TECHNICAL SPECIFICATION

ISO/TS 21177

First edition
2019-08

Intelligent transport systems — ITS station security services for secure session establishment and authentication between trusted devices

Reference number
ISO/TS 21177:2019(E)

Contents

Page

Foreword vi
Introduction vii
1 Scope 1
2 Normative references 1
3 Terms and definitions 1
4 Symbols and abbreviated terms 2
5 Overview 3
5.1 Goals 3
5.2 Architecture and functional entities 4
5.3 Cryptomaterial handles 7
5.4 Session IDs and state 7
5.5 Access control and authorisation state 8
5.6 Application level non-repudiation 8
5.7 Service primitive conventions 8
6 Process flows and sequence diagrams 9
6.1 General 9
6.2 Overview of process flows 9
6.3 Sequence diagram conventions 10
6.4 Configure 11
6.5 Start Session 12
6.6 Send data 14
6.7 Send access control PDU 17
6.8 Receive PDU 18
6.9 Secure connection brokering 23
6.9.1 Goals 23
6.9.2 Prerequisites 24
6.9.3 Overview 24
6.9.4 Detailed specification 25
6.10 Force end session 33
6.11 Session terminated at session layer 35
6.12 Deactivate 35
6.13 Secure session example 36
7 Security Subsystem: interfaces and data types 38
7.1 General 38
7.2 Access control policy and state 39
7.3 Enhanced authentication 40
7.3.1 Definition and possible states 40
7.3.2 States for owner role enhanced authentication 40
7.3.3 State for accessor role enhanced authentication 41
7.3.4 Use by Access Control 42
7.3.5 Methods for providing enhanced authentication 42
7.3.6 Enhanced authentication using SPAKE2 42
7.4 Extended authentication 43
7.5 Data types 44
7.5.1 General 44
7.5.2 Imports 44
7.5.3 Iso21177AccessControlPdu 44
7.5.4 AccessControlResult 44
7.5.5 ExtendedAuthPdu 44
7.5.6 ExtendedAuthRequest 45
7.5.7 InnerExtendedAuthRequest 45
7.5.8 AtomicExtendedAuthRequest 46

7.5.9 ExtendedAuthResponse 46
7.5.10 ExtendedAuthResponsePayload 46
7.5.11 EnhancedAuthPdu 47
7.5.12 SpakeRequest 47
7.5.13 SpakeResponse 47
7.5.14 SpakeRequesterResponse 48
7.6 App-Sec Interface 48
7.6.1 App-Sec-Configure.request 48
7.6.2 App-Sec-Configure.confirm 49
7.6.3 App-Sec-StartSession.indication 49
7.6.4 App-Sec-Data.request 50
7.6.5 App-Sec-Data.confirm 50
7.6.6 App-Sec-Incoming.request 51
7.6.7 App-Sec-Incoming.confirm 51
7.6.8 App-Sec-EndSession.request 52
7.6.9 App-Sec-EndSession.confirm 52
7.6.10 App-Sec-EndSession.indication 52
7.6.11 App-Sec-Deactivate.request 53
7.6.12 App-Sec-Deactivate.confirm 53
7.6.13 App-Sec-Deactivate.indication 53
7.7 Security Subsystem internal interface 54
7.7.1 General 54
7.7.2 Sec-AuthState.request 54
7.7.3 Sec-AuthState.confirm 55

8 Adaptor Layer: Interfaces and data types 55
8.1 General 55
8.2 Data types 56
8.2.1 General 56
8.2.2 Iso21177AdaptorLayerPDU 56
8.2.3 Apdu 57
8.2.4 Access Control 57
8.2.5 TlsClientMsg1 57
8.2.6 TlsServerMsg1 57
8.3 App-AL Interface 57
8.3.1 App-AL-Data.request 57
8.3.2 App-AL-Data.confirm 58
8.3.3 App-AL-Data.indication 58
8.3.4 App-AL-EnableProxy.request 59
8.4 Sec-AL Interface 61
8.4.1 Sec-AL-AccessControl.request 61
8.4.2 Sec-AL-AccessControl.confirm 61
8.4.3 Sec-AL-AccessControl.indication 61
8.4.4 Sec-AL-EndSession.request 62
8.4.5 Sec-AL-EndSession.confirm 62

9 Secure Session services 62
9.1 General 62
9.2 App-Sess interfaces 62
9.2.1 App-Sess-EnableProxy.request 62
9.3 Sec-Sess interface 63
9.3.1 Sec-Sess-Configure.request 63
9.3.2 Sec-Sess-Configure.confirm 65
9.3.3 Sec-Sess-Start.indication 65
9.3.4 Sec-Sess-EndSession.indication 66
9.3.5 Sec-Sess-Deactivate.request 66
9.3.6 Sec-Sess-Deactivate.confirm 67
9.4 AL-Sess interface 67
9.4.1 AL-Sess-Data.request 67

9.4.2 AL-Sess-Data.confirm ... 67
9.4.3 AL-Sess-Data.indication ... 68
9.4.4 AL-Sess-EndSession.request ... 68
9.4.5 AL-Sess-EndSession.confirm ... 68
9.4.6 AL-Sess-ClientHelloProxy.request ... 69
9.4.7 AL-Sess-ClientHelloProxy.indication ... 69
9.4.8 AL-Sess-ServerHelloProxy.request ... 70
9.4.9 AL-Sess-ServerHelloProxy.indication ... 70
9.4.10 AL-Sess-EndSession.request ... 71
9.4.11 AL-Sess-EndSession.confirm ... 72
9.5 Permitted mechanisms ... 72
9.5.1 TLS 1.3 ... 72
9.5.2 DTLS 1.3 ... 73
Annex A (informative) **Usage scenarios** ... **74**
Annex B (normative) **ASN.1 module** ... **81**
Bibliography ... **82**

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document is about ITS station security services required to ensure the authenticity of the source and confidentiality and integrity of application activities taking place between ***trusted devices***.

The trust relation between two devices is illustrated in Figure 1. Two devices cooperate in a trusted way, i.e. exchange information with optional explicit bi-directional protection.

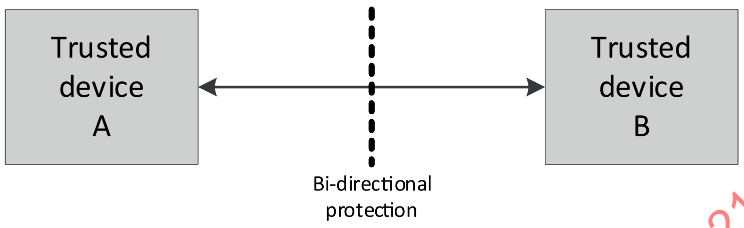


Figure 1 — Interconnection of trusted devices

According to ISO 21217, an ITS station unit (ITS-SU), i.e. the physical implementation of the ITS station (ITS-S) functionality, is a trusted device, and an ITS-SU may be composed of ITS station communication units (ITS-SCU) that are interconnected via an ITS station-internal network. Thus an ITS-SCU is the smallest physical entity of an ITS-SU that is referred to as a trusted device.

NOTE 1 ISO 21217 fully covers the functionality of EN 302 665[15], which is a predecessor of ISO 21217.

NOTE 2 An ITS-SU can be composed of ITS-SCUs from different vendors where each ITS-SCU is linked to a different ITS-SCU configuration and management centre specified in ISO 24102-2[5] and ISO 17419. Station-internal management communications between ITS-SCUs of the same ITS-SU is specified in ISO 24102-4[7]. European C-ITS regulation refers to the "ITS-SCU configuration and management centre" as "C-ITS station operator" meaning the entity responsible for the operation of a C-ITS station. The C-ITS station operator can be responsible for the operation of one single C-ITS station (fixed or mobile), or a C-ITS infrastructure composed of a number of fixed C-ITS stations, or a number of mobile ITS-Stations.

Four implementation contexts of communication nodes in ITS communications networks are identified in the ITS station and communication architecture ISO 21217, each comprised of ITS-station units (ITS-SU) taking on a particular role; personal, vehicular, roadside, or central. These ITS-SUs are ITS-secured communication nodes as required in ISO 21217 that participate in a wide variety of ITS services related to, e.g. sustainability, road safety and transportation efficiency.

Over the last decade, ITS services have arisen that require secure access to data from Sensor and Control Networks (SCN), e.g. from In-Vehicle Networks (IVN) and from Infrastructure/Roadside Networks (IRN), some of which require secure local access to time-critical information; see Figures 2 and 3.

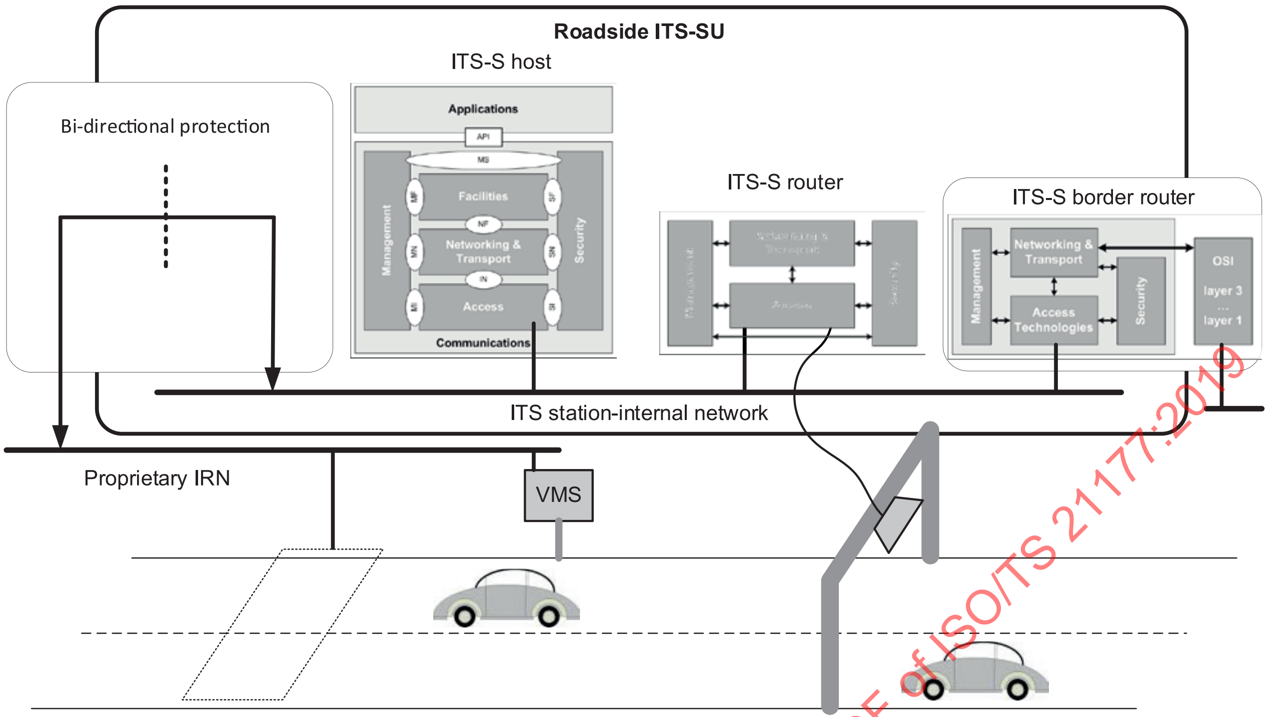


Figure 2 — Example of a roadside ITS-SU connected with proprietary IRN

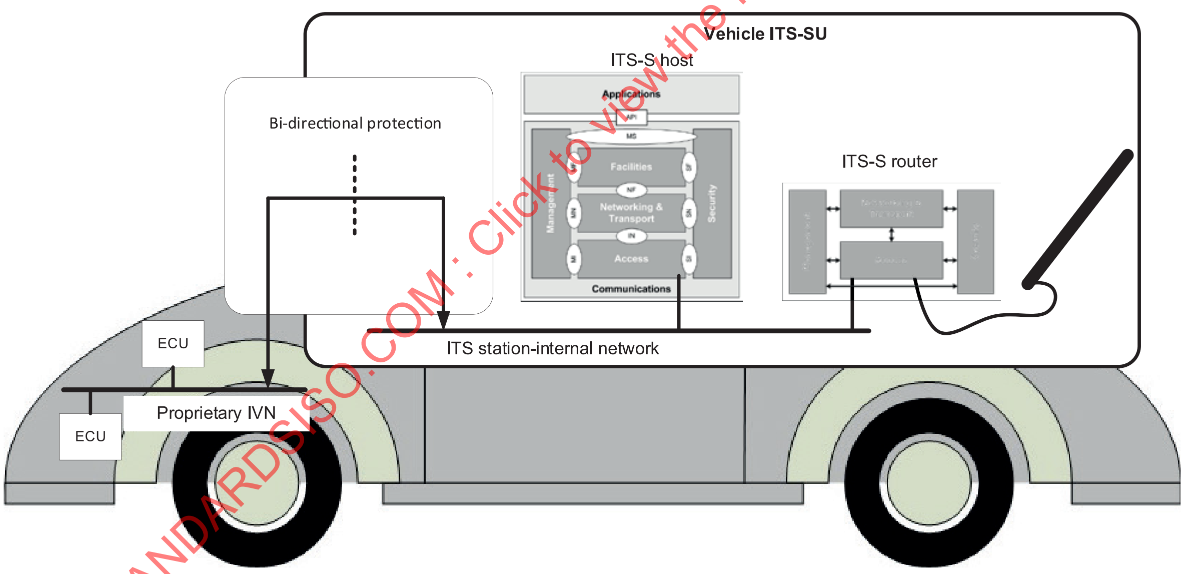


Figure 3 — Example of a vehicle ITS-SU connected with proprietary IRN

Trust in the ITS domain primarily is between ITS Station Communication Units (ITS-SCUs) introduced in ISO 21217; see Figure 4.

