the store. |
| in | *cipherSuiteIds* | A vector containing the list of supported cipher suites (ciphers defined in TLS- QLAH). If vector is empty (or contain only invalid options), default cipher pre defined use case will be used (TLSCipherSuiteUseCasesSettings::CSUSDefault use case). |
| in | *checkTime* | do the time check in addition to the certificate validity check. This check will verify if the certificate check time. This check can be omitted, by using null for this parameter. |
| in | *httpPublicKeyPinningHashs* | this is optional to support the HTTP Public Key pinning according to RFC 7469 (see `https://tools.ietf.org/html/rfc7469` for the RFC and `https://en.wikipedia.org/wiki/HTTP_↩ Public_Key_Pinning` for more details). basically this means at least one pin value must match any certificate in the full certificate chain. |
| in | *revocationCheckEnabled* | this is optional if set OCSP will be used. |

**Returns**

TLSClientSocketResult with socket or error code the expected error code can be:

- RC_TLS_SUCCESSFUL
- RC_TLS_INIT_FAILED
- RC_TLS_CONNECT_FAILED
- RC_TLS_IO_ERROR
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE
- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION

- RC_TLS_PEER_CLOSED
- RC_TLS_SEND_AFTER_SHUTDOWN
- RC_TLS_PUBLIC_KEY_PINNING_FAILED
- RC_TLS_BAD_CERTIFICATE
- RC_TLS_UNSUPPORTED_CERTIFICATE
- RC_TLS_CERTIFICATE_REVOKED
- RC_TLS_CERTIFICATE_EXPIRE
- RC_TLS_CERTIFICATE_UNKNOWN
- RC_TLS_UNKNOWN_CA

**Deprecated** this method becomes deprecated since 1.1.0, please use method with ALPN support.

#### 8.12.3.7 createTlsClient() [2/2]

```
virtual TLSClientSocketResult vwg::tls::ITLSSocketFactory::createTlsClient (
            const TLSConnectionSettings & connectionSettings,
            const std::shared_ptr< IOStream > stream,
            const std::string & hostName,
            const CertStoreID & certStoreId,
            const ClientCertificateSetID & clientCertificateSetID,
            const TimeCheckTime & checkTime,
            const std::vector< HashSha256 > & httpPublicKeyPinningHashs,
            const bool revocationCheckEnabled = false )  [pure virtual], [noexcept]
```

Factory for creation of TLS secured client end point on top of a given socket using certificates, using a stream instead of a socket.

This connection will use the common TLS certificate based handshake according to the RFC 5246 for mutual authorization ( `https://www.ietf.org/rfc/rfc5246.txt`). this factory method will a session endpoint on top of a given OS client socket (see `http://pubs.opengroup.↩ org/onlinepubs/7908799/xns/socket.html`). It assumes the socket is already bounded and accepted, by the user of the method. In general it is within the method user responsibility to manage the socket. Especially it is important the method user will not manipulate the socket in parallel nor call the creatTlsClient↩ Endpoint multiple times on the same socket. Any derivation may cause unexpected behavior. The method will try to make the TLS handshake on the given connection, which may fail to the undefined state of the socket connection. In contrast to the EB/Conti solution the network socket is created by the TLSSocket and the confidentiality is a mandatory parameter. The reasons for this are a) to be independent, form the manifest management. So this api can also be used for certificate based TLS connections, which will not have an security manifest (see CE Device Support). b) in case of port multiplexing in conjunction with IP routing this can be difficult to calculate the confidentiality correct. Therefore it may be a useful method to have the method getConfidentality4Port as a separated function.

Security aspects.

1. The TLS connect will be always use "Service Name Indication". The "Service Name Indication" will be implemented according to **RFC 6066** (see `https://tools.ietf.org/html/rfc6066`) The "Service Name Indication" check will using the given domain name, which have to to be compliant to **RFC 5890**.
2. Certificates....

**Parameters**

| | | |
|---|---|---|
| in | *connectionSettings* | This basic setting is used to define the ALPN mode and the set of cipher suite used. There is a set of predefined setting which can be used. |
| in | *stream* | this is stream implementation playing the role of the socket where the encrypted data are written to or read from. The stream must be connected before the creating. If a multi-threaded system is used, make sure that the stream implementation includes a timeout value in the send and receive operations, without compromising the server's ability to listen and accept overtime. |
| in | *hostName* | a) use the name to ensure the backend server will be authentic (server ID verification). b) this must be valid host(domain) name for performing "Service Name Indication" (SNI) (see also ps://de.wikipedia.org/wiki/Server_Name_Indication) domainName must not be empty, it is mandatory to perform the "Service Name Indication" and "server ID verification" in any case. |
| in | *certStoreId* | the ID of the certificate store. This certificate store shall be located in the trust zone and contain all relevant certificates. predefined "VMKS": for VKMS Root Certificate(s), other for Trust Stores as contained in VI Trust Store Container. |
| in | *clientCertificateSetID* | this defines the usage of the client key. This will define the if the key is used, if yes the location where the key is located and the key ID within the store. |
| in | *checkTime* | do the time check in addition to the certificate validity check. This check will verify if the certificate check time. This check can be omitted, by using null for this parameter. |
| in | *httpPublicKeyPinningHashs* | this is an optional to support the HTTP Public Key pinning according to RFC 7469 (see `https://tools.ietf.org/html/rfc7469` for the RFC and `https://en.wikipedia.org/wiki/HTTP↩_Public_Key_Pinning` for more details). basically this means at least one pin value must match any certificate in the full certificate chain. |
| in | *revocationCheckEnabled* | this is optional if set OCSP will be used. |

**Returns**

TLSClientSocketResult with socket or an error code, the expected error code can be:

- RC_TLS_SUCCESSFUL
- RC_TLS_INIT_FAILED
- RC_TLS_CONNECT_FAILED
- RC_TLS_IO_ERROR
- RC_TLS_WOULD_BLOCK_READ
- RC_TLS_WOULD_BLOCK_WRITE
- RC_TLS_UNEXPECTED_MESSAGE
- RC_TLS_BAD_RECORD_MAC
- RC_TLS_RECORD_OVERFLOW
- RC_TLS_DECOMPRESSION_FAILURE

- RC_TLS_HANDSHAKE_FAILURE
- RC_TLS_ILLEGAL_PARAMETER
- RC_TLS_ACCESS_DENIED
- RC_TLS_DECODE_ERROR
- RC_TLS_DECRYPT_ERROR
- RC_TLS_PROTOCOL_VERSION
- RC_TLS_INSUFFICIENT_SECURITY
- RC_TLS_NO_RENEGOTIATION
- RC_TLS_UNSUPPORTED_EXTENSION
- RC_TLS_PEER_CLOSED
- RC_TLS_SEND_AFTER_SHUTDOWN
- RC_TLS_PUBLIC_KEY_PINNING_FAILED
- RC_TLS_BAD_CERTIFICATE
- RC_TLS_UNSUPPORTED_CERTIFICATE
- RC_TLS_CERTIFICATE_REVOKED
- RC_TLS_CERTIFICATE_EXPIRE
- RC_TLS_CERTIFICATE_UNKNOWN
- RC_TLS_NO_APPLICATION_PROTOCOL
- RC_TLS_UNKNOWN_CA

**Since**

1.1.0

#### 8.12.3.8 getApiVersion()

```
virtual ApiVersionType vwg::tls::ITLSSocketFactory::getApiVersion ( )  [pure virtual]
```

Gets the api version which is implemented.

**Returns**

the API Version.

**Since**

1.1.0

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSSocketFactory.h

## 8.13 vwg::tls::TimeCheckTime Struct Reference

This is a structure that will be used to pass the authentic time. basically this time will be compared with the system time, as shown below.

```
#include <TLSApiTypes.h>
```

### Public Attributes

- std::time_t expectedTime

  *This is expected time to be compared with the system time. please keep in mind that the expected time can be either the authentic time provided by the authentic time service oder the UTC provided by the time service. The time service must be used because the system time is currently not defined and only the ICAS1 will have a RTC.*
- int permittedDeviation

  *A permitted deviation shall be given in seconds.*

### 8.13.1 Detailed Description

This is a structure that will be used to pass the authentic time. basically this time will be compared with the system time, as shown below.

|expectedTime - system_time.now() | <= |permittedDeviation|

If the difference of the |expectedTime - system_time.now() | is in the range of the |permittedDeviation| then the handshake will regarded as legal. The permittedDeviation shall be less than one day (86400sec), if the permitted↩ Deviation is above this it will be used MAX_PERMITTED_DEVIATION if the expectedTime is 0, then time check is not required.

Definition at line 124 of file TLSApiTypes.h.

### 8.13.2 Member Data Documentation

#### 8.13.2.1 expectedTime

```
std::time_t vwg::tls::TimeCheckTime::expectedTime
```

This is expected time to be compared with the system time. please keep in mind that the expected time can be either the authentic time provided by the authentic time service oder the UTC provided by the time service. The time service must be used because the system time is currently not defined and only the ICAS1 will have a RTC.

Definition at line 131 of file TLSApiTypes.h.

#### 8.13.2.2 permittedDeviation

```
int vwg::tls::TimeCheckTime::permittedDeviation
```

A permitted deviation shall be given in seconds.

Definition at line 136 of file TLSApiTypes.h.

The documentation for this struct was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.14 vwg::tls::TLSConnectionSettings Class Reference

this class is used to define the TLS connection properties for a backend TLS connection. This class contains a set of configuration properties for the TLS connection.

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- TLSConnectionSettings (const AlpnMode &alpnMode, TLSCipherSuiteUseCasesSettings cipherSuite↩
Settings=TLSCipherSuiteUseCasesSettings::CSUSDefault, const std::string &connectionLoggingName="")

    *Constructor.*
- TLSConnectionSettings (const AlpnMode &alpnMode, std::shared_ptr< ITLSOcspHandler > ocspHandler, const UInt32 ocspTimeoutMs=DEFAULT_OCSP_ONLINE_TIMEOUT_MS, TLSCipherSuiteUseCasesSettings cipherSuiteSettings=TLSCipherSuiteUseCasesSettings::CSUSDefault, const std::string &connection↩
LoggingName="")

    *Constructor.*
- TLSConnectionSettings (const AlpnMode &alpnMode, const std::string &cipherSuiteSettings, const std::string &connectionLoggingName="")

    *Creates a TLSConnectionSettings data config object to parametrize the TLS session.*
- ∼TLSConnectionSettings ()=default
- const TLSCipherSuiteUseCasesSettings & getCipherSuiteUseCasesSettings () const

    *Gets the cipher suite use case settings.*
- const AlpnMode & getAlpnMode () const

    *Gets the ALPN mode.*
- const std::shared_ptr< ITLSOcspHandler > & getOcspHandler () const

    *Gets the OCSP handler.*
- const UInt32 & getOcspTimeoutMs () const

    *Gets the OCSP timeout in milliseconds.*
- std::string getConnectionLoggingName () const

    *get the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging*

### Private Attributes

- const AlpnMode m_alpnMode
- std::shared_ptr< ITLSOcspHandler > m_ocspHandler
- const UInt32 m_ocspTimeoutMs
- TLSCipherSuiteUseCasesSettings m_cipherSuiteSettings
- std::string m_connectionLoggingName

### 8.14.1 Detailed Description

this class is used to define the TLS connection properties for a backend TLS connection. This class contains a set of configuration properties for the TLS connection.

**alpnMode**
The given ALPN Mode, set detail for ALPN mode at the according class

**cipherSuiteSettings**
Supported cipher suite set ( `https://devstack.vwgroup.com/jira/browse/IMAN-46128`) the parameter is given as a string, so it give maximal portability. If the given sting is not valid the default set is used.

**ocspHandler**

**ocspTimeoutMs**

**connectionLoggingName**
the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging

**Since**

1.2.0

**Since**

1.1.0

Definition at line 811 of file TLSApiTypes.h.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 TLSConnectionSettings() [1/3]

```
vwg::tls::TLSConnectionSettings::TLSConnectionSettings (
            const AlpnMode & alpnMode,
            TLSCipherSuiteUseCasesSettings cipherSuiteSettings = TLSCipherSuiteUseCasesSettings↩
::CSUSDefault,
            const std::string & connectionLoggingName = "" )  [inline]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *alpnMode* | The given ALPN Mode. |
| in | *cipherSuiteSettings* | Supported cipher suite set ( https://devstack.vwgroup.↩com/jira/browse/IMAN-46128). |
| in | *connectionLoggingName* | the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging |

Definition at line 825 of file TLSApiTypes.h.

### 8.14.2.2 TLSConnectionSettings() [2/3]

```
vwg::tls::TLSConnectionSettings::TLSConnectionSettings (
            const AlpnMode & alpnMode,
            std::shared_ptr< ITLSOcspHandler > ocspHandler,
            const UInt32 ocspTimeoutMs = DEFAULT_OCSP_ONLINE_TIMEOUT_MS,
            TLSCipherSuiteUseCasesSettings cipherSuiteSettings = TLSCipherSuiteUseCasesSettings↩
::CSUSDefault,
            const std::string & connectionLoggingName = "" )  [inline]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *alpnMode* | The given ALPN Mode. |
| in | *ocspHandler* | OCSP handler. |
| in | *ocspTimeoutMs* | OCSP timeout in milliseconds. |
| in | *cipherSuiteSettings* | Supported cipher suite set ( https://devstack.vwgroup.↩com/jira/browse/IMAN-46128). |
| in | *connectionLoggingName* | the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging |

Definition at line 849 of file TLSApiTypes.h.

### 8.14.2.3 TLSConnectionSettings() [3/3]

```
vwg::tls::TLSConnectionSettings::TLSConnectionSettings (
            const AlpnMode & alpnMode,
            const std::string & cipherSuiteSettings,
            const std::string & connectionLoggingName = "" )  [inline]
```

Creates a TLSConnectionSettings data config object to parametrize the TLS session.

**Parameters**

| | | |
|---|---|---|
| in | *alpnMode* | The given ALPN Mode. |
| in | *cipherSuiteSettings* | Supported cipher suite set ( `https://devstack.vwgroup.com/jira/browse/IMAN-46128`) the parameter is given as a string, so it give maximal portability. If the given string is invalid then the default set is used. |
| in | *connectionLoggingName* | the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging |

**Since**

1.1.0

Definition at line 877 of file TLSApiTypes.h.

References vwg::tls::CSUSDefaultWithSoftFail, vwg::tls::CSUSDefaulWithSoftFailtStr, vwg::tls::CSUSIanaRecommended, vwg::tls::CSUSIanaRecommendedStr, vwg::tls::CSUSLegacy, vwg::tls::CSUSLegacyStr, vwg::tls::CSUSLongtermSecure, and vwg::tls::CSUSLongtermSecureStr.

#### 8.14.2.4 ∼TLSConnectionSettings()

```
vwg::tls::TLSConnectionSettings::∼TLSConnectionSettings ( )  [default]
```

### 8.14.3 Member Function Documentation

#### 8.14.3.1 getAlpnMode()

```
const AlpnMode& vwg::tls::TLSConnectionSettings::getAlpnMode ( ) const  [inline]
```

Gets the ALPN mode.

**Returns**

The ALPN mode.

Definition at line 919 of file TLSApiTypes.h.