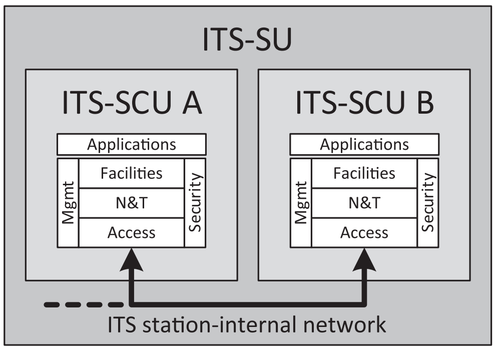


Figure 4 — Interconnection of ITS-SCUs in an ITS-SU

ITS-SCUs are interconnected via an ITS station-internal network. Applying basic security means specified in this document, the ITS-SCUs trust each other. Additionally, protocol data units exchanged between ITS-SCUs may be further protected by additional means, e.g. applying encryption. Major application domains of secure communications between ITS-SCUs of the same ITS-SU are local station management specified in ISO 24102-1[4] using station-internal management communications specified in ISO 24102-4[7].

Trust in the ITS domain further is between ITS-SUs introduced in ISO 21217; see Figure 5.

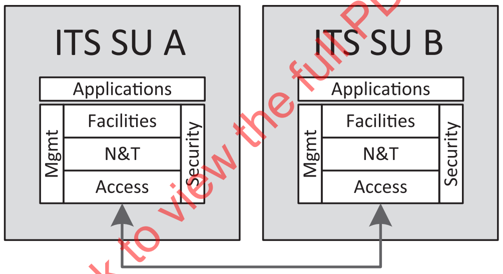


Figure 5 — Interconnection of ITS-SUs

Applying basic security means specified in this document, the ITS-SUs can establish secure application sessions. Establishment of sessions either needs a-priori knowledge about a session partner or can be achieved by means of service announcement specified in ISO 22418[3]. Further on, broadcast of messages is secured by means of authenticating the sender of such a message, applicable for the service advertisement message (SAM) specified in ISO/TS 16460[1] and used in ISO 22418[3]. Additionally, other security means may be applied, e.g. encryption of messages.

A further trust relation in the ITS domain is between an ITS-SU consisting of one or several ITS-SCUs and a sensor and control network (SCN). Trust is achieved by applying security means in an interface as illustrated in Figure 6 with details specified in this document.

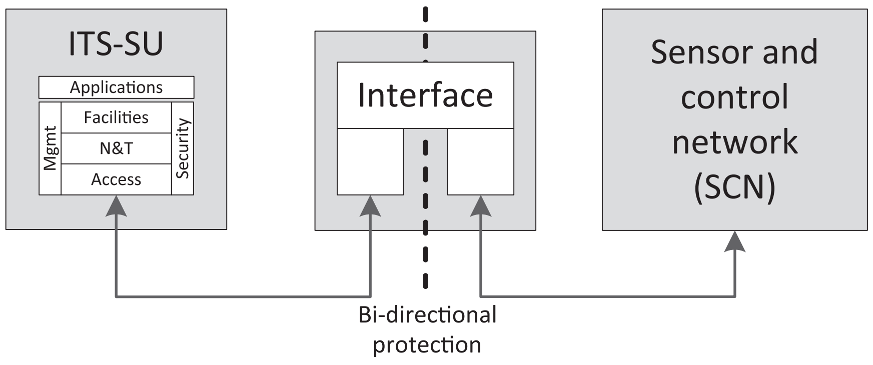


Figure 6 — Interface between ITS-SU and sensor and control network

The interface presented in Figure 6 may be a stand-alone device, or may be integrated in the ITS-SU, or may be part of the SCN. Examples of SCNs are "In-Vehicle Networks" (IVN) and "Infrastructure/Roadside Networks" (IRN).

Related use cases of these ITS services have largely been derived from regulatory requirements and ITS operational needs, and they include:

— secure real-time access to time-critical vehicle-related data for safety of life and property applications, e.g. collision avoidance, emergency electronic brake light and event determination;

— secure local access to detailed real-time data for efficiency applications (traffic management), e.g. intersection interaction, congestion avoidance, dynamic priorities;

— protection of private data, e.g. in compliance with the European "General Data Protection Regulation" (GDPR)[16];

— local access to certified real-time data for sustainability applications, e.g. dynamic emission zones (controlled zones as currently standardized in CEN TC 278 within the Project Team PT 1705 funded by the European Commission), intersection priorities based on emissions, interactive optimum vehicle settings to minimize fuel consumption.

There are many use cases of ITS services currently identified where real-time exchange of time-critical information between ITS-SUs in close proximity is essential, and the number will grow, see e.g. the US National ITS Reference Architecture[17]. It is critical that ultimately all ITS-SUs in a given area are able to be engaged in these distributed services. This, in turn, requires vehicle ITS-SUs to have real-time access to vehicle data, and roadside ITS-SUs to have real-time access to infrastructure data. All ITS-SUs need being capable of secure software updates.

According to ISO 21217, an ITS-SCU of an ITS-SU can communicate with devices that, in a strict sense, are not compliant with the architecture specified in ISO 21217. However, in order to have trusted communications, a certain minimum level of security measures must be shared between an ITS-SCU and such an external device. Examples of such external devices are a node in the Internet, or a node in a sensor and control network. In this document, the assumption is made that ITS-S application processes operating on ITS-SUs are issued with *certificates* by a Certificate Authority (CA), and that the CA is a trusted third party in the sense that before issuing the certificate to the ITS-S application process, it ensures that the ITS-SU on which the ITS-S application process is resident meets the minimum security requirements for that application. This allows peer ITS-S application processes which observe that an ITS-S application process possesses a valid certificate to have a level of assurance that the ITS-S application process is in fact secure and trustworthy.

The subject of this document thus is three-fold:

1) Specify ITS station security services for enabling trust between ITS-S application processes running on different ITS-SCUs of the same ITS-SU, i.e. establishing a trusted processing platform, considering also trust inside an ITS-SCU:
   - — protection of applications from the actions of other applications;
   - — protection of shared information;
   - — protection of shared processing resources such as communications software and hardware, which includes methods of prioritisation and restricted access.
2) Specify ITS station security services for enabling trust between ITS-S application processes running on the same ITS-SU.
3) Extend these ITS security services for enabling trust between an ITS-SCU and devices being part of a sensor and control network.

NOTE 3 It is intended to extend the subject of this document in future editions.

Such security services include e.g. the basic security features of:

a) authentication and authorisation;

b) confidentiality and privacy;

c) data integrity;

d) non-repudiation.

Tasks related to communications are:

a) establishing secure sessions for bi-directional communications, e.g. based on service advertisement specified in ISO 22418[3];

b) authenticating a sender of broadcast messages, e.g. CAM, DENM, BSM, SPaT, MAP, FSAM, WSA;

c) encrypting messages.

NOTE 4 Tasks b) and c) above related to communications are already specified in other standards, see e.g. IEEE Std. 1609.2™ and several related standards from ETSI TC ITS.

Intelligent transport systems — ITS station security services for secure session establishment and authentication between trusted devices

1 Scope

This document contains specifications for a set of ITS station security services required to ensure the authenticity of the source and integrity of information exchanged between trusted entities:

— devices operated as bounded secured managed entities, i.e. "ITS Station Communication Units" (ITS-SCU) and "ITS station units" (ITS-SU) specified in ISO 21217, and

— between ITS-SUs (composed of one or several ITS-SCUs) and external trusted entities such as sensor and control networks.

These services include authentication and secure session establishment which are required to exchange information in a trusted and secure manner.

These services are essential for many ITS applications and services including time-critical safety applications, automated driving, remote management of ITS stations (ISO 24102-2[5]), and roadside/infrastructure related services.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 17419, *Intelligent transport systems — Cooperative systems — Globally unique identification*

IEEE Std 1609.2™, *IEEE Standard for Wireless Access in Vehicular Environments — Security Services for Applications and Management Messages*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
Access Control PDU
PDU generated by the Security Subsystem for purposes of establishing the authorisation status of a peer ITS-S application process

3.2
Access Control Policy
data source governing what access to resources is permissible by peer Applications

3.3
application
functional entity, i.e. an ITS-S application process

3.4
security subsystem
functional entity providing security functionality for use by an ITS-S application process

3.5
Security Adaptor Layer
functional entity providing multiplexing and demultiplexing functionality for data and session control commands

3.6
Secure Session
functional entity providing confidentiality, integrity, authentication, guaranteed in-order delivery, and replay protection on the datagrams that are passed over it

3.7
resources
functional entity constituting endpoints of ITS-S application process activity

3.8
Cryptomaterial Handle
reference to Cryptomaterial allowing that Cryptomaterial to be used in cryptographic operations, i.e. sign, verify, encrypt, decrypt

3.9
Cryptomaterial
cryptographic keys and associated material, either a secret key for a symmetric algorithm, or a private key for an asymmetric algorithm, and the associated public key or certificate

3.10
Secure Session Service
functional entity responsible for establishing secure communications sessions with its peer instances

4 Symbols and abbreviated terms

ACK	acknowledge
ALPDU	adaptor layer PDU
APDU	application protocol data unit
DTLS	datagram TLS
ID	identifier
IRN	infrastructure/roadside network
ITS	intelligent transport systems
ITS-S	ITS station [SOURCE: ISO 21217:2014]
ITS-SCP	ITS station communication profile Note to entry: From ISO 21217

ITS-SU	ITS station unit [SOURCE: ISO 21217:2014]
IVN	in-vehicle network
OSI	open system interconnection
OTP	one time password
PDU	protocol data unit
PDU	protocol data unit
PSID	provider service identifier
SAM	service advertisement message [SOURCE: ISO/TS 16460]
SCN	sensor and control network
SDEE	secure data exchange entity
SPAKE2	secure password authenticated key exchange 2
SRM	service response message [SOURCE: ISO/TS 16460]
SSP	service specific permission
SSTD	secure session between trusted devices
TLS	transport layer security

5 Overview

5.1 Goals

Clause 5 presents the logical architecture followed in this document. The logical architecture is designed to accomplish the following goals:

— Two peer ITS-S application processes can communicate securely, i.e. in an authorized, integrity protected and confidential manner.

— The ITS-S application processes can authenticate to each other using role- or attribute-based access control.

— Each individual incoming application protocol data unit (APDU) can be subject to individual access control processes.

— The security state of the connection (i.e. the authentication status of one ITS-S application process with respect to access to the other connection) can be updated within the secure session as follows.

— An ITS-S application process can prove to the other that it knows a shared secret (Enhanced Authentication, see 7.3) — the intended use of this is to allow the owner or other legitimate operator of one ITS-S application process to permit access by a specific peer ITS-S application process, see Clause 6 for further discussion.

— An ITS-S application process can provide additional authentication within the secure session (Extended Authentication, see 7.4) — for example to provide an identity as well as application permissions, or to provide additional application permissions.

— Secure session communication uses the same credentials as ITS-S application processes use.

— An ITS-S application process can configure a secure session so that it terminates under specific conditions (e.g. timeouts of different kinds) and can also terminate the secure session directly.

— To allow secure and private service discovery, the initialization stage (the "handshake") of one secure session between two peer ITS-S application processes can be proxied over an existing secure session between two different peers.

5.2 Architecture and functional entities

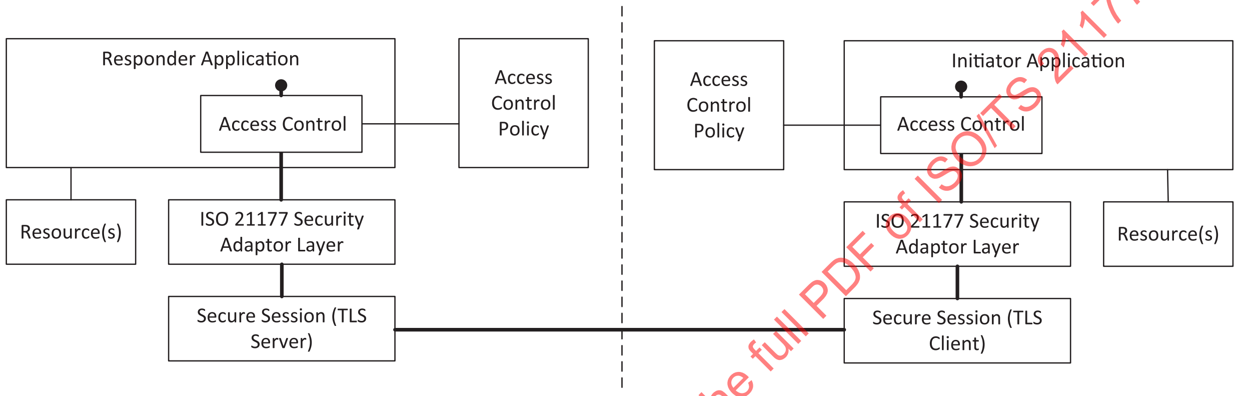


Figure 7 — Logical Architecture

Figure 7 shows the functional entities and data sources involved in secure communications between pairs of ITS-S application processes using mechanisms specified in this document. A pair of ITS-S application processes can reside

— either on the same ITS-SCU,

— or in different ITS-SCUs of the same ITS-SU,

— or in different ITS-SUs,

— or in an ITS-SU and another trusted entity not being a dedicated ITS device.

This description refers in general to activities on the "home trusted entity" and the "peer trusted entity".

NOTE 1 If an entity is described as "fully specified in this document", that includes the case where some of the specification is in external documents that are incorporated in this document by reference.

Functional entities are:

— **Resources:** These are endpoints of ITS-S application process activity and may be inside or outside the ITS-SU on which the ITS-S application process resides.

— **Application:** This is an ITS-S application process that uses input from resources, from a peer ITS-S application process (peer Application), and from its own state to carry out application activities.

NOTE 2 To simplify reading of this document, the term "Application" is used as a replacement for "ITS-S application process", where appropriate. The term "application" is used in its generic meaning.

— **Security Subsystem:** The security subsystem applies security mechanisms to determine what actions the peer Application is permitted to take and implements support functionality in support of making those decisions. The Security Subsystem contains some non-application-specific functionality, specified in this document, and also application-specific functionality (it has to understand the semantics of application PDUs in order to make access control decisions); as such, it is shown within the Application in Figure 7.

The Security Subsystem is configured using input from the Access Control Policy data source. For outgoing communications, it takes as input commands and data from the Application and applies appropriate security processing. For incoming communications, it takes as input commands and data from the Security Adaptor Layer and applies the appropriate access control policy; as a result of applying the access control policy it either passes the input to the application or generates an access control response. See Clause 7.