#### 8.14.3.2 getCipherSuiteUseCasesSettings()

```
const TLSCipherSuiteUseCasesSettings& vwg::tls::TLSConnectionSettings::getCipherSuiteUse↩
CasesSettings ( ) const  [inline]
```

Gets the cipher suite use case settings.

**Returns**

The cipher suite use case settings.

Definition at line 908 of file TLSApiTypes.h.

#### 8.14.3.3 getConnectionLoggingName()

```
std::string vwg::tls::TLSConnectionSettings::getConnectionLoggingName ( ) const  [inline]
```

get the ConnectionLoggingName This is a optional name to identify the connection for logging reasons. This name shall be provided by the user of the TLS library to identify the connection in logging

**Returns**

Tthe ConnectionLoggingName

**Since**

1.2.0

Definition at line 953 of file TLSApiTypes.h.

#### 8.14.3.4 getOcspHandler()

```
const std::shared_ptr<ITLSOcspHandler>& vwg::tls::TLSConnectionSettings::getOcspHandler ( )
const  [inline]
```

Gets the OCSP handler.

**Returns**

The OCSP handler.

Definition at line 930 of file TLSApiTypes.h.

#### 8.14.3.5 getOcspTimeoutMs()

```
const UInt32& vwg::tls::TLSConnectionSettings::getOcspTimeoutMs ( ) const  [inline]
```

Gets the OCSP timeout in milliseconds.

**Returns**

The OCSP handler.

Definition at line 941 of file TLSApiTypes.h.

### 8.14.4 Member Data Documentation

#### 8.14.4.1 m_alpnMode

```
const AlpnMode vwg::tls::TLSConnectionSettings::m_alpnMode  [private]
```

Definition at line 960 of file TLSApiTypes.h.

#### 8.14.4.2 m_cipherSuiteSettings

```
TLSCipherSuiteUseCasesSettings vwg::tls::TLSConnectionSettings::m_cipherSuiteSettings  [private]
```

Definition at line 963 of file TLSApiTypes.h.

#### 8.14.4.3 m_connectionLoggingName

```
std::string vwg::tls::TLSConnectionSettings::m_connectionLoggingName  [private]
```

Definition at line 964 of file TLSApiTypes.h.

#### 8.14.4.4 m_ocspHandler

```
std::shared_ptr<ITLSOcspHandler> vwg::tls::TLSConnectionSettings::m_ocspHandler  [private]
```

Definition at line 961 of file TLSApiTypes.h.

#### 8.14.4.5 m_ocspTimeoutMs

```
const UInt32 vwg::tls::TLSConnectionSettings::m_ocspTimeoutMs  [private]
```

Definition at line 962 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.15 vwg::tls::TLSOcspCachedResponse Class Reference

This class represents a cached OCSP response message.

```
#include <TLSApiTypes.h>
```

### Public Member Functions

- TLSOcspCachedResponse (const std::vector< UInt8 > &response, const UInt64 requestUniqueId, const std::string &producedAtDate, const std::string &nextUpdateDate, const std::string &thisUpdateDate)

  *Constructor.*
- TLSOcspCachedResponse (TLSOcspCachedResponse &&)=default
- TLSOcspCachedResponse (const TLSOcspCachedResponse &)=default
- TLSOcspCachedResponse & operator= (const TLSOcspCachedResponse &)=default
- TLSOcspCachedResponse & operator= (TLSOcspCachedResponse &&)=default
- ~TLSOcspCachedResponse ()=default
- const std::vector< UInt8 > & getResponse () const noexcept

  *Gets the OCSP response message.*
- UInt64 getRequestUniqueId () const noexcept

  *Gets the unique ID of the related OCSP request for this OCSP response.*
- const std::string & getProducedAt () const noexcept

  *Gets producedAt date parameter from the response.*
- const std::string & getNextUpdate () const noexcept

  *Gets nextUpdate date parameter from the response.*
- const std::string & getThisUpdate () const noexcept

  *Gets thisUpdate date parameter from the response.*

### Private Attributes

- std::vector< UInt8 > m_response
- UInt64 m_requestUniqueId
- std::string m_producedAt
- std::string m_nextUpdate
- std::string m_thisUpdate

### 8.15.1 Detailed Description

This class represents a cached OCSP response message.

Definition at line 626 of file TLSApiTypes.h.

### 8.15.2 Constructor & Destructor Documentation

#### 8.15.2.1 TLSOcspCachedResponse() [1/3]

```
vwg::tls::TLSOcspCachedResponse::TLSOcspCachedResponse (
            const std::vector< UInt8 > & response,
            const UInt64 requestUniqueId,
            const std::string & producedAtDate,
            const std::string & nextUpdateDate,
            const std::string & thisUpdateDate )  [inline]
```

Constructor.

**Note**

all dates are expressed according to ISO8601 in UTC - YYYYMMDDHHMMSSZ.

**Parameters**

| | | |
|---|---|---|
| in | *response* | Vector of bytes that contains raw OCSP response message encoded in BER format. |
| in | *request↩ UniqueId* | Unique ID of the related OCSP request for this OCSP response. |
| in | *producedAtDate* | The time at which the OCSP responder signed this OCSP response. |
| in | *nextUpdateDate* | The time at or before which newer information will be available about the status of the certificate. |
| in | *thisUpdateDate* | The most recent time at which the status being indicated is known by the OCSP responder to have been correct. |

Definition at line 642 of file TLSApiTypes.h.

#### 8.15.2.2 TLSOcspCachedResponse() [2/3]

```
vwg::tls::TLSOcspCachedResponse::TLSOcspCachedResponse (
            TLSOcspCachedResponse &&  )  [default]
```

#### 8.15.2.3 TLSOcspCachedResponse() [3/3]

```
vwg::tls::TLSOcspCachedResponse::TLSOcspCachedResponse (
            const TLSOcspCachedResponse &  )  [default]
```

#### 8.15.2.4 ∼TLSOcspCachedResponse()

```
vwg::tls::TLSOcspCachedResponse::∼TLSOcspCachedResponse ( )  [default]
```

### 8.15.3 Member Function Documentation

#### 8.15.3.1 getNextUpdate()

```
const std::string& vwg::tls::TLSOcspCachedResponse::getNextUpdate ( ) const  [inline], [noexcept]
```

Gets nextUpdate date parameter from the response.

**Note**

Date is expressed according to ISO8601 in UTC - YYYYMMDDHHMMSSZ.

**Returns**

String which contains the date in ISO8601 format.

Definition at line 706 of file TLSApiTypes.h.

#### 8.15.3.2 getProducedAt()

```
const std::string& vwg::tls::TLSOcspCachedResponse::getProducedAt ( ) const  [inline], [noexcept]
```

Gets producedAt date parameter from the response.

**Note**

Date is expressed according to ISO8601 in UTC - YYYYMMDDHHMMSSZ.

**Returns**

String which contains the date in ISO8601 format.

Definition at line 693 of file TLSApiTypes.h.

#### 8.15.3.3 getRequestUniqueId()

```
UInt64 vwg::tls::TLSOcspCachedResponse::getRequestUniqueId ( ) const  [inline], [noexcept]
```

Gets the unique ID of the related OCSP request for this OCSP response.

**Returns**

OCSP request message unique ID.

Definition at line 680 of file TLSApiTypes.h.

#### 8.15.3.4 getResponse()

```
const std::vector<UInt8>& vwg::tls::TLSOcspCachedResponse::getResponse ( ) const  [inline],
[noexcept]
```

Gets the OCSP response message.

**Returns**

Vector of bytes that contains the response in BER encoding.

Definition at line 669 of file TLSApiTypes.h.