The Security Subsystem in its own turn contains two distinct functional elements: the Authentication State Subsystem and the Access Control Subsystem. The Authentication State Subsystem stores the authentication state of the secure session, and can also generate Access Control request PDUs to enable this state to be updated. The Access Control Subsystem makes decisions on the permissibility of actions requested by the peer entity in the secure session, using the authentication state as an input. This document defines the Authentication State Subsystem in full, as this system is not application-specific. The Access Control Subsystem is part of the application specification for an individual application and is defined as part of that specification..

— **Security Adaptor Layer:** This is a multiplexer/demultiplexer that allows both data, i.e. communications between the Applications themselves, and session control commands, i.e. communications between peer instances of Security Subsystem or the Security Adaptor Layer, to be sent over the same secure session. See Clause 8.

— **Secure session:** This provides confidentiality, integrity, authentication, guaranteed in-order delivery, and replay protection on the datagrams that are passed over it. In this version of this document there are two types of secure session:

  — **Cryptographic secure session:** this uses cryptography to achieve the listed security properties. Any secure session which passes outside the secure boundary of the ITS-SU shall be a cryptographic secure session. In this version of this document the only supported cryptographic secure session mechanism is TLS 1.3 with certificates specified in IEEE Std. 1609.2™. See Clause 9.

  — **Physical secure session**: this is a session between two Applications running in the same ITS-SU, i.e. the information flow does not pass outside the ITS-SU secure boundary. In this case, because the ITS-SU is a trusted domain, all the security properties listed above are assumed to hold. This document does not provide a specification of a physical secure session but permits the use of a physical secure session.

The "Access Control Policy" data source governs whether a secure session is cryptographic, physical, or both.

Data sources are:

— **Access Control Policy**: Governs what access to resources is permissible by peer applications communicating via the mechanisms in this document. Used as input by the Security Subsystem functional entity. Data from the Access Control Policy is provided to the Secure Session functional entity to configure secure sessions. This document does not provide a specification of the Access Control Policy language.

The relationships between instances of these functional elements and data sources on a single trusted entity are shown in Figure 8. In Figure 8, heavy arrows between functional elements indicate flows containing information that is exchanged with processes on the peer trusted entity, and light arrows with dots between functional elements indicate control flows within the home ITS-SU. Grey dashed arrows within functional elements indicate relationships between different data flows handled by those functional elements.

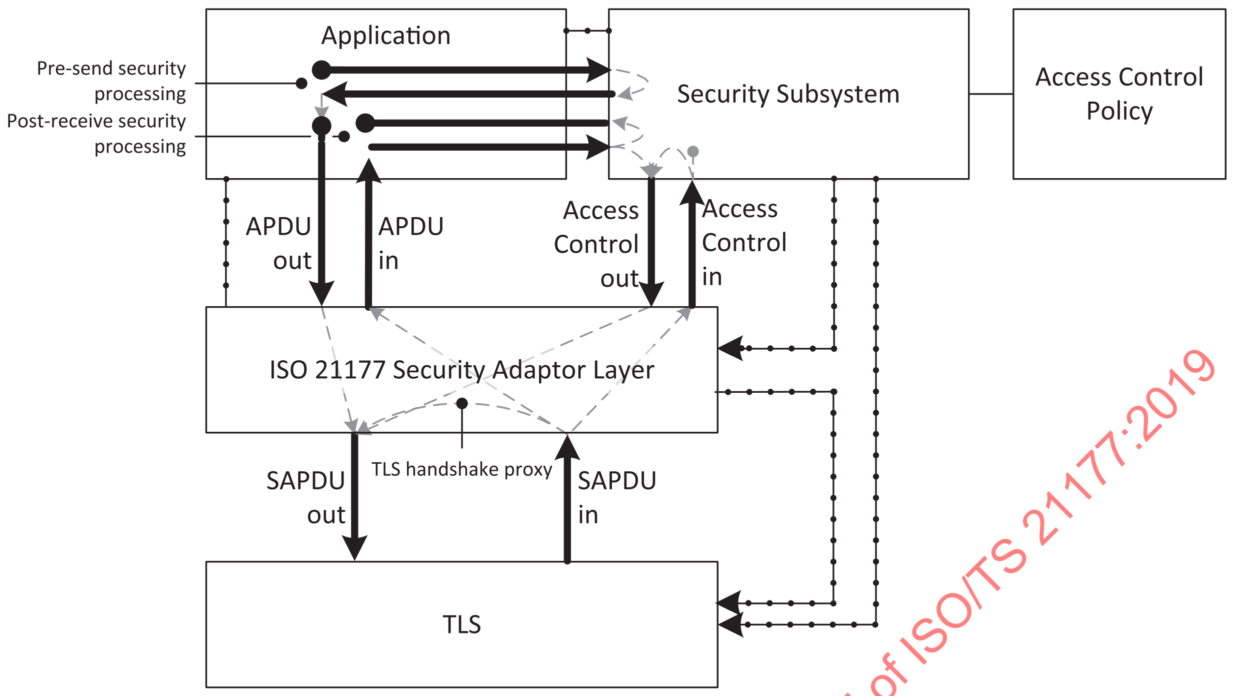


Figure 8 — Interactions between local functional elements

— The **Application** acts as a source and a sink for APDUs.

— The **Application-Security Subsystem** interface is specified in 7.5. It is used:

— Optionally, by the Application, to request IEEE 1609.2 application-level processing for outgoing APDUs specified in IEEE Std. 1609.2™as described in 6.6.

— By the Application, to request that access control services are applied to activities that might be carried out as a result of incoming APDUs. Specifically, in this document, the interface provided is an internal interface between the Authorization State Subsystem and the Access Control Subsystem that allows the Access Control Subsystem to request the authorization state. A full application specification also contains a specification of the interface between the Application and the Access Control Subsystem that allows the Application to request access control decisions and provide full context. This second interface is application-specific and is not provided in this document.

— By the Application, to configure, end, and deactivate secure session services.

— By the Security Subsystem, to notify the Application of a decision to end or deactivate secure session services.

— The **Security Subsystem** is specified in Clause 7. It acts as:

— A source for outgoing Access Control commands, which may be generated without a trigger or in response to incoming Access Control commands or APDUs.

— A sink for incoming Access Control commands, which may result in the Security Subsystem updating its state or in the Security Subsystem generating an outgoing Access Control commands (or both).

— A source and a sink for security configuration commands.

— The **Security Subsystem-Adaptor Layer** interface is specified in 8.4. It is used to exchange APDUs and Access Control PDUs between the Adaptor Layer and the other functional entities, and for the Application or Security Subsystem to control and configure the Adaptor Layer.

- The **Adaptor Layer** is specified in Clause 8. It:
  - Wraps outgoing APDUs and outgoing Access Control PDUs as security Adaptor Layer Protocol Data Units (ALPDUs) and passes them to the Secure Session service.
  - Acts as a proxy for TLS handshake PDUs as specified in 6.9.
  - Inspects incoming ALPDUs and
    - Unwraps incoming APDUs and incoming Access Control PDUs and passes them to the Security Subsystem.
    - Proxies incoming TLS handshake proxy PDUs as specified in 6.9.4.
- The **Security Subsystem-Secure Session service** interface is specified in 9.2. It is used to control and configure persistent properties of the Secure Session service.
- The **Adaptor Layer-Secure Session service** interface is specified in 9.5. It is used to control and configure session-specific properties of the Secure Session service.
- The **Secure Session service** is specified in Clause 9. In this version of this document it is required to be based on TLS, DTLS, or physical secure sessions. It:
  - Receives ALPDUs from the Security Adaptor Layer and treats them as application input to record layer operations, performing outgoing TLS processing on them.
  - Receives secure session datagrams from the network stack and performs incoming TLS processing on them, resulting in ALPDUs which the TLS service passes to the Security Adaptor Layer.

5.3 Cryptomaterial handles

This document uses the concept of Cryptomaterial Handles specified in IEEE Std 1609.2™ to represent logical access to cryptographic keys and, if appropriate, their associated certificates. The reader is referred to IEEE Std 1609.2™ for more details.

5.4 Session IDs and state

The Security Subsystem service, Adaptor Layer service, and Secure Sessions service maintain state information of each (application, session)-tuple.

NOTE The interfaces presented in this document are passing (application ID, session ID)-tuples to the relevant primitives.

Unique identification of an instance of an ITS-S application process of an ITS application is by means of the identifier ITS-SAPIID of ASN.1 type `ITSapiid` specified in ISO 17419.

An implementation shall ensure that state information corresponding to a specific (application, session)-tuple is not made available for use by activities corresponding to a different (application, session)-tuple except through the interfaces defined in this document.

In the case where an Application has the role of the client in a secure session, the session ID is generated by this Application. The Application ensures that the session ID is unique among active sessions for that Application. The Application supplies the session ID to the client via the `App-Sec-Configure.request` and `Sec-Sess-Configure.request` primitives.

In the case where an Application has the role of the server in a secure session, the session ID is generated by the server in response to an incoming connection. The server ensures that the session ID is unique among active sessions for that application. The session ID is provided to the application using the `Sec-Sess-StartSession.indication` and `App-Sec-StartSession.indication` primitives.

5.5 Access control and authorisation state

The Security Subsystem maintains the authorisation state of the session. The authorisation state is the collection of all “authorisation statements” made by the peer party which were valid at the time made and which have not timed out (where the timeout conditions are set by the access control policy). An "authorisation statement" is any PDU sent by the peer party that contains cryptographically protected information about the authorisation of that peer party. Specifically, in this document, the authorisation statements presented in Table 1 are defined.

Table 1 — Authorisation statements

Type	Description	Related subclauses	Service primitives for provision to Security Subsystem
Initial authorisation	The credentials presented during the handshake of the secure session, if one occurs	6.5	`Sec-Sess-StartSession.indication`
Enhanced authorisation	A demonstration that the peer party shares a secret value with the home party, presented during the secure session via an Access Control PDU of type `EnhancedAuthPdu` as specified in 7.5.11	6.7, 6.8, 7.3	`Sec-AL-AccessControl.indication`
Extended authorisation	Additional credentials presented during the secure session via an Access Control PDU of type Extend-edAuthPdu as specified in 7.5.5.	6.7, 6.8, 7.4	`Sec-AL-AccessControl.indication`

This document defines Access Control Request PDUs and Access Control Response PDUs in 7.5.

An Access Control PDU containing an extended authorisation statement may be generated in response to an Access Control Request PDU or may be generated without a request. An Access Control PDU containing an enhanced authorisation statement is always made in response to an Access Control PDU containing an enhanced authorisation request. Sending Access Control PDUs is specified in 6.7. Receiving Access Control PDUs is specified in 6.8.

Exactly how the authorisation statements are used by the Security Subsystem and governed by the Access Control Policy is to be specified in the specification of an application that uses the mechanisms provided by this document.

5.6 Application level non-repudiation

An Application may apply non-repudiation to individual application PDUs by signing them with the mechanisms of IEEE 1609.2. An Application that applies non-repudiation to individual APDUs shall format any non-signed APDUs as Ieee1609Dot2Data structures of type unsecured, to ensure consistent interpretation of the incoming APDUs. A detailed specification of this process is provided in 6.6 for outgoing APDUs and 6.8 for incoming APDUs.

5.7 Service primitive conventions

This document uses service primitives to specify information flows between functional entities, i.e. information flows through "Service Access Points" (SAPs). This document is organized such that each functional entity is specified in a separate clause:

— Security Subsystem in Clause 7,

— Adaptor Layer in Clause 8,

— Secure Session services in Clause 9.

NOTE ISO 24102-3[6] specifies service primitives and service primitive functions for various SAPs introduced in ISO 21217 at the level of ASN.1 type definitions. The service primitive functions identified in this document can be used to complement the specifications in ISO 24102-3[6].

Service primitives are not testable but indicate the information that one functional entity must make available to another in order for the second functional entity to carry out the indicated activities.

Following the conventions of the layered OSI model specified in ISO/IEC 7498-1[10], in this document, a message from a higher layer entity to a lower layer entity is specified with an

— `XXX.request` service primitive of the service *XXX*,

and the response is specified with an

— `XXX.confirm` service primitive.

Similarly a message from a lower layer entity to a higher layer entity is specified with an

— `XXX.indication` service primitive,

and the response from the higher layer entity to the lower layer entity is specified with an

— `XXX.response` service primitive.

The information contained in a service primitive is referred to as "service primitive function"; see ISO 24102-3[6].

This document uses the convention that a `.confirm` service primitive can be correctly associated with its corresponding `.request` service primitive, and a `.response` service primitive can be correctly associated with its corresponding .indication service primitive. In other words, for example, a `.confirm` primitive's parameters do not include an identifier for the corresponding `.request` primitive; the recipient of the `.confirm` primitive is assumed to “just know” which `.request` primitive it is associated with.

6 Process flows and sequence diagrams

6.1 General

Clause 6 specifies process flows supported by the security services specified in this document and the sequence of data flows associated with each of these process flows.

6.2 provides an overview on process flows. 6.3 presents state diagram conventions.

6.2 Overview of process flows

This document supports the following process flows; see Annex A for use cases that make use of these process flows:

— Configure, see 6.4: Prepare the functional elements to participate in secure communications.

— Start Session, see 6.5: Perform processing once a secure session has been established but before data is exchanged.

— Send data, see 6.6: Send data from the home Application to the peer Application.

— Send Access Control PDU, see 6.7: Exchange messages between the home Security Subsystem and the peer Security Subsystem with a goal of updating the state of one or both.

— Receive PDU, see 6.8: Receive an APDU, an Access Control PDU, or a proxied TLS handshake packet.

— Secure connection brokering, see 6.9: Proxy a TLS handshake between two other applications, one of which already has a secure connection to the home Application and one of which already has a secure connection to the peer Application.

— Force end session, see 6.10: Terminate a secure session between the home Application and one peer Application, with the termination initiated by the Application or the Security Subsystem.

— Session terminated at session layer, see 6.11.

— Deactivate, see 6.12: Prevent the functional elements from starting any more sessions with the current set of parameters.

An example of how these process flows are combined to create a full secure session, start to end, is given in 6.13.