#### 8.15.3.5 getThisUpdate()

```
const std::string& vwg::tls::TLSOcspCachedResponse::getThisUpdate ( ) const  [inline], [noexcept]
```

Gets thisUpdate date parameter from the response.

**Note**

Date is expressed according to ISO8601 in UTC - YYYYMMDDHHMMSSZ.

**Returns**

String which contains the date in ISO8601 format.

Definition at line 719 of file TLSApiTypes.h.

#### 8.15.3.6 operator=() [1/2]

```
TLSOcspCachedResponse& vwg::tls::TLSOcspCachedResponse::operator= (
             const TLSOcspCachedResponse &  )  [default]
```

#### 8.15.3.7 operator=() [2/2]

```
TLSOcspCachedResponse& vwg::tls::TLSOcspCachedResponse::operator= (
             TLSOcspCachedResponse &&  )  [default]
```

### 8.15.4 Member Data Documentation

#### 8.15.4.1 m_nextUpdate

```
std::string vwg::tls::TLSOcspCachedResponse::m_nextUpdate  [private]
```

Definition at line 728 of file TLSApiTypes.h.

#### 8.15.4.2 m_producedAt

```
std::string vwg::tls::TLSOcspCachedResponse::m_producedAt  [private]
```

Definition at line 727 of file TLSApiTypes.h.

#### 8.15.4.3 m_requestUniqueId

```
UInt64 vwg::tls::TLSOcspCachedResponse::m_requestUniqueId  [private]
```

Definition at line 726 of file TLSApiTypes.h.

#### 8.15.4.4 m_response

```
std::vector<UInt8> vwg::tls::TLSOcspCachedResponse::m_response  [private]
```

Definition at line 725 of file TLSApiTypes.h.

#### 8.15.4.5 m_thisUpdate

`std::string vwg::tls::TLSOcspCachedResponse::m_thisUpdate [private]`

Definition at line 729 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.16 vwg::tls::TLSOcspRequest Class Reference

This class represents a wrapper for a raw OCSP request message.

`#include <TLSApiTypes.h>`

### Public Member Functions

- TLSOcspRequest (const std::string &url, const std::vector< UInt8 > &request)
  *Constructor.*
- TLSOcspRequest (const std::string &url, const std::vector< UInt8 > &request, const UInt64 uniqueId)
  *Constructor.*
- TLSOcspRequest (TLSOcspRequest &&)=default
- TLSOcspRequest (const TLSOcspRequest &)=default
- TLSOcspRequest & operator= (const TLSOcspRequest &)=default
- TLSOcspRequest & operator= (TLSOcspRequest &&)=default
- ∼TLSOcspRequest ()=default
- UInt64 getUniqueId () const noexcept
  *Gets unique ID that identifies the request.*
- const std::vector< UInt8 > & getRequest () const noexcept
  *Gets the OCSP request message.*
- const std::string & getRequestUrl () const noexcept
  *Gets request's OCSP Responder URL.*

### Private Member Functions

- void calculateUniqueId ()
  *Calculates request's unique ID.*

### Private Attributes

- std::string m_responderUrl
- std::vector< UInt8 > m_request
- UInt64 m_uniqueId

## Static Private Attributes

- static constexpr UInt8 OCSP_REQUEST_WITHOUT_EXTENSIONS_SIZE = 73

  *Contains OCSP request size in bytes without "OCSP extensions" size).*

## 8.16.1 Detailed Description

This class represents a wrapper for a raw OCSP request message.

Definition at line 415 of file TLSApiTypes.h.

## 8.16.2 Constructor & Destructor Documentation

### 8.16.2.1 TLSOcspRequest() `[1/4]`

```
vwg::tls::TLSOcspRequest::TLSOcspRequest (
            const std::string & url,
            const std::vector< UInt8 > & request )  [inline]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *url* | String which contains the OCSP Responder's URL. |
| in | *request* | Vector of bytes which contains a single OCSP Request encoded in BER format. |

Definition at line 424 of file TLSApiTypes.h.

### 8.16.2.2 TLSOcspRequest() `[2/4]`

```
vwg::tls::TLSOcspRequest::TLSOcspRequest (
            const std::string & url,
            const std::vector< UInt8 > & request,
            const UInt64 uniqueId )  [inline]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *url* | String which contains the OCSP Responder's URL. |
| in | *request* | Vector of bytes which contains a single OCSP Request message encoded in BER format. |
| in | *unique↩ Id* | OCSP Request's unique hash ID. |

Definition at line 438 of file TLSApiTypes.h.

#### 8.16.2.3 TLSOcspRequest() [3/4]

```
vwg::tls::TLSOcspRequest::TLSOcspRequest (
            TLSOcspRequest &&  )  [default]
```

#### 8.16.2.4 TLSOcspRequest() [4/4]

```
vwg::tls::TLSOcspRequest::TLSOcspRequest (
            const TLSOcspRequest &  )  [default]
```

#### 8.16.2.5 ∼TLSOcspRequest()

```
vwg::tls::TLSOcspRequest::∼TLSOcspRequest ( )  [default]
```

### 8.16.3 Member Function Documentation

#### 8.16.3.1 calculateUniqueId()

```
void vwg::tls::TLSOcspRequest::calculateUniqueId ( )  [inline], [private]
```

Calculates request's unique ID.

this method calculates a unique ID by doing operations on the OCSP request (without "OCSP extensions") and the responder URL.

Definition at line 499 of file TLSApiTypes.h.

#### 8.16.3.2 getRequest()

```
const std::vector<UInt8>& vwg::tls::TLSOcspRequest::getRequest ( ) const  [inline], [noexcept]
```

Gets the OCSP request message.

**Returns**

Vector of bytes that contains the request in BER encoding.

Definition at line 475 of file TLSApiTypes.h.

#### 8.16.3.3 getRequestUrl()

```
const std::string& vwg::tls::TLSOcspRequest::getRequestUrl ( ) const  [inline], [noexcept]
```

Gets request's OCSP Responder URL.

**Returns**

string that tells the OCSP responder URL.

Definition at line 486 of file TLSApiTypes.h.

#### 8.16.3.4 getUniqueId()

```
UInt64 vwg::tls::TLSOcspRequest::getUniqueId ( ) const  [inline], [noexcept]
```

Gets unique ID that identifies the request.

This shall be uniquely identifiable the OCSP request so it can be cached. Assuming that the same OCSP request will lead to the same OCSP response (apart from the fact the server is down, cert is revoked or network is not available etc...), one can save and rerun the OCSP request and can use the cached OCSP response.

**Returns**

OCSP request message unique ID.

Definition at line 464 of file TLSApiTypes.h.

#### 8.16.3.5 operator=() [1/2]

```
TLSOcspRequest& vwg::tls::TLSOcspRequest::operator= (
             const TLSOcspRequest &  )  [default]
```

#### 8.16.3.6 operator=() [2/2]

```
TLSOcspRequest& vwg::tls::TLSOcspRequest::operator= (
             TLSOcspRequest &&  )  [default]
```

### 8.16.4 Member Data Documentation

#### 8.16.4.1 m_request

```
std::vector<UInt8> vwg::tls::TLSOcspRequest::m_request  [private]
```

Definition at line 517 of file TLSApiTypes.h.

#### 8.16.4.2 m_responderUrl

```
std::string vwg::tls::TLSOcspRequest::m_responderUrl  [private]
```

Definition at line 516 of file TLSApiTypes.h.

#### 8.16.4.3 m_uniqueId

```
UInt64 vwg::tls::TLSOcspRequest::m_uniqueId  [private]
```

Definition at line 518 of file TLSApiTypes.h.

#### 8.16.4.4 OCSP_REQUEST_WITHOUT_EXTENSIONS_SIZE

```
constexpr UInt8 vwg::tls::TLSOcspRequest::OCSP_REQUEST_WITHOUT_EXTENSIONS_SIZE = 73  [static],
[constexpr], [private]
```

Contains OCSP request size in bytes without "OCSP extensions" size).