6.3 Sequence diagram conventions

The sequence diagrams follow the following graphical conventions. In this description the word "message" is used in the UML sequence diagram sense of a communication between two participants and is not meant to imply any over the air communications.

— In each sequence diagram, each message between participants is associated with a descriptive name and with the identifier of the service primitive that is used to implement that message. For example, in the clipping of Figure 9, the message contains an APDU and is sent using the primitive `App-Sec-Data.request`, see 7.6.4.

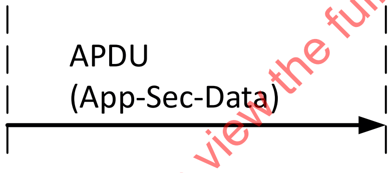


Figure 9 — Convention for sequence diagram — 1

— If the descriptive name is in italics, it is associated with a local message, i.e. a message that does not directly contain data that has passed or is intended to pass to the peer trusted entity. For example, in the clipping of Figure 10, the message *configure* contains only local information and is sent using the `App-Sec-Configure.request` service primitive.

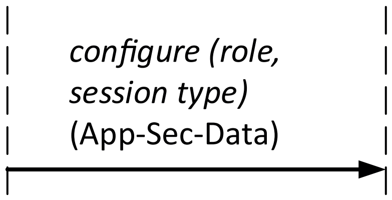


Figure 10 — Convention for sequence diagram — 2

— **Mapping to primitives**: See 5.7 on service primitive conventions. In the sequence diagrams presented in Clause 6, the suffixes `.request`, `.confirm`, `.indication` and `.response` are omitted for compactness. In addition, also for compactness, the `.confirm` and `.response` service primitives are not shown in the diagrams of Clause 6 unless they are significant information flows (in other words, amount to anything other than an ACK). In all cases, the description of the diagrams includes the service primitive suffixes and describes whether the `.confirm` and `.response` service primitives are used as anything other than an ACK.

6.4 Configure

In this process flow the Application configures the Security Subsystem with configuration information necessary to run the secure communications.

The process steps for this process flow are:

a) The Application uses the `App-Sec-Configure.request` primitive, see 7.6.1, to inform the Security Subsystem of:

   1) The role it has in the secure session (client or server).

   2) The socket to be used for communications with the peer trusted entity (this is pass-through information for the Secure Session service). This includes an indication of whether the other endpoint is inside or outside the secure boundary. This socket is not used by the Security Subsystem but is passed on to the Secure Session services via `Sec-Sess-Configure.request`.

   3) If the other endpoint is outside the secure boundary, i.e. if cryptographic security mechanisms are to be used to protect the session:

      i) whether the socket shall be used for reliable or unreliable communications (i.e. TLS or DTLS);

      ii) which cryptomaterial handle should be used for authentication of the secure session.

b) The Security Subsystem:

   1) Uses the Access Policy to determine what secure session types are permitted. In this version of this document, three secure session types are permitted:

      — TLS for reliable transport mechanisms;

      — DTLS for unreliable transport mechanisms;

      — No specific security mechanism for communications sessions within a secure boundary of an ITS-SCU (which are secure by assumption).

   2) Acknowledges the request from the Application using `App-Sec-Configure.confirm`, see 7.6.2, (not shown in the diagram). This may include an indication that the secure session type requested (physical versus cryptographic) is not permitted for this Application by the access control policy. If the secure session type is not permitted, the sequence terminates here. Otherwise, go to the next step.

   3) If a cryptographic secure session is requested, uses the `Sec-Sess-Configure.request`, see 9.3.1, to pass the following to the secure session service:

      i) From the application:

         I) role,

         II) cryptomaterial handle,

         III) socket type.

      ii) From the access control policy:

         I) The permissions constraints to apply to incoming 1609.2 certificates.

         II) The SDEE Identifier used to identify the Application to the 1609.2 Secure Data Service.

         III) The timeout parameters for the session:

            — How long the session may be inactive before a fresh handshake is required.

            — How long the session may be active before a fresh handshake is required.

— If the Secure Session services are playing a server role, how long the service shall be open to incoming requests.

c) The Secure Session service uses `Sec-Sess-Configure.confirm`, see 9.3.2, to ACK the exchange.

A sequence diagram for this process flow is given in Figure 11.

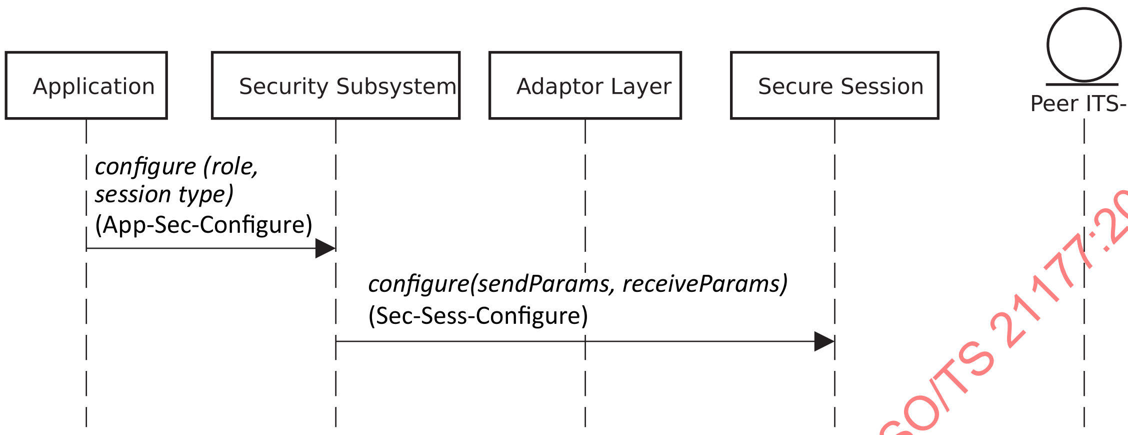

Application
Security Subsystem
Adaptor Layer
Secure Session
Peer ITS-S
configure (role, session type) (App-Sec-Configure)
configure(sendParams, receiveParams) (Sec-Sess-Configure)

Figure 11 — Sequence diagram for Configure process flow

NOTE To change the key (the cryptomaterial handle) to be used for sessions, the application or security subsystem can Deactivate the current Secure Session instance as specified in 6.11 and Configure a new Secure Session instance using the mechanisms of this sub-clause.

6.5 Start Session

In this process flow, following a handshake between the Secure Session service on the home ITS-SU and the Secure Session Service on the peer trusted entity, the Secure Session service provides the Security Subsystem with the credentials received from the peer trusted entity and the Security Subsystem initiates any additional Access Control activities that the Access Policy specifies must complete before data can be exchanged.

In the case of a physical secure session, the session may start without credentials being exchanged. In this case the extended authentication mechanisms of this document may be used to send credentials during the session.

Prequisite:

— The Secure Session is a secure session as defined in 5.2.

— The Secure Session service has been configured using the methods of 6.4.

— The certificate received in the handshake of the secure session (if applicable) matches the constraints provided to the Secure Session service during the Configure process flow.

The process steps for this process flow are:

a) The local Secure Session Service uses the `Sec-Sess-StartSession.indication` service primitive to communicate with the Security Subsystem, see 9.3.3, providing:

1) An indication that the secure session has started and passed the validity conditions that were specified to be checked by the Secure Session Service.

2) The certificate from the Secure Session Service on the peer trusted entity.

   3) If the secure session services are acting in the server role, the Session ID to be used to identify the session; otherwise, the Session ID provided in `Sec-Sess-Configure.request`, see 9.3.1.

b) The Security Subsystem uses the Access Control Policy to determine the status of the incoming connection. The status may be:

   1) Success;

   2) Unauthorised, communication rejected. In this case the Security Subsystem ends the secure session as specified in 6.10;

   3) Additional Access Control actions are necessary. The additional access control actions specified in this document are:

      i) Request Extended Authentication, as specified in 7.4.

      ii) Request Enhanced Authentication, as specified in 7.3.

c) If additional access control actions are to be carried out or if the policy indicates that success or failure are to be communicated to the peer:

   1) The Security Subsystem generates an Access Control PDU and passes it to the Adaptor Layer via the `Sec-AL-AccessControl.request` service primitive, see 8.4.1.

      i) In the case where success is being communicated to the peer, this is an Access Control PDU of type `AccessControlResult.success`, as specified in 7.5.4.

      ii) In the case where failure is being communicated to the peer, this is an Access Control PDU of type `AccessControlResult.failure`, as specified in 7.5.4.

      iii) In the case of Enhanced or Extended Authentication, the contents are as specified in 7.3 and 7.4 respectively.

   2) The Adaptor Layer acknowledges the Access Control PDU via the `Sec-AL-AccessControl.confirm` primitive, see 8.4.2.

   3) The Adaptor Layer wraps the Access Control PDU in an Adaptor Layer PDU of the appropriate subtype as specified in 8.2.

   4) The Adaptor Layer passes the ALPDU to the Secure Session service via the `AL-Sess-Data.request` primitive, see 9.4.1.

   5) The Secure Session service acknowledges the ALPDU via the `AL-Sess-Data.confirm` service primitive, see 9.4.2, secures the ALPDU, and passes it to the network stack for transmission.

d) If the status is success and the secure session services are acting in the server role, the Security Subsystem uses the `App-Sec-StartSession.indication` service primitive, see 7.6.3, to provide the session ID to the Application.

Additional access control requests may be sent within the secure session, see 6.7 (sending) and 6.8 (receiving).

A sequence diagram for this process flow is given in Figure 12.

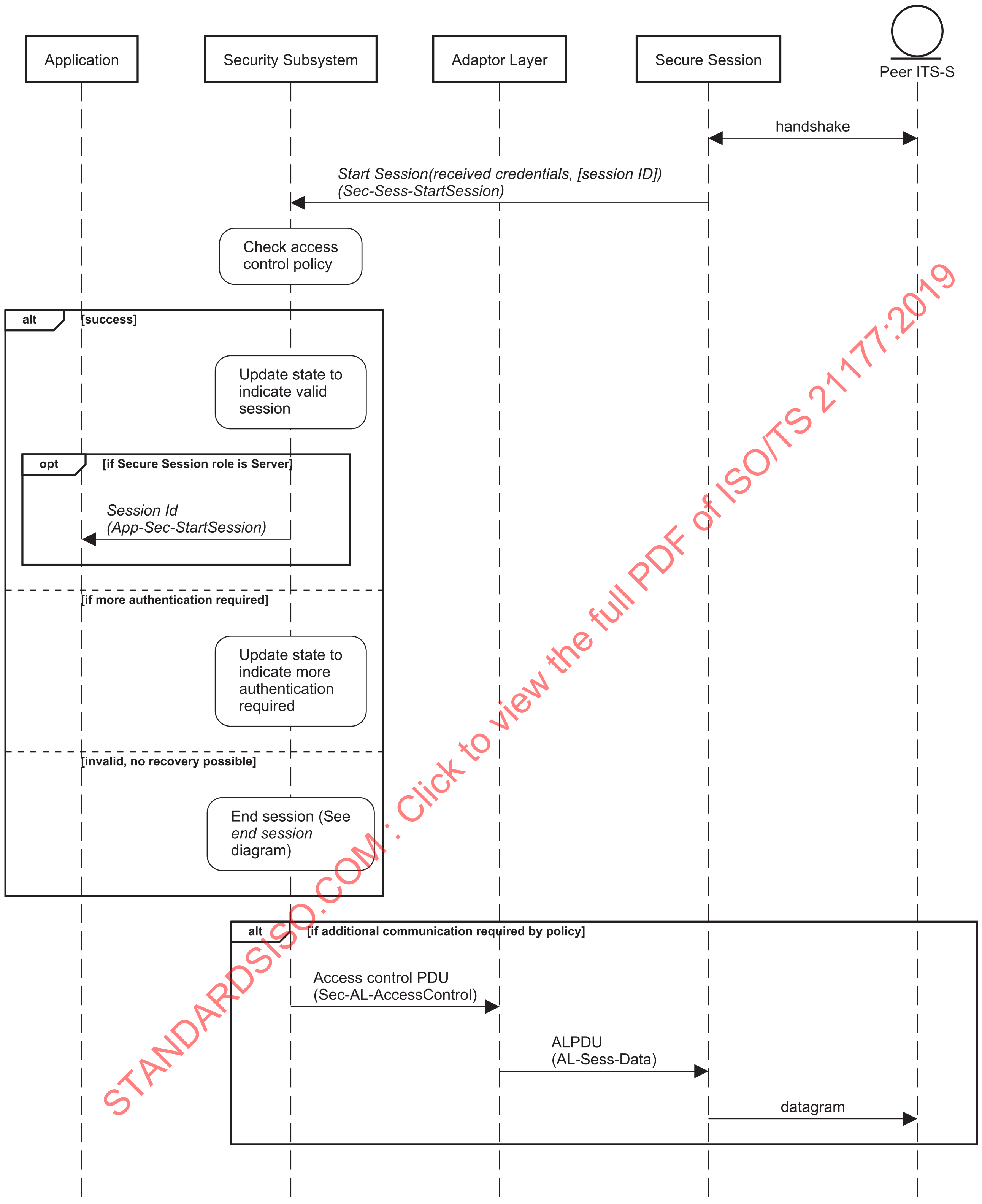

Application
Security Subsystem
Adaptor Layer
Secure Session
Peer ITS-S
handshake
Start Session(received credentials, [session ID])
(Sec-Sess-StartSession)
Check access control policy
alt
[success]
Update state to indicate valid session
opt
[if Secure Session role is Server]
Session Id
(App-Sec-StartSession)
[if more authentication required]
Update state to indicate more authentication required
[invalid, no recovery possible]
End session (See end session diagram)
alt
[if additional communication required by policy]
Access control PDU
(Sec-AL-AccessControl)
ALPDU
(AL-Sess-Data)
datagram

Figure 12 — Sequence diagram for Start Session

6.6 Send data

In this process flow the app on the home ITS-SU sends a datagram over an established ISO/TS 21177 secure session. This sub-clause describes sending activities. The receive-side sequence is specified in 6.8.

Prerequisites:

— The Secure Session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as defined in 5.2, there has been an initialize step using the methods of 6.5.

There are two alternate paths for this process flow. In one path, the Application has the option to apply Application-level non-repudiation to individual APDUs. In the other, the Application never applies Application-level non-repudiation to individual APDUs. This document does not provide a mechanism to negotiate which of these paths is followed; this choice is assumed to be a part of the Application specification.

The process steps for this process flow are:

a) If the Application has the option to apply Application-level non-repudiation to individual APDUs:

  1) If the current APDU shall apply non-repudiation, the Application uses `App-Sec-Data.request`, see 7.6.4, to send the APDU to the Security Subsystem for signing. The Security Subsystem returns the signed APDU (or, if appropriate, an error message) via `App-Sec-Data.confirm`, see 7.6.5.

  2) Otherwise, the Application creates an Ieee1609Dot2Data as defined in IEEE 1609.2, of type unsecured, containing the APDU.

b) The Application uses the `App-AL-Data.request` primitive, see 8.3.1, to provide the original APDU or the APDU output from step a), as appropriate, to the Adaptor Layer.

c) The Adaptor Layer:

  1) ACKs the data using `App-Sec-Data.confirm`, see 7.6.5.

  2) Adds the session non-repudiation service if so configured. This service is not specified in this version of this Document but is planned for future versions.

  3) adds the Data header as specified in 8.2, to indicate that the ALPDU is an Application datagram.

  4) The Adaptor Layer provides the data to the Secure Session service via the `AL-Sess-Data.request` primitive, see 9.4.1.

d) The Secure Session service:

  1) ACKs the data using `AL-Sess-Data.confirm`, see 9.4.2;

  2) fragments (if necessary) and cryptographically protects the data and passes it to the network stack for transmission to the peer trusted entity.

A sequence diagram for this process flow is given in Figure 13.

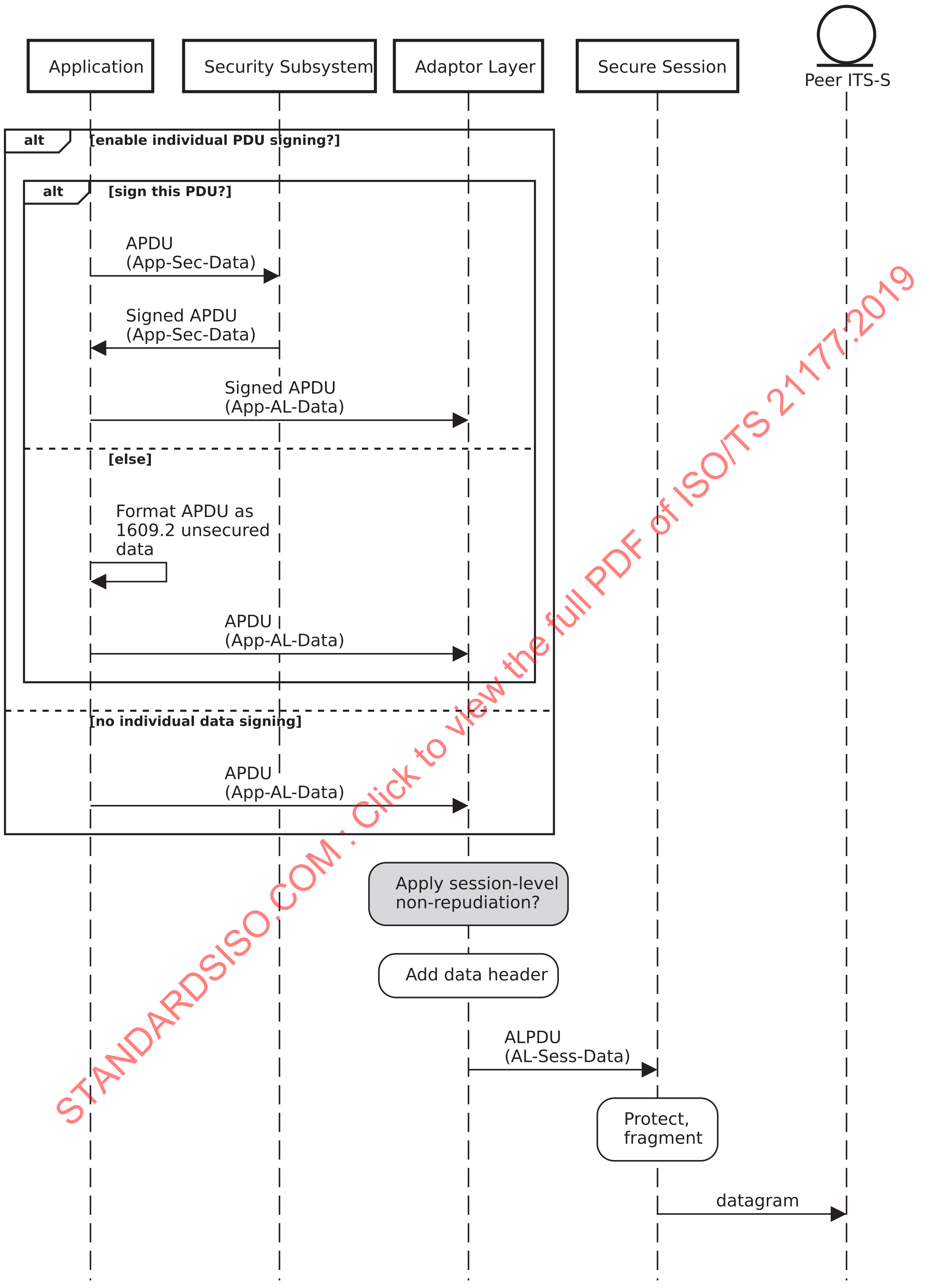


Figure 13 — Sequence diagram for sending data

6.7 Send access control PDU

In this process flow the Security Subsystem creates an access control PDU to establish access control status with the peer Security Subsystem. The access control PDU can be created as a result of a local access policy decision, or as a result of an Application command (through an application/security subsystem interaction to be specified in the specification of an application that uses the mechanisms provided by this document), or as the result of an incoming data or access control PDU. This sub-clause describes sending activities. The corresponding receive-side sequence is specified in 6.8.

Prerequisites:

— The Secure Session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as defined in 5.2, there has been an initialize step using the methods of 6.5.

The process steps for this process flow are:

a) The Security Subsystem generates an Access Control PDU and passes it to the Adaptor layer using the `Sec-AL-AccessControl.request` primitive, see 8.4.1.

NOTE In principle, the Access Control PDU may or may not be individually signed. In this version of this document all the Access Control PDUs are not signed and are authenticated to the peer trusted entity by virtue of being sent over the secure session.

b) The Adaptor Layer:

1) ACKs the Access Control PDU using `Sec-AL-AccessControl.confirm`.

2) Creates an ALPDU, which is an `Iso21177AdaptorLayerPDUA` with the component `messageId` equal to `accessControlId` and the component value equal to the received Access Control PDU, as specified in 8.2.

3) Provides the ALPDU to the Secure Session service via the `AL-Sess-Data.request` primitive, see 9.4.1.

c) The Secure Session service:

1) ACKs the ALPDU using `AL-Sess-Data.confirm`, see 9.4.2;

2) Fragments (if necessary) and cryptographically protects the data and passes it to the network stack for transmission to the peer trusted entity.

A sequence diagram for this process flow is given in Figure 14.

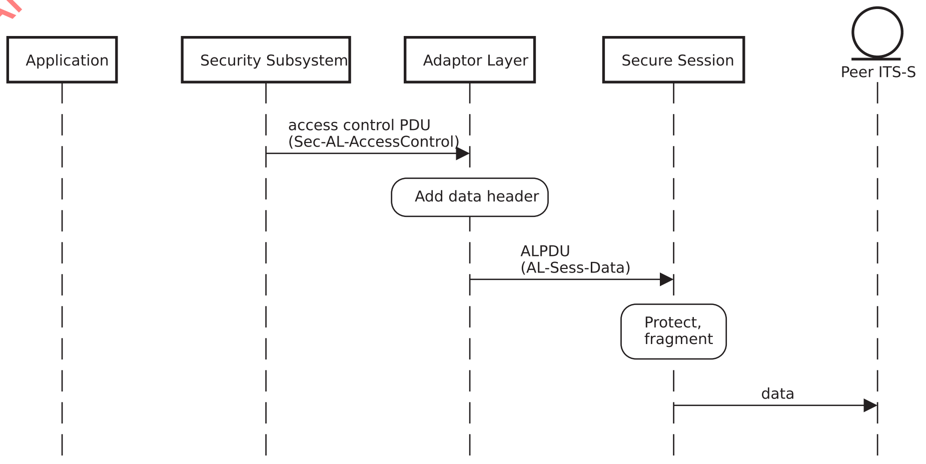


Figure 14 — Sequence diagram for sending Access Control PDU

6.8 Receive PDU

In this process flow the peer trusted entity sends a datagram, which is received by the host ITS-SU and routed to the Secure Session instance servicing the app. This sub-clause describes receiving activities. The send-side sequences are specified in 6.6 and 6.7.

Prerequisites:

— The Secure Session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as defined in 5.2, there has been an initialize step using the methods of 6.5.

The process steps for this process flow are:

a) The Secure Session service receives a datagram from the peer trusted entity and checks that the session has not timed out per the parameters set in b) 3) ii) III) in 6.4.

   1) If the session has timed out, the Secure Session handles it according to the specification of the secure session. See 9.5 for specification of how this is handled for TLS. The process flow then ends.

   2) Otherwise, i.e. if the session has not timed out, the Secure Session decrypts, integrity validates, and defragments the record layer datagram to obtain the ALPDU. It passes the ALPDU to the Adaptor Layer using the `AL-Sess-Data.indication` primitive, see 9.4.3.

b) The Adaptor Layer:

   1) Receives the ALPDU via `AL-Sess-Data.indication`.

   2) If session non-repudiation is specified, checks that the non-repudiation code (i.e. the signature on the session) is valid. This service is not specified in this version of this document but is planned for future versions.

   3) Checks the Adaptor Layer subtype of the ALPDU and acts according to the subtype as specified in the following steps.

c) If the ALPDU is a TLS handshake proxy PDU:

   1) The Adaptor Layer forwards it to the appropriate TLS handshake proxy counterparty as specified in 6.9.4.

d) If the ALPDU is an Access Control PDU:

   1) The Adaptor layer passes it to the Security Subsystem using the `Sec-AL-AccessControl.indication`, see 8.4.3.

   2) The Security Subsystem applies the access control policy to determine what action to take. The outcome is one of the following:

      i) The Access Control PDU is valid and relevant and results in an update of the Security Subsystem state, meaning that there is a change to which incoming data packets are accepted or rejected. See 5.5 for more details.

      ii) The Access Control PDU is not valid or not relevant. The Security Subsystem state does not change. Depending on policy and on the details of the incoming PDU, the Security Subsystem will do one of the following:

         I) Generate and send an Access Control PDU in response as specified in 6.7.

II) Take no action.

e) If the ALPDU is an APDU:

1) The Adaptor layer passes it to the Application using the `App-AL-Data.indication` primitive, see 8.3.3.

2) The Application:

i) Optionally, performs pre-processing based on Application logic to determine whether to accept the APDU. The only decision the Application shall make based on this pre-processing is to take one of the following actions:

I) Discard the APDU without responding or updating the state of the Application or of any resources accessed by the Application. In this case this control flow ends at this point.

II) Pass the APDU to the Security Subsystem for Access Control processing, as specified in step c) below, using the `App-Sec-Incoming.request` primitive, see 7.6.6. This service primitive also includes an indication of whether the session gives the option of using Application-level non-repudiation.

3) The Security Subsystem carries out the following steps. Steps e) 3) i) and e) 3) ii) in 6.4 may be carried out in either order.

i) If the `App-Sec-Incoming.request` primitive indicated that the APDU could have Application-level non-repudiation applied:

I) If the APDU is an Ieee1609Dot2Data of type signed, the Security Subsystem attempts to verify it per IEEE Std. 1609.2™.

If verification fails, the Security Subsystem returns that indication via `App-Sec-Incoming.confirm`, see 7.6.7, and the process flow ends.

If verification succeeds, processing continues.

II) If the APDU is an Ieee1609Dot2Data of type unsecured, processing continues.

III) If the APDU is an Ieee1609Dot2Data of some other type, it is invalid. The Security Subsystem returns that indication via `App-Sec-Incoming.confirm` and the process flow ends.

ii) The Security Subsystem applies the access control policy to the APDU to determine what action to take. In this case the Access Control Subsystem within the Security Subsystem uses `Sec-AuthState.request` and `Sec-AuthState.confirm` to obtain the authorization state. The access control decision is made on the basis of the input PDU contents, the authorization state, and other information provided to the Access Control subsystem. The other information used to make the access control decision is application-specific and an application specification that uses other information will define an application-specific extension to the App-Sec Interface to communicate this other information.

The outcome is one of the following:

I) The APDU is not acceptable per the access control policy. The Security Subsystem indicates this to the Application via `App-Sec-Incoming.confirm`. Depending on policy and on the details of the incoming PDU, the Security Subsystem may generate and send an Access Control PDU in response as specified in 6.7. The process flow then ends.

II) The APDU is acceptable per the access control policy and processing continues.

iii) If this step has been reached, the APDU is acceptable. The Security Subsystem indicates this to the Application via `App-Sec-Incoming.confirm`.

4) The Application may now “make use of” the APDU, e.g. the Application may alter its state, grant access to resources, send responses, etc.

Sequence diagrams for this process flow are given in Figure 15 (overview), Figure 16 (APDUs detail), Figure 17 (Access Control PDUs detail).