Definition at line 523 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.17 vwg::tls::TLSOcspRequestResponse Class Reference

This class represents a wrapper for a raw OCSP response message which used as a result object from the OCSP. Proxy process after requests processing.

```
#include <TLSApiTypes.h>
```

## Public Member Functions

- TLSOcspRequestResponse (const std::vector< UInt8 > &response, const Boolean isCached, const UInt64 requestUniqueId)

  *Constructor.*
- TLSOcspRequestResponse (const UInt64 requestUniqueId)

  *Constructor.*
- TLSOcspRequestResponse (TLSOcspRequestResponse &&)=default
- TLSOcspRequestResponse (const TLSOcspRequestResponse &)=default
- TLSOcspRequestResponse & operator= (const TLSOcspRequestResponse &)=default
- TLSOcspRequestResponse & operator= (TLSOcspRequestResponse &&)=default
- ∼TLSOcspRequestResponse ()=default
- Boolean getIsCached () const noexcept

  *Gets an OCSP Response caching status.*
- const std::vector< UInt8 > & getResponse () const noexcept

  *Gets the OCSP response message.*
- UInt64 getRequestUniqueId () const noexcept

  *Gets the unique ID of the related OCSP request for this OCSP response.*
- Boolean isCorrupted () const noexcept

  *Gets a boolean that tells if the response corrupted.*

## Private Attributes

- Boolean m_isCached
- std::vector< UInt8 > m_response
- UInt64 m_requestUniqueId
- Boolean m_isCorrupted

## 8.17.1 Detailed Description

This class represents a wrapper for a raw OCSP response message which used as a result object from the OCSP. Proxy process after requests processing.

Definition at line 530 of file TLSApiTypes.h.

## 8.17.2 Constructor & Destructor Documentation

### 8.17.2.1 TLSOcspRequestResponse() [1/4]

```
vwg::tls::TLSOcspRequestResponse::TLSOcspRequestResponse (
            const std::vector< UInt8 > & response,
            const Boolean isCached,
            const UInt64 requestUniqueId )  [inline]
```

Constructor.

**Parameters**

| | | |
|---|---|---|
| in | *response* | Vector of bytes which contains a single OCSP response encoded message in BER format. |
| in | *isCached* | Indicates if the object cached. |
| in | *request↩ UniqueId* | The unique ID of the related OCSP request. |

Definition at line 540 of file TLSApiTypes.h.

### 8.17.2.2 TLSOcspRequestResponse() [2/4]

```
vwg::tls::TLSOcspRequestResponse::TLSOcspRequestResponse (
            const UInt64 requestUniqueId )  [inline]
```

Constructor.

**Note**

Use this constructor to build an OCSP request response object with is corrupted.

**Parameters**

| | | |
|---|---|---|
| in | *request↩ UniqueId* | The unique ID of the related OCSP request. |

Definition at line 555 of file TLSApiTypes.h.

### 8.17.2.3 TLSOcspRequestResponse() [3/4]

```
vwg::tls::TLSOcspRequestResponse::TLSOcspRequestResponse (
            TLSOcspRequestResponse &&  )  [default]
```

### 8.17.2.4 TLSOcspRequestResponse() [4/4]

```
vwg::tls::TLSOcspRequestResponse::TLSOcspRequestResponse (
            const TLSOcspRequestResponse &  )  [default]
```

### 8.17.2.5 ∼TLSOcspRequestResponse()

```
vwg::tls::TLSOcspRequestResponse::∼TLSOcspRequestResponse ( )  [default]
```

### 8.17.3 Member Function Documentation

#### 8.17.3.1 getIsCached()

`Boolean vwg::tls::TLSOcspRequestResponse::getIsCached ( ) const [inline], [noexcept]`

Gets an OCSP Response caching status.

**Returns**

A boolean flag that indicates if OCSP Response cached or not cached.

Definition at line 577 of file TLSApiTypes.h.

#### 8.17.3.2 getRequestUniqueId()

`UInt64 vwg::tls::TLSOcspRequestResponse::getRequestUniqueId ( ) const [inline], [noexcept]`

Gets the unique ID of the related OCSP request for this OCSP response.

**Returns**

OCSP request message unique ID.

Definition at line 599 of file TLSApiTypes.h.

#### 8.17.3.3 getResponse()

`const std::vector<UInt8>& vwg::tls::TLSOcspRequestResponse::getResponse ( ) const [inline], [noexcept]`

Gets the OCSP response message.

**Returns**

Vector of bytes that contains the response in BER encoding.

Definition at line 588 of file TLSApiTypes.h.

#### 8.17.3.4 isCorrupted()

```
Boolean vwg::tls::TLSOcspRequestResponse::isCorrupted ( ) const  [inline], [noexcept]
```

Gets a boolean that tells if the response corrupted.

**Returns**

Response corruption status.

Definition at line 610 of file TLSApiTypes.h.

#### 8.17.3.5 operator=() [1/2]

```
TLSOcspRequestResponse& vwg::tls::TLSOcspRequestResponse::operator= (
             const TLSOcspRequestResponse &  )  [default]
```

#### 8.17.3.6 operator=() [2/2]

```
TLSOcspRequestResponse& vwg::tls::TLSOcspRequestResponse::operator= (
             TLSOcspRequestResponse &&  )  [default]
```

### 8.17.4 Member Data Documentation

#### 8.17.4.1 m_isCached

```
Boolean vwg::tls::TLSOcspRequestResponse::m_isCached  [private]
```

Definition at line 617 of file TLSApiTypes.h.

#### 8.17.4.2 m_isCorrupted

```
Boolean vwg::tls::TLSOcspRequestResponse::m_isCorrupted  [private]
```

Definition at line 620 of file TLSApiTypes.h.

#### 8.17.4.3 m_requestUniqueId

```
UInt64 vwg::tls::TLSOcspRequestResponse::m_requestUniqueId  [private]
```

Definition at line 619 of file TLSApiTypes.h.

#### 8.17.4.4 m_response

```
std::vector<UInt8> vwg::tls::TLSOcspRequestResponse::m_response  [private]
```

Definition at line 618 of file TLSApiTypes.h.

The documentation for this class was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSApiTypes.h

## 8.18 vwg::tls::TLSResult< T > Struct Template Reference

This is a struct to return the return code or the value in case the operation is performed successful. Basically it will take a payload or an return code. One can assume that the paylod is empty if the operation failed. One have to use failed or succeeded first to check if the payload is set or not first. Currently it is assumed that the access of a empty payload will fail and an error is raised.

```
#include <TLSResult.h>
```

### Public Types

- using TT = TLSResult< T >

### Public Member Functions

- TLSResult ()
- TLSResult (TLSReturnCodes code)
- TLSResult (T payload)
- TT & operator= (const TT &other)
- bool failed ()

  *Checks if the operation failed.*
- bool succeeded ()

  *Checks if the operation failed.*
- T getPayload ()

  *Gets the payload.* ***Caution!</> this will raise an error if the payload is empty. please check the result with failed and succeeded before hand.***
- TLSReturnCodes getErrorCode ()

  *Gets the error code.*

## Private Attributes

- Boolean m_isEmpty
- TLSReturnCodes m_rc
- T m_payload

## 8.18.1 Detailed Description

**template<class T>**
**struct vwg::tls::TLSResult< T >**

This is a struct to return the return code or the value in case the operation is performed successful. Basically it will take a payload or an return code. One can assume that the paylod is empty if the operation failed. One have to use failed or succeeded first to check if the payload is set or not first. Currently it is assumed that the access of a empty payload will fail and an error is raised.

Definition at line 53 of file TLSResult.h.