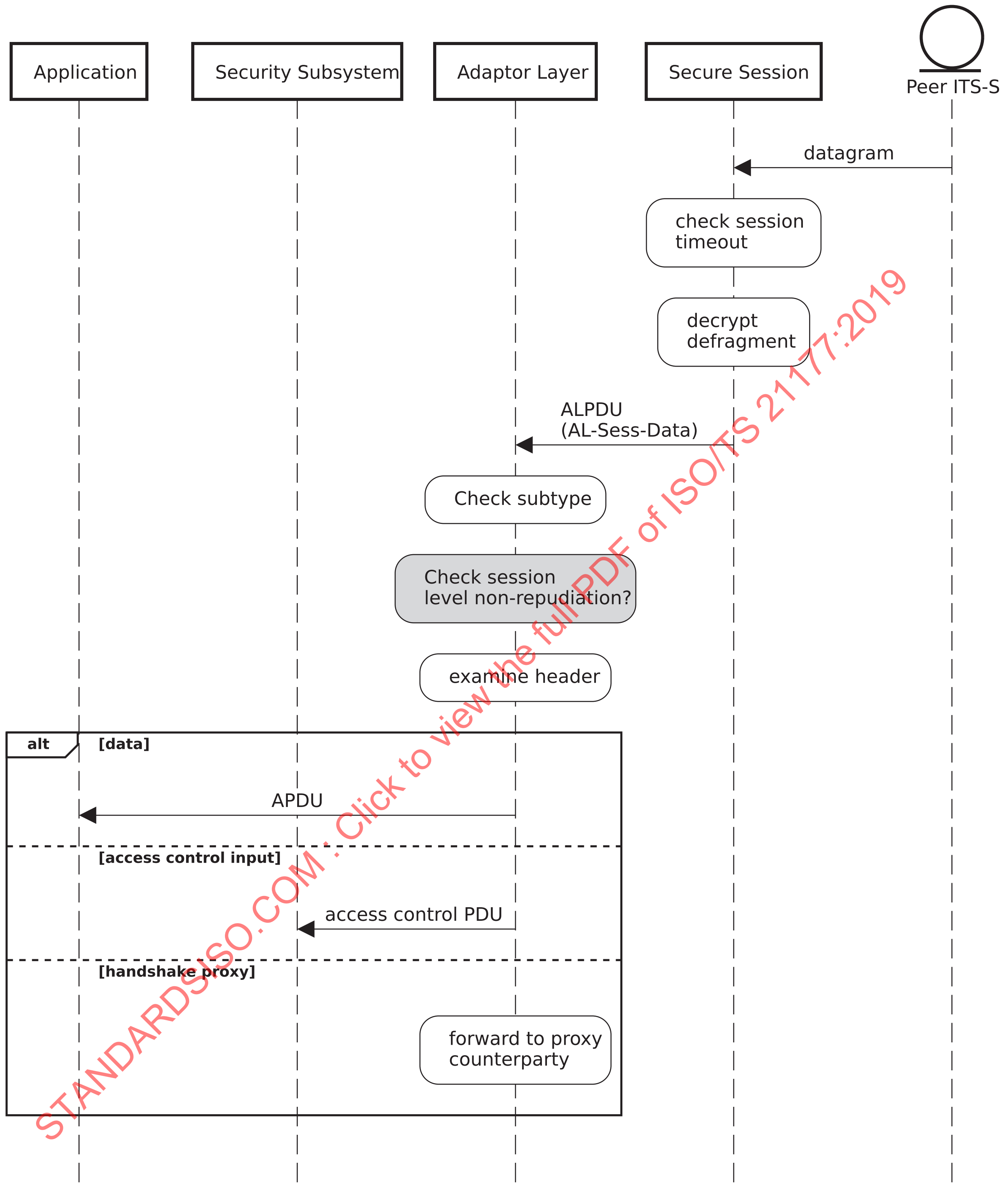


Figure 15 — Sequence diagram for incoming datagrams (Overview)

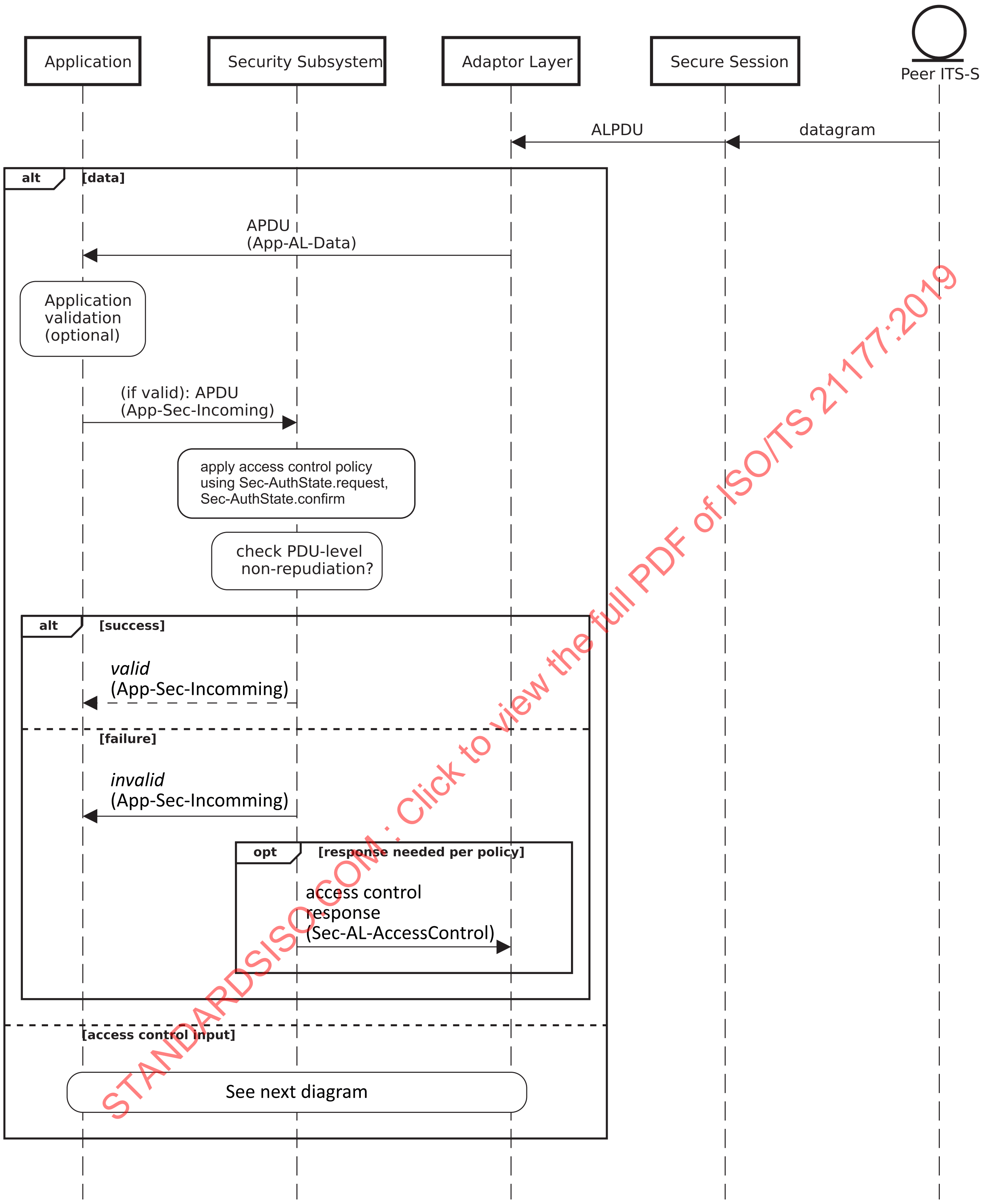


Figure 16 — Sequence diagram for handling of incoming APDUs

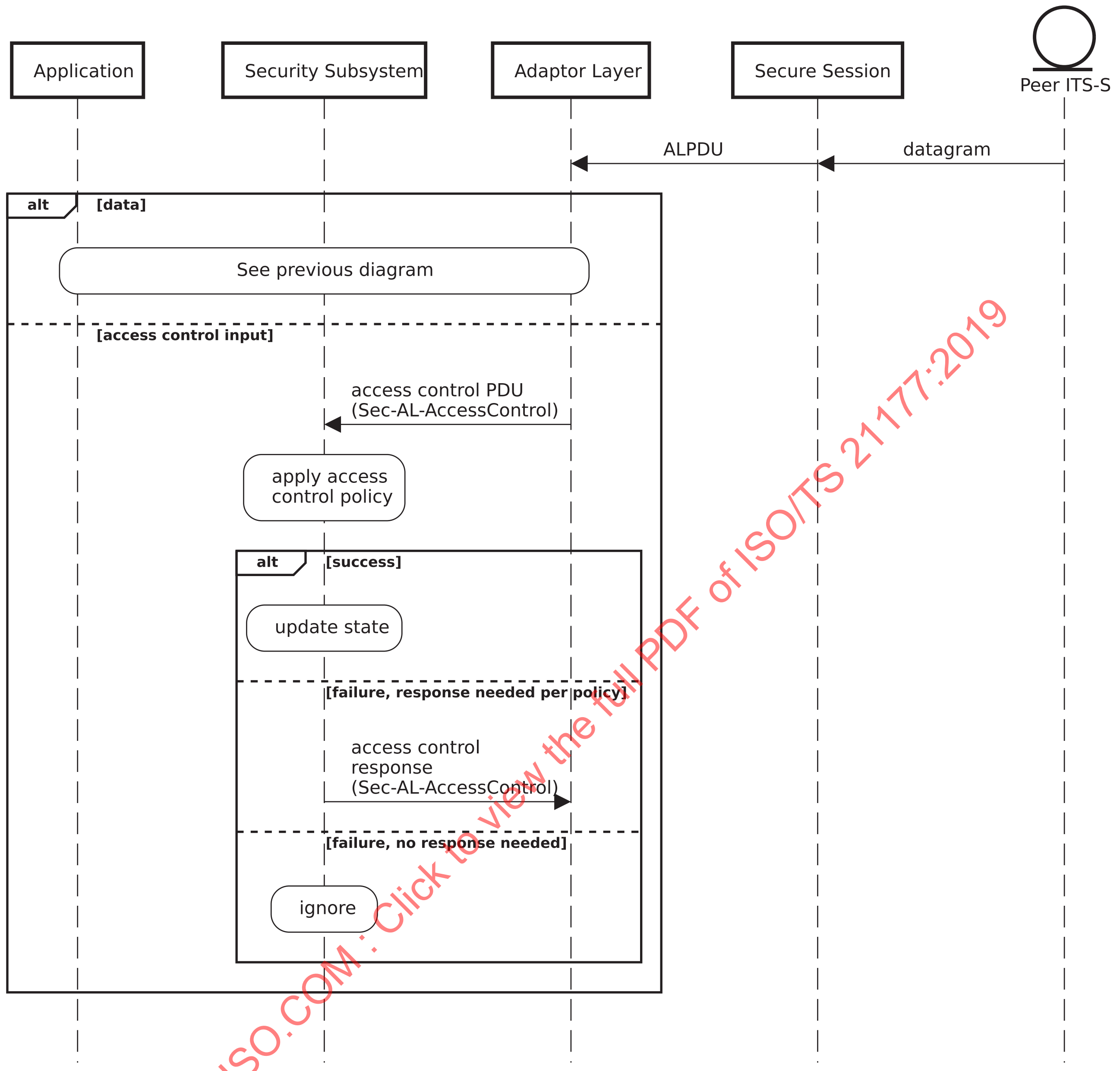


Figure 17 — Sequence diagram for handling of incoming Access Control PDUs

6.9 Secure connection brokering

6.9.1 Goals

"Secure Connection Brokering" is the process by which two Applications, each fronted by a broker, may securely connect via those brokers. The goal is to reduce the information that an eavesdropper can use to learn information about the connection between the “fronted” Applications. Specifically, in a standard TLS 1.3 connection, the server certificate is unencrypted, which provides information to an eavesdropper as to which Application the client is communicating with. In the design in this document, the initial TLS communications are protected by the secure session between the two brokers, and the communication transitions to being directly between the two fronted Applications only once the initial handshake has completed and no further unencrypted information will be sent between the two fronted applications.

The specification in this document supports TLS only, not DTLS.

This process flow supports secure service discovery as described in A.6.

6.9.2 Prerequisites

The prerequisites for secure connection brokering are as follows:

— A Fronted Client application is fronted by a Client Broker application; each may be on different or the same ITS-Ss. There is a secure session between the Fronted Client and Client Broker, per the Secure Session requirements of Clause 9.

— A Fronted Server application is fronted by a Server Broker application; each may be on different or the same ITS-Ss. There is a secure session between the Fronted Server and Server Broker, per the Secure Session requirements of Clause 9.

— The Server Broker and Client Broker are also securely connected according to the Secure Session requirements of Clause 9.

— Before a connection is brokered, each participant has demonstrated appropriate authorisation within the secure session:

  — The Fronted Client has demonstrated to the Client Broker that it is authorised to act as a client for the indicated application;

  — The Client Broker has demonstrated to the Fronted Client and the Server Broker that is it authorised to act as a Client Broker;

  — The Server Broker has demonstrated to the Fronted Server and the Client Broker that is it authorised to act as a Server Broker;

  — The Fronted Server has demonstrated to the Server Broker that it is authorised to act as a server for the indicated application.

The specifics of how this authorization is demonstrated and the specifics of the service discovery process are to be specified separately. Any service discovery process is suitable for use with the mechanisms of this document if, when the service discovery process is completed, the following is true:

— The Fronted Server is registered to the Server Broker as able to receive incoming connections for the indicated applications.

— The Fronted Client is registered to the Client Broker as able to receive notifications that the indicated application is available to connect to.

— A "private connection information" field referred to as the brokerInfo is known to all four parties. The brokerInfo is used to route connections to the Fronted Server within the secure session between the Client Broker and the Server Broker. This document does not place any requirements on the syntax or semantics of the brokerInfo.

— The Fronted Client and the Fronted Server have been provided with the IP addresses and port numbers for the direct connection and have created an internal representation of a socket for that connection.

6.9.3 Overview

An overview of the process, omitting some details and primitive invocations, is shown in Figure 18. The details of the process are given in the next subsection and constitute the normative specification of this process. This and the following subsections follow the convention that "(Secure) Session 1" is the session between the Fronted Client (resp. Fronted Server) and the Client Broker (resp. Server Broker), and "(Secure) Session 2" is the session between the Fronted Client and the Fronted Server.

At the highest level of description:

— There is a setup stage during which the functional entities on each participating device are configured to allow the brokering process.

— The Fronted Client initiates a TLS session by generating a `ClientHello`, which is sent over the existing (Fronted Client <-> Client Broker), (Client Broker <-> Server Broker), and (Server Broker <-> Fronted Server) secure sessions.

— On receipt of the `ClientHello`, the Fronted Server creates a new secure session which creates a `ServerHello` message and returns this to the Fronted Client over the existing (Server Broker <-> Fronted Server), (Client Broker <-> Server Broker), and (Fronted Client <-> Client Broker) secure sessions.

— Once the `ServerHello` has been sent, the Fronted Server transfers the session to a direct socket connection to the Fronted Client.

— Once the `ServerHello` is received, the Fronted Client sends the next TLS handshake message (and the first data, if any) over a direct socket connection to the Fronted Server.

— All following communications, including the conclusion of the TLS handshake, happen over this direct connection between the Fronted Client and the Fronted Server.

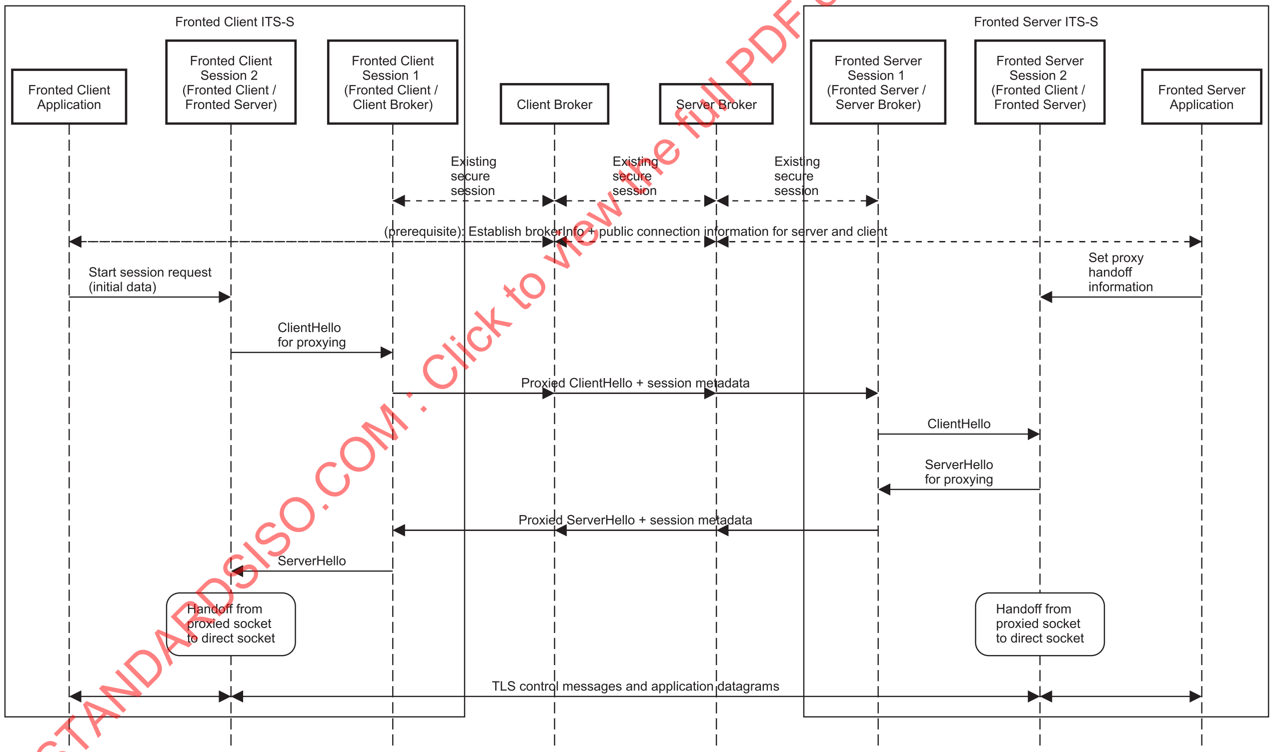

Fronted Client ITS-S
Fronted Server ITS-S
Fronted Client Application
Fronted Client Session 2 (Fronted Client / Fronted Server)
Fronted Client Session 1 (Fronted Client / Client Broker)
Client Broker
Server Broker
Fronted Server Session 1 (Fronted Server / Server Broker)
Fronted Server Session 2 (Fronted Client / Fronted Server)
Fronted Server Application
Existing secure session
Existing secure session
Existing secure session
(prerequisite): Establish brokerInfo + public connection information for server and client
Start session request (initial data)
Set proxy handoff information
ClientHello for proxying
Proxied ClientHello + session metadata
ClientHello
ServerHello for proxying
Proxied ServerHello + session metadata
ServerHello
Handoff from proxied socket to direct socket
Handoff from proxied socket to direct socket
TLS control messages and application datagrams

Figure 18 — Overview sequence diagram for secure connection brokering

6.9.4 Detailed specification

Once the prerequisites have been satisfied, the following setup activities take place.

a) On the Fronted Client ITS-S:

   1) The Fronted Client Application instructs the Security Subsystem to configure a Secure Session instance to send a proxied `ClientHello` using the `App-Sec-Configure.request` service primitive, see 7.6.1, with parameters Proxied = true, BrokerInfo equal to the established brokerInfo, and Proxying Session ID equal to the ID of the connection to the Client Broker.

2) The Security Subsystem instructs the Secure Session instance (Session 2) to send a proxied Client Hello using the `Sec-Sess-Configure.request` primitive, see 9.3.1, with parameters Proxied = true, BrokerInfo equal to the established brokerInfo, and Proxying Session ID equal to the ID of the connection to the Client Broker.

b) On the Client Broker ITS-S:

1) The Client Broker Application uses the `App-AL-EnableProxy.request` service primitive, see 8.3.4, to configure the Adaptor Layer to permit proxying of handshake messages for the indicated connection.

c) On the Server Broker ITS-S:

1) The Server Broker Application uses the `App-AL-EnableProxy.request` service primitive to configure the Adaptor Layer to permit proxying of handshake messages for the indicated connection.

d) On the Fronted Server ITS-S:

1) The Fronted Server Application uses the `App-Sess-EnableProxy.request` service primitive, see 9.2.1, to configure a Secure Session instance to permit proxying of handshake messages for the indicated connection.

2) The Fronted Server Application uses the `App-AL-EnableProxy.request` service primitive, see 8.3.4, to configure the Adaptor Layer to permit proxying of handshake messages for the indicated connection.

Following the setup activities, the process flow is as follows. The specific process flows on the Fronted Client, the Client Proxy, the Server Proxy, and the Fronted Server are given in Figures 19, 20, 21 and 22, respectively.

a) On the Fronted Client:

1) The Fronted Client Session 2:

i) Creates a `ClientHello`.

ii) Since it was configured in setup step a) to use proxying, uses the `AL-Sess-ClientHelloProxy.indication` service primitive, see 9.4.7, to send the `ClientHello` to the Adaptor Layer.

2) The Adaptor Layer:

i) Uses the `ClientHello` and brokerInfo parameters from the received `AL-Sess-ClientHelloProxy.indication` to create an ALPDU of type `TlsClientMsg1`, see 8.2.5.

ii) Uses the `AL-Sess-Data.request` service primitive, see 9.4.1, to pass the ALPDU to Secure Session 1 (the Session ID parameter indicates the session that shall handle the ALPDU).

iii) Configures itself to remember that the presented brokerInfo corresponds to connections between Secure Session 1 and Secure Session 2.

3) The Secure Session 1 sends the ALPDU as one or more datagrams to the peer Secure Session instance on the Client Broker.

b) On the Client Broker:

1) The Secure Session connection to the Fronted Client:

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses the `AL-Sess-Data.indication` service primitive, see 9.4.3, to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via the `AL-Sess-Data.indication` primitive.

ii) Determines that the ALPDU is of type `TlsClientMsg1`, see 8.2.5.

iii) Determines that it has been configured to proxy `ClientHello` messages with the brokerInfo field of the ALPDU, as done in setup step b), and that this configuration has not timed out or completed.

iv) Determines from the information provided in setup step b) which Secure Session instance to provide the ALPDU to.

v) Uses the `AL-Sess-Data.request` primitive, see 9.4.1, to provide the ALPDU to the Secure Session with the Server Broker.

3) The Secure Session connection to the Server Broker sends the ALPDU as one or more datagrams to the peer Secure Session instance on the Server Broker.

c) On the Server Broker:

1) The Secure Session connection to the Client Broker:

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses the `AL-Sess-Data.indication` primitive, see 9.4.3, to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via `AL-Sess-Data.indication`.

ii) Determines that the ALPDU is of type `TlsClientMsg1`, see 8.2.5.

iii) Determines that it has been configured to proxy `ClientHello` messages with the brokerInfo field of the ALPDU, as done in setup step c), and that this configuration has not timed out or completed.

iv) Determines from the information provided in setup step c) which Secure Session instance to provide the ALPDU to.

v) Uses the `AL-Sess-Data.request` primitive to provide the ALPDU to the Secure Session with the Fronted Server.

3) The Secure Session connection to the Fronted Server sends the ALPDU as one or more datagrams to the peer Secure Session instance on the Fronted Server.

d) On the Fronted Server:

1) The Secure Session connection to the Server Broker (Secure Session 1):

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses `AL-Sess-Data.indication` to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via `AL-Sess-Data.indication`

ii) Determines that the ALPDU is of type `TlsClientMsg1`.

iii) Determines that it has been configured to accept proxied `ClientHello` messages with the brokerInfo field of the ALPDU, as done in setup step d), and that this configuration has not timed out or completed.

iv) Uses the `AL-Sess-ClientHelloProxy.request` service primitive, see 9.4.6, to provide the `ClientHello` and brokerInfo to the Secure Session services.

3) Secure Session 2:

i) Receives the Client Hello and metadata via `AL-Sess-ClientHelloProxy.request`.

ii) Creates a secure session instance and secure session ID.

iii) Generates the TLS response message (`ServerHello` and other server handshake messages) and forms a single datagram containing the messages.

iv) Uses the `AL-Sess-ServerHelloProxy.indication` service primitive, see 9.4.9, to provide the `ServerHello` and metadata to the Adaptor Layer.

v) Configures itself to accept incoming connections from the server socket provided by the `App-Sess-EnableProxy.request` service primitive in setup step d) 1) and transfers the TLS state to that connection.

4) The Adaptor Layer:

i) Uses the `ServerHello` datagram and the brokerInfo from the received `AL-Sess-ServerHelloProxy.indication` service primitive to create an ALPDU of type `TlsServerMsg1`.

ii) Determines from the information provided in setup step d) which Secure Session instance to provide the ALPDU to.

iii) Uses the `AL-Sess-Data.request` service primitive to provide the ALPDU Secure Session 1.

iv) Disables proxying with respect to that brokerInfo.

5) The Secure Session 1 sends the ALPDU as one or more datagrams to the peer Secure Session instance on the Server Broker.

e) On the Server Broker:

1) The Secure Session connection to the Fronted Server:

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses `AL-Sess-Data.indication` to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via the `AL-Sess-Data.indication` primitive.

ii) Determines that the ALPDU is of type `TlsServerMsg1`.

iii) Determines that it has been configured to proxy `ServerHello` messages with the brokerInfo field of the ALPDU, as done in setup step c), and that this configuration has not timed out or completed.

iv) Determines from the information provided in setup step c) which Secure Session instance to provide the ALPDU to.

v) Uses the `AL-Sess-Data.request` primitive, see 9.4.1, to provide the ALPDU to the Secure Session with the Server Broker.

vi) Disables proxying with respect to that brokerInfo.

3) The Secure Session connection to the Client Broker sends the ALPDU as one or more datagrams to the peer Secure Session instance on the Client Broker.

f) On the Client Broker:

1) The Secure Session connection to the Server Broker:

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses the `AL-Sess-Data.indication` service primitive, see 9.4.3, to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via the `AL-Sess-Data.indication` service primitive.

ii) Determines that the ALPDU is of type `TlsServeMsg1`, see 8.2.6.

iii) Determines that it has been configured to proxy `ServerHello` messages with the brokerInfo field of the ALPDU, as done in setup step b), and that this configuration has not timed out or completed.

iv) Determines from the information provided in setup step b) which Secure Session instance to provide the ALPDU to.

v) Uses the `AL-Sess-Data.request` service primitive, see 9.4.1, to provide the ALPDU to the Secure Session with the Client Broker.

vi) Disables proxying with respect to that brokerInfo.

3) The Secure Session connection to the Fronted Client sends the ALPDU as one or more datagrams to the peer Secure Session instance (Secure Session 1) on the Fronted Client.

g) On the Fronted Client:

1) The Secure Session connection to the Client Broker (Secure Session 1):

i) Receives the datagrams and decrypts and reconstructs the ALPDU.

ii) Uses the `AL-Sess-Data.indication` service primitive, see 9.4.3, to pass the ALPDU to the Adaptor Layer.

2) The Adaptor Layer:

i) Receives the ALPDU via the `AL-Sess-Data.indication` service primitive.

ii) Determines that the ALPDU is of type `TlsServeMsg1`, see 8.2.6.

iii) Per its self-configuration in step a) 2) iii), determines that the `ServerHello` message and metadata should be provided to Secure Session 2.

iv) Uses the `AL-Sess-ClientHelloProxy.request` primitive to provide the `ServerHello` message and metadata to Secure Session 2.

v) Disables proxying with respect to that brokerInfo.