## 8.18.2 Member Typedef Documentation

### 8.18.2.1 TT

```
template<class T >
using vwg::tls::TLSResult< T >::TT = TLSResult<T>
```

Definition at line 54 of file TLSResult.h.

## 8.18.3 Constructor & Destructor Documentation

### 8.18.3.1 TLSResult() `[1/3]`

```
template<class T >
vwg::tls::TLSResult< T >::TLSResult ( )  [inline]
```

Definition at line 62 of file TLSResult.h.

### 8.18.3.2 TLSResult() `[2/3]`

```
template<class T >
vwg::tls::TLSResult< T >::TLSResult (
             TLSReturnCodes code )  [inline]
```

Definition at line 66 of file TLSResult.h.

#### 8.18.3.3 TLSResult() [3/3]

```
template<class T >
vwg::tls::TLSResult< T >::TLSResult (
             T payload )  [inline]
```

Definition at line 70 of file TLSResult.h.

### 8.18.4 Member Function Documentation

#### 8.18.4.1 failed()

```
template<class T >
bool vwg::tls::TLSResult< T >::failed ( )  [inline]
```

Checks if the operation failed.

**Returns**

true if operation failed and the payload is empty.

Definition at line 100 of file TLSResult.h.

#### 8.18.4.2 getErrorCode()

```
template<class T >
TLSReturnCodes vwg::tls::TLSResult< T >::getErrorCode ( )  [inline]
```

Gets the error code.

**Returns**

the error code.

Definition at line 136 of file TLSResult.h.

#### 8.18.4.3 getPayload()

```
template<class T >
T vwg::tls::TLSResult< T >::getPayload ( )  [inline]
```

Gets the payload. **Caution!</> this will raise an error if the payload is empty. please check the result with failed and succeeded before hand.**

**Returns**

the payload.

Definition at line 124 of file TLSResult.h.

#### 8.18.4.4 operator=()

```
template<class T >
TT& vwg::tls::TLSResult< T >::operator= (
             const TT & other )  [inline]
```

Definition at line 79 of file TLSResult.h.

References vwg::tls::TLSResult< T >::m_isEmpty, vwg::tls::TLSResult< T >::m_payload, and vwg::tls::TLS↩
Result< T >::m_rc.

#### 8.18.4.5 succeeded()

```
template<class T >
bool vwg::tls::TLSResult< T >::succeeded ( )  [inline]
```

Checks if the operation failed.

**Returns**

true if operation failed and the payload is not empty.

Definition at line 111 of file TLSResult.h.

References vwg::tls::RC_TLS_SUCCESSFUL.

### 8.18.5 Member Data Documentation

#### 8.18.5.1 m_isEmpty

```
template<class T >
Boolean vwg::tls::TLSResult< T >::m_isEmpty  [private]
```

Definition at line 57 of file TLSResult.h.

Referenced by vwg::tls::TLSResult< T >::operator=().

#### 8.18.5.2 m_payload

```
template<class T >
T vwg::tls::TLSResult< T >::m_payload  [private]
```

Definition at line 59 of file TLSResult.h.

Referenced by vwg::tls::TLSResult< T >::operator=().

#### 8.18.5.3 m_rc

```
template<class T >
TLSReturnCodes vwg::tls::TLSResult< T >::m_rc  [private]
```

Definition at line 58 of file TLSResult.h.

Referenced by vwg::tls::TLSResult< T >::operator=().

The documentation for this struct was generated from the following file:

- /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TLSResult.h

# Chapter 9

# File Documentation

## 9.1 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAP↩ I/doxygen/mainTLSStreamAndSocketAPI.dox File Reference

## 9.2 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/↩ CipherSuitesDefenitions.h File Reference

```
#include "vwgtypes.h"
```

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Typedefs

- using vwg::tls::CipherSuiteIds = std::string

### Enumerations

- enum vwg::tls::CipherSuiteId : vwg::types::UInt16 {
  vwg::tls::TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = 0xCCA9, vwg::tls::TLS_ECDHE_ECDSA_WITH_A
  = 0xC02C, vwg::tls::TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 = 0xC02B, vwg::tls::TLS_ECDHE_RSA_WITH_AE
  = 0xC030,
  vwg::tls::TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 = 0xC02F, vwg::tls::TLS_DHE_RSA_WITH_AES_256_GCM_SH
  = 0x009F, vwg::tls::TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 = 0x009E, vwg::tls::TLS_ECDHE_ECDSA_WITH_AES_1
  = 0xC023,
  vwg::tls::TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = 0xCCA8, vwg::tls::TLS_DHE_RSA_WITH_CHACHA2

= 0xCCAA, vwg::tls::TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA = 0xC009, vwg::tls::TLS_ECDHE_ECDSA_WITH_AE
= 0xC00A,
vwg::tls::TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 = 0xC027, vwg::tls::TLS_ECDHE_RSA_WITH_AES_128_CBC_
= 0xC013, vwg::tls::TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA = 0xC014, vwg::tls::TLS_DHE_RSA_WITH_AES_128_CB
= 0x0067,
vwg::tls::TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 = 0x006B, vwg::tls::TLS_RSA_WITH_AES_128_GCM_SHA256
= 0x009C, vwg::tls::TLS_RSA_WITH_AES_256_GCM_SHA384 = 0x009D, vwg::tls::TLS_RSA_WITH_AES_128_CBC_SHA25
= 0x003C,
vwg::tls::TLS_RSA_WITH_AES_256_CBC_SHA256 = 0x003D, vwg::tls::TLS_RSA_WITH_AES_128_CBC_SHA
= 0x002F, vwg::tls::TLS_RSA_WITH_AES_256_CBC_SHA = 0x0035, vwg::tls::TLS_RSA_WITH_3DES_EDE_CBC_SHA
= 0x000A }

*This enum defines the list of permitted cipher suits.*

## 9.3 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/↩ InetAddress.h File Reference

```
#include <memory>
#include <sys/socket.h>
#include <netinet/in.h>
#include <cstddef>
#include <cstring>
#include <string>
#include "vwgtypes.h"
#include "TLSResult.h"
```

### Classes

- class vwg::tls::IInetAddress

  *Representation an interface of an IP address. Basically this will give you an immutable IP address interface.*
- class vwg::tls::InetAddressFactory

  *This a definition of a the factory to create instances of the IInetAddress. The supplier has to provide the implementation of the static methods by this class. Basically there is no need to create an instance of this class.*

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Typedefs

- using vwg::tls::SPIInetAddress = std::shared_ptr< IInetAddress >
- using vwg::tls::IInetAddressResult = TLSResult< SPIInetAddress >

## 9.4 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/IO↩ Stream.h File Reference

```
#include "vwgtypes.h"
```

### Classes

- class vwg::tls::IOStream

  *Representation an interface of an I/O stream. Can read, write and close.*

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Enumerations

- enum vwg::tls::StreamReturnCode { vwg::tls::RC_STREAM_WOULD_BLOCK = -1, vwg::tls::RC_STREAM_IO_ERROR = -2 }

  *Error values for receiving or sending data.*

## 9.5 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TL↩ SApiTypes.h File Reference

```
#include <ctime>
#include <functional>
#include <future>
#include <queue>
#include "InetAddress.h"
#include "vwgtypes.h"
```

## Classes

- class vwg::tls::IANAProtocolFunction

  *This class contains some helper methods when conversion from the IANAProtocol enum value to Protocol name.*
- struct vwg::tls::TimeCheckTime

  *This is a structure that will be used to pass the authentic time. basically this time will be compared with the system time, as shown below.*
- class vwg::tls::AlpnMode

  *A setting container for ALPN supporting. There are basically three modes possible:*
- class vwg::tls::TLSOcspRequest

  *This class represents a wrapper for a raw OCSP request message.*
- class vwg::tls::TLSOcspRequestResponse

  *This class represents a wrapper for a raw OCSP response message which used as a result object from the OCSP. Proxy process after requests processing.*
- class vwg::tls::TLSOcspCachedResponse

  *This class represents a cached OCSP response message.*
- class vwg::tls::ITLSOcspHandler

  *This interface defines APIs to process and handle OCSP messages.*
- class vwg::tls::TLSConnectionSettings

  *this class is used to define the TLS connection properties for a backend TLS connection. This class contains a set of configuration properties for the TLS connection.*
- class vwg::tls::ITLSErrorListener
- class vwg::tls::ITLSSocketBase

  *This is an interface which defines a set of operation and features have to be available on each socket and session endpoint.*

## Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

## Typedefs

- using vwg::tls::ApiVersionType = std::string
- typedef void(∗ vwg::tls::ErrorHandler) (SPIInetAddress inet, const UInt16 port, const TLSReturnCodes errorCode)

## Enumerations

- enum vwg::tls::IANAProtocol { vwg::tls::NONE = 0, vwg::tls::HTTP = 1, vwg::tls::HTTP2 = 2 }

  *This enum defines the supported protocols which can be used in case ALPN is used. Please see the IANAProtocol definitions in RFC7230* `https://tools.ietf.org/html/rfc7230`*.*
- enum vwg::tls::TLSCipherSuiteUseCasesSettings : UInt32 {
  vwg::tls::CSUSDefault = 0, vwg::tls::CSUSLegacy = 1, vwg::tls::CSUSLongtermSecure = 2, vwg::tls::CSUSIanaRecommended = 3,
  vwg::tls::CSUSDefaultWithSoftFail = 4, vwg::tls::CSUSEndOfEnum }
- enum vwg::tls::SecurityLevel : UInt32 { vwg::tls::AUTHENTIC_WITHPSK = 0, vwg::tls::CONFIDENTIAL_WITHPSK = 1 }

  *Defines the SSOA confidentiality.*
- enum vwg::tls::SocketType : UInt32 { vwg::tls::SOCKETTYPE_STREAM = 0, vwg::tls::SOCKETTYPE_DATAGRAM = 1 }

  *Defines the socket type.*
- enum vwg::tls::TLSDropSuppot : UInt32 { vwg::tls::TLS_NOT_DROPABLE = 0, vwg::tls::TLS_DROPABLE = 1 }

### Functions

- const ApiVersionType vwg::tls::ApiVersion ("TLS_API_1.3")

### Variables

- const static unsigned int vwg::tls::MAX_PERMITTED_DEVIATION = 86400

  *Defines the maximum permitted deviation of |expectedTime - system_time.now()|. since 1.1.0.*
- const static TimeCheckTime vwg::tls::CHECK_TIME_OFF = {0, 0}

  *Defines that time check is not required.*
- const static UInt32 vwg::tls::DEFAULT_OCSP_ONLINE_TIMEOUT_MS = 30000

  *Defines a default OCSP timeout in milliseconds.*
- const static AlpnMode vwg::tls::ALPN_OFF = AlpnMode(std::vector<IANAProtocol>{NONE})

  *Defines that ALPN is off and the protocol is undecided, this is identical to TLS without any ALPN support.*
- const static AlpnMode vwg::tls::ALPN_DEFAULT = AlpnMode(std::vector<IANAProtocol>{HTTP})

  *Defines the default ALPN.*
- const static AlpnMode vwg::tls::ALPN_HTTP2 = AlpnMode(std::vector<IANAProtocol>{IANAProtocol::HT↩ TP2})

  *Defines HTTP2 ALPN.*
- const static AlpnMode vwg::tls::ALPN_ANY = AlpnMode(std::vector<IANAProtocol>{IANAProtocol::HTTP2, IANAProtocol::HTTP})

  *Defines all supported ALPN.*
- const static std::string vwg::tls::CSUSDefaultStr = "default"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.*
- const static std::string vwg::tls::CSUSDefaulWithSoftFailtStr = "default_with_soft_fail"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSDefault for more detail.*
- const static std::string vwg::tls::CSUSLegacyStr = "legacy"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLegacy for more detail.*
- const static std::string vwg::tls::CSUSLongtermSecureStr = "longterm_secure"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSLongtermSecure for more detail.*
- const static std::string vwg::tls::CSUSIanaRecommendedStr = "iana_recommended"

  *Defines a string constant for the cipher suits set, with is parallel to the enum. a string is more flexible for the interface design, but not as an enum. therefore the enum is used inside the TLS library. see TLSCipherSuiteUseCases↩ Settings::CSUSIanaRecommended for more detail.*
- const UInt32 vwg::tls::MODE_BLOCKING = 0
- const UInt32 vwg::tls::MODE_ASYNC = 1

## 9.6 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TL↩ SCertStore.h File Reference

### Functions

- CertStoreID createMOSKeyStore ()

### 9.6.1 Function Documentation

#### 9.6.1.1 createMOSKeyStore()

```
CertStoreID createMOSKeyStore ( )
```

**Copyright**

Experimantal API for a x509 keystore This is not part of the TLS API, but will belong to the set of API needed to implement features for the backend TLS. enum keystores list all MOS keystores create a MOS keystore

## 9.7 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TL↩ SLibApi.h File Reference

```
#include <memory>
#include "TLSResult.h"
#include "InetAddress.h"
#include "TLSSockets.h"
#include "TLSSocketFactory.h"
```

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Functions

- ITLSSocketFactoryResult vwg::tls::initTLSLib ()

  *This is the entry point for the library. This will return the Socket factory when all initialization needed are successfully performed. These is basically initialization of:*
- void vwg::tls::cleanupTLSLib ()

  *Use this method to cleanup the implementation. This can be used to cleanup the TLS library (e.g. Wolf SSL or Botan SSL). after this the ITLSSocketFactory will not return any socket instance.*

## 9.8 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TL↩ SResult.h File Reference

```
#include <TLSReturnCodes.h>
#include <memory>
#include <cassert>
#include "vwgtypes.h"
```

### Classes

- struct vwg::tls::TLSResult< T >

  *This is a struct to return the return code or the value in case the operation is performed successful. Basically it will take a payload or an return code. One can assume that the paylod is empty if the operation failed. One have to use failed or succeeded first to check if the payload is set or not first. Currently it is assumed that the access of a empty payload will fail and an error is raised.*

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call* ***initTLSLib*** *to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

## 9.9 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/TL↩ SReturnCodes.h File Reference

```
#include "vwgtypes.h"
```

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call* ***initTLSLib*** *to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Enumerations

- enum vwg::tls::TLSReturnCodes : Int32 {
  vwg::tls::RC_TLS_SUCCESSFUL = 0, vwg::tls::RC_TLS_INIT_FAILED = 1, vwg::tls::RC_TLS_CONNECT_FAILED,
  vwg::tls::RC_TLS_ACCEPT_FAILED,
  vwg::tls::RC_TLS_INVALID_DOMAIN, vwg::tls::RC_TLS_KEY_MISSING, vwg::tls::RC_TLS_KEY_ERROR,
  vwg::tls::RC_TLS_USAGE_AFTER_CLEANUP,
  vwg::tls::RC_TLS_IO_ERROR, vwg::tls::RC_TLS_WOULD_BLOCK_READ, vwg::tls::RC_TLS_WOULD_BLOCK_WRITE,
  vwg::tls::RC_TLS_PEER_CLOSED,
  vwg::tls::RC_TLS_AUTHENTIC_TIMECHECK_FAILED, vwg::tls::RC_TLS_MAX_PERMITTED_DEVIATION,
  vwg::tls::RC_TLS_SEND_AFTER_SHUTDOWN, vwg::tls::RC_TLS_INVALID_IP = 1000,
  vwg::tls::RC_TLS_DROPPING_NOTSUPPORTED, vwg::tls::RC_TLS_DROPPING_FAILED, vwg::tls::RC_TLS_PUBLIC_KEY
  vwg::tls::RC_TLS_UNEXPECTED_MESSAGE = 2010,
  vwg::tls::RC_TLS_BAD_RECORD_MAC = 2020, vwg::tls::RC_TLS_RECORD_OVERFLOW = 2022,
  vwg::tls::RC_TLS_DECOMPRESSION_FAILURE = 2030, vwg::tls::RC_TLS_HANDSHAKE_FAILURE =
  2040,
  vwg::tls::RC_TLS_BAD_CERTIFICATE = 2042, vwg::tls::RC_TLS_UNSUPPORTED_CERTIFICATE = 2043,
  vwg::tls::RC_TLS_CERTIFICATE_REVOKED = 2044, vwg::tls::RC_TLS_CERTIFICATE_EXPIRED = 2045,
  vwg::tls::RC_TLS_CERTIFICATE_UNKNOWN = 2046, vwg::tls::RC_TLS_ILLEGAL_PARAMETER = 2047,
  vwg::tls::RC_TLS_UNKOWN_CA = 2048, vwg::tls::RC_TLS_UNKNOWN_CA = 2048,
  vwg::tls::RC_TLS_ACCESS_DENIED = 2049, vwg::tls::RC_TLS_DECODE_ERROR = 2050, vwg::tls::RC_TLS_DECRYPT_EF
  = 2051, vwg::tls::RC_TLS_PROTOCOL_VERSION = 2070,
  vwg::tls::RC_TLS_INSUFFICIENT_SECURITY = 2071, vwg::tls::RC_TLS_NO_RENEGOTIATION = 2100,
  vwg::tls::RC_TLS_UNSUPPORTED_EXTENSION = 2110, vwg::tls::RC_TLS_CERTIFICATE_UNOBTAINABLE
  = 2111,
  vwg::tls::RC_TLS_UNRECOGNIZED_NAME = 2112, vwg::tls::RC_TLS_BAD_CERTIFICATE_STATUS_RESPONSE
  = 2113, vwg::tls::RC_TLS_BAD_CERTIFICATE_HASH_VALUE = 2114, vwg::tls::RC_TLS_NO_APPLICATION_PROTOCOL
  = 2120,
  vwg::tls::RC_TLS_TEE_ACCESS_ERROR = 3000, vwg::tls::RC_TLS_CERTSTORE_NOT_FOUND,
  vwg::tls::RC_TLS_UNKNOWN_CLIENT_CERTIFICATE_SET_ID, vwg::tls::RC_TLS_CLIENT_CERTIFICATE_SET_IDERROF
  vwg::tls::RC_TLS_PROGRAMMING_ERROR_RESULT = -1000 }

## 9.10 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/T↩ LSSession.h File Reference

```
#include <functional>
#include <string>
#include <memory>
#include "TLSApiTypes.h"
#include "vwgtypes.h"
#include "TLSReturnCodes.h"
```

### Classes

- class vwg::tls::ITLSSessionEndpoint

  *Represents a communication session between a service provider and a service consumer. This interface must be implemented by the supplier.*

## Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

## Typedefs

- using vwg::tls::SPITLSSessionEndpoint = std::shared_ptr< ITLSSessionEndpoint >
- using vwg::tls::TLSSessionStatusListener = std::function< void(SPITLSSessionEndpoint endpoint, const T↩ LSSessionStatus status)>
- using vwg::tls::TLSDropStatusListener = std::function< void(SPITLSSessionEndpoint endpoint, const TLS↩ DropStatus status)>
- using vwg::tls::SPTLSSessionEndpoint = std::shared_ptr< ITLSSessionEndpoint >
- using vwg::tls::TLSSessionEndpointResult = TLSResult< SPTLSSessionEndpoint >

## Enumerations

- enum vwg::tls::TLSDropStatus : UInt32 {
  vwg::tls::TLSDROP_SECURED, vwg::tls::TLSDROP_DROPPED, vwg::tls::TLSDROP_REQUESTED, vwg::tls::TLSDROP_SEND_LOCKED,
  vwg::tls::TLSDROP_PERFORMED }
- enum vwg::tls::TLSSessionStatus : UInt32 { vwg::tls::TLSSESSION_SECURED, vwg::tls::TLSSESSION_UNSECURED, vwg::tls::TLSSESSION_BROKEN, vwg::tls::TLSSESSION_CLOSED }

  *Defines the possible status values of the session.*

## Variables

- const int vwg::tls::TLS_EOF = 0

  *Defines the EOF value 0 in case that the connection is closed. This can happen if a closed on a socket is made and there are pending receive and send. Please be aware of that EOF is defined as -1.*

## 9.11 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/T↩ LSSocketFactory.h File Reference

```
#include <memory>
#include <vector>
#include "vwgtypes.h"
#include "TLSApiTypes.h"
#include "TLSSession.h"
#include "TLSSockets.h"
#include "IOStream.h"
#include "CipherSuitesDefenitions.h"
```

## Classes

- class vwg::tls::ITLSSocketFactory

  *This is the interface of the socket factory. One need to get an instance of this interface to create a server or a client socket. Use the function initTLSLib to get the instance of the factory. The implementation will have only one instance of the factory.*

## Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

## Typedefs

- using vwg::tls::ClientCertificateSetID = std::string
- using vwg::tls::HashSha256 = std::vector< char >
- using vwg::tls::CertStoreID = std::string
- using vwg::tls::ITLSSocketFactoryResult = TLSResult< std::shared_ptr< ITLSSocketFactory > >

## Variables

- const ClientCertificateSetID vwg::tls::CLINET_CERTICATE_SET_BASE = "BASE"

## 9.12 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAPI/includes/T↩LSSockets.h File Reference

```
#include <memory>
#include "vwgtypes.h"
#include "TLSApiTypes.h"
#include "TLSResult.h"
#include "TLSSession.h"
```

## Classes

- class vwg::tls::ITLSServerSocket

  *Server TLS-PSK aware server socket interface. This interface must be implemented by the supplier.*
- class vwg::tls::ITLSClientSocket

  *Server TLS-PSK aware client socket interface. This interface must be implemented by the supplier.*

## Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::tls

### Typedefs

- using vwg::tls::SPTLSClientSocket = std::shared_ptr< ITLSClientSocket >
- using vwg::tls::SPTLSServerSocket = std::shared_ptr< ITLSServerSocket >
- using vwg::tls::TLSClientSocketResult = TLSResult< SPTLSClientSocket >
- using vwg::tls::TLSServerSocketResult = TLSResult< SPTLSServerSocket >

## 9.13 /home/dor/projects/e3_security_tlsapi/tlsAPI-WS/tlsAP←I/includes/vwgtypes.h File Reference

```
#include <cstdint>
#include <array>
```

### Namespaces

- vwg

  *This is the entry point of the library, basically one user have to call **initTLSLib** to create a factory in order to retrieve the objects for the communication between provider and consumer.*
- vwg::types

### Typedefs

- using vwg::types::Boolean = bool
- typedef std::uint8_t vwg::types::UInt8
- typedef std::uint16_t vwg::types::UInt16
- typedef std::uint32_t vwg::types::UInt32
- typedef std::uint64_t vwg::types::UInt64
- typedef std::int8_t vwg::types::Int8
- typedef std::int16_t vwg::types::Int16
- typedef std::int32_t vwg::types::Int32
- typedef std::int64_t vwg::types::Int64
- using vwg::types::Byte = UInt8
- using vwg::types::UUID = std::array< UInt8, 16 >

# Index