3) Secure Session 2:

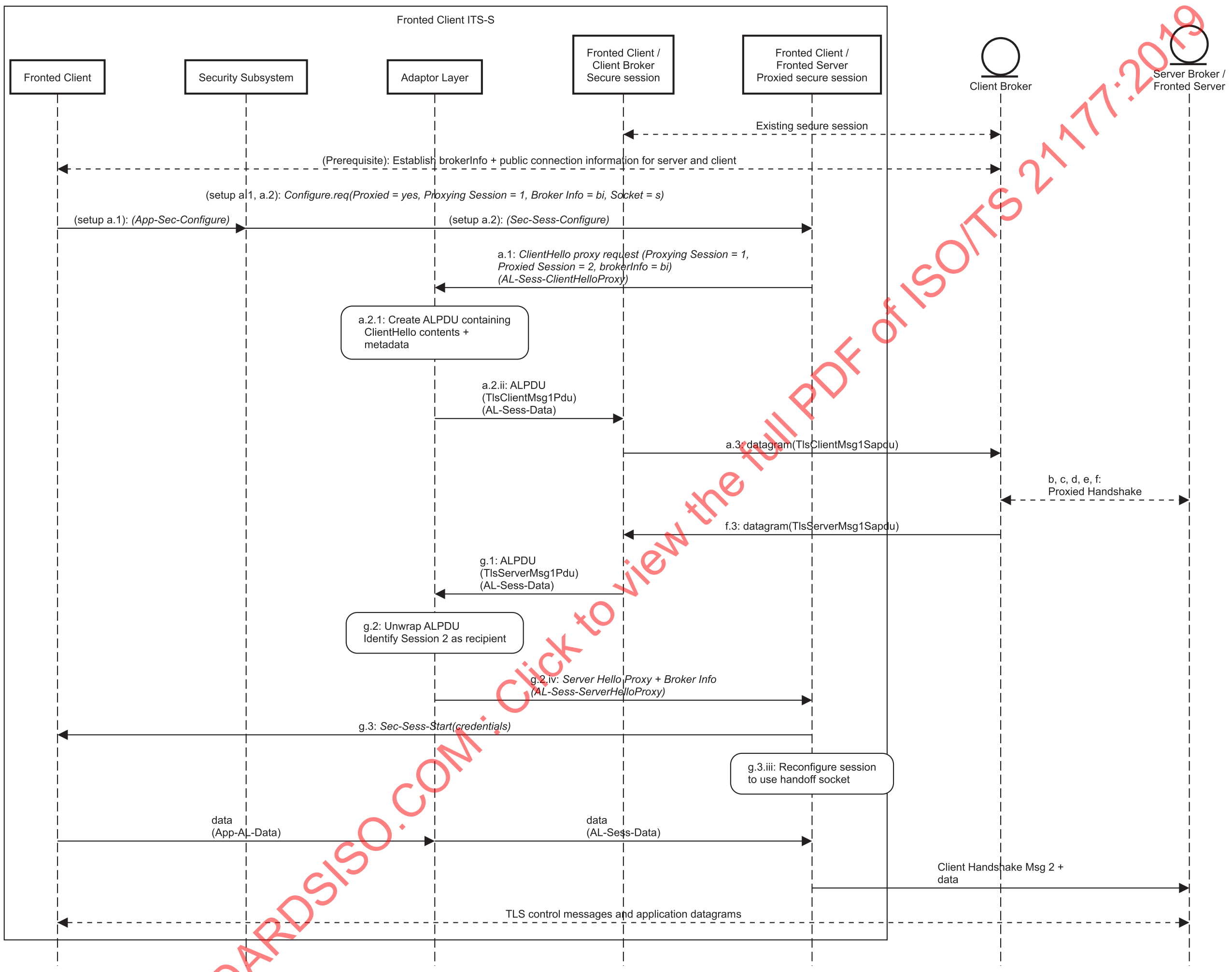
i) Receives the `Server Hello` and metadata via the `AL-Sess-ClientHelloProxy.request` service primitive, see 9.4.6.

ii) Generates the TLS response message.

iii) Reconfigures itself to use the communications socket provided in setup step a) (which provides a direct connection to the Fronted Server), transfers the TLS state to that connection, and sends the TLS response message via that connection.

iv) Uses the `Sec-Sess-Start.indication` primitive, see 9.3.3, to indicate that the session has been established.

At this point the secure session has been directly established between the Fronted Client and the Fronted Server and any additional activities within the session are covered by other process flows in this document.



Fronted Client ITS-S
Fronted Client
Security Subsystem
Adaptor Layer
Fronted Client / Client Broker Secure session
Fronted Client / Fronted Server Proxied secure session
Client Broker
Server Broker / Fronted Server
Existing secure session
(Prerequisite): Establish brokerInfo + public connection information for server and client
(setup a1, a.2): Configure.req(Proxied = yes, Proxying Session = 1, Broker Info = bi, Socket = s)
(setup a.1): (App-Sec-Configure)
(setup a.2): (Sec-Sess-Configure)
a.1: ClientHello proxy request (Proxying Session = 1, Proxied Session = 2, brokerInfo = bi) (AL-Sess-ClientHelloProxy)
a.2.1: Create ALPDU containing ClientHello contents + metadata
a.2.ii: ALPDU (TlsClientMsg1Pdu) (AL-Sess-Data)
a.3: datagram(TlsClientMsg1Sapdu)
b, c, d, e, f: Proxied Handshake
f.3: datagram(TlsServerMsg1Sapdu)
g.1: ALPDU (TlsServerMsg1Pdu) (AL-Sess-Data)
g.2: Unwrap ALPDU Identify Session 2 as recipient
g.2.iv: Server Hello Proxy + Broker Info (AL-Sess-ServerHelloProxy)
g.3: Sec-Sess-Start(credentials)
g.3.iii: Reconfigure session to use handoff socket
data (App-AL-Data)
data (AL-Sess-Data)
Client Handshake Msg 2 + data
TLS control messages and application datagrams

Figure 19 — Sequence diagram for secure connection brokering operations on the Fronted Client

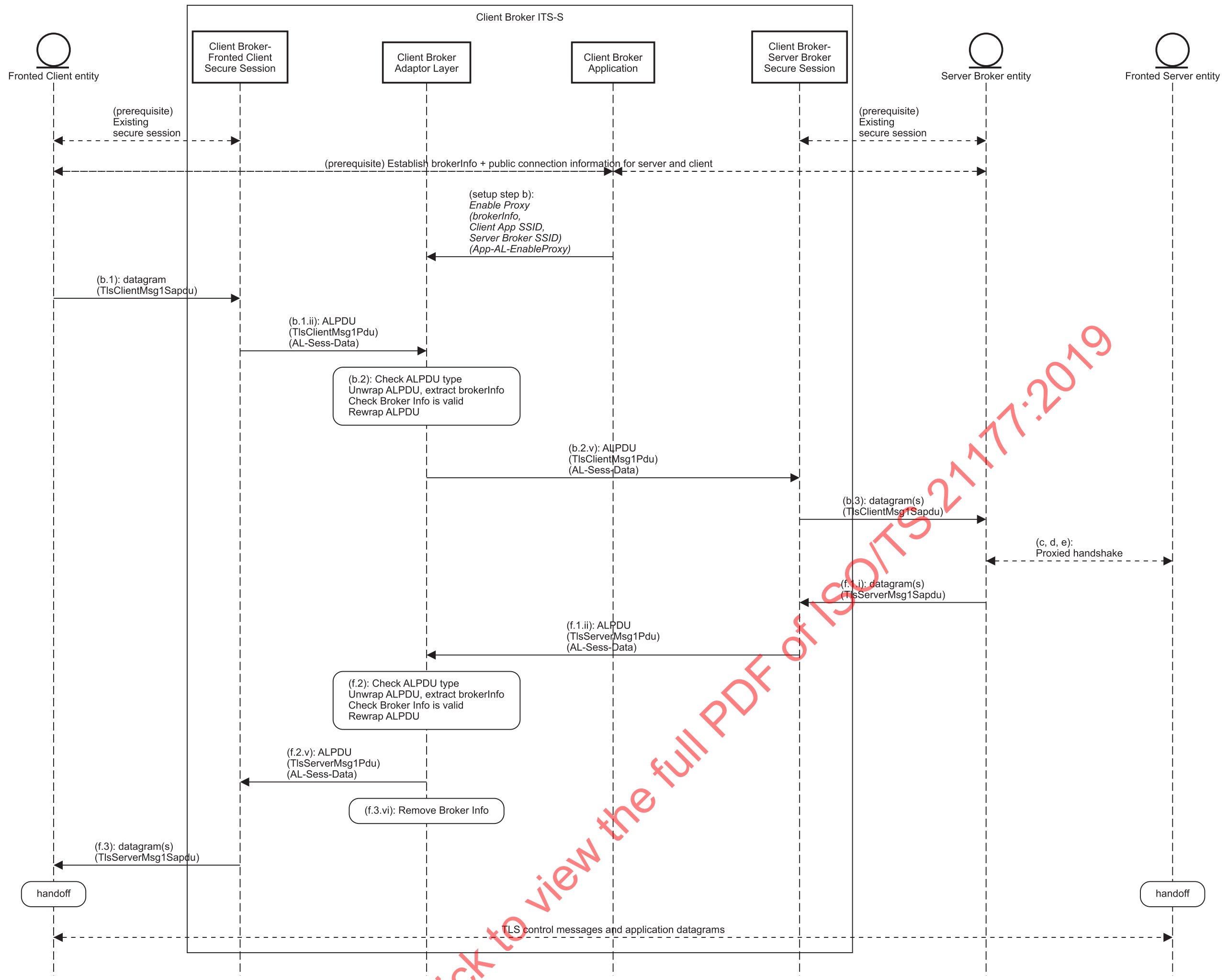

Client Broker ITS-S
Fronted Client entity
Client Broker-Fronted Client Secure Session
Client Broker Adaptor Layer
Client Broker Application
Client Broker-Server Broker Secure Session
Server Broker entity
Fronted Server entity
(prerequisite) Existing secure session
(prerequisite) Existing secure session
(prerequisite) Establish brokerInfo + public connection information for server and client
(setup step b): Enable Proxy (brokerInfo, Client App SSID, Server Broker SSID) (App-AL-EnableProxy)
(b.1): datagram (TlsClientMsg1Sapdu)
(b.1.ii): ALPDU (TlsClientMsg1Pdu) (AL-Sess-Data)
(b.2): Check ALPDU type Unwrap ALPDU, extract brokerInfo Check Broker Info is valid Rewrap ALPDU
(b.2.v): ALPDU (TlsClientMsg1Pdu) (AL-Sess-Data)
(b.3): datagram(s) (TlsClientMsg1Sapdu)
(c, d, e): Proxied handshake
(f.1.i): datagram(s) (TlsServerMsg1Sapdu)
(f.1.ii): ALPDU (TlsServerMsg1Pdu) (AL-Sess-Data)
(f.2): Check ALPDU type Unwrap ALPDU, extract brokerInfo Check Broker Info is valid Rewrap ALPDU
(f.2.v): ALPDU (TlsServerMsg1Pdu) (AL-Sess-Data)
(f.3.vi): Remove Broker Info
(f.3): datagram(s) (TlsServerMsg1Sapdu)
handoff
handoff
TLS control messages and application datagrams

Figure 20 — Sequence diagram for secure connection brokering operations on the Client Broker

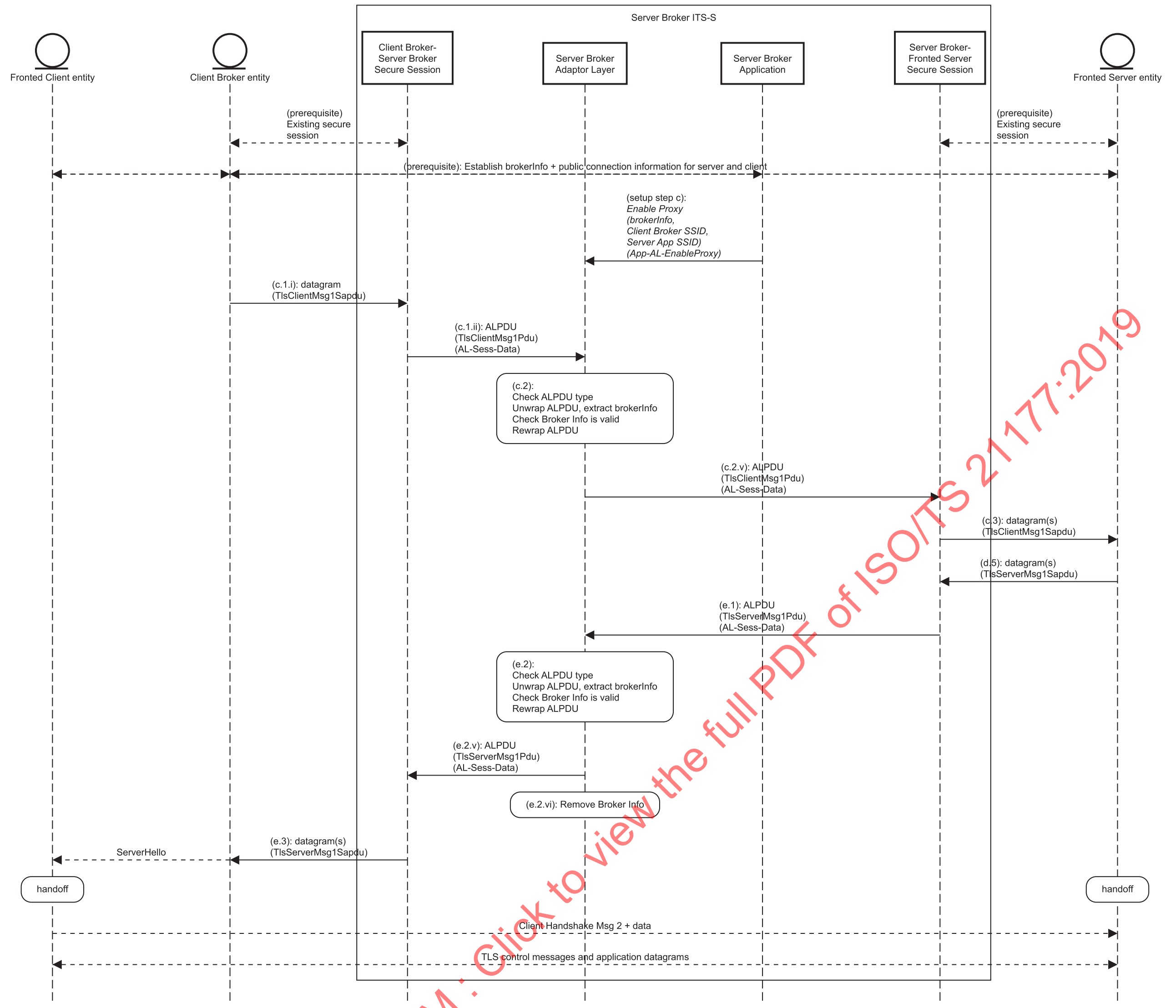

Server Broker ITS-S
Fronted Client entity
Client Broker entity
Client Broker-Server Broker Secure Session
Server Broker Adaptor Layer
Server Broker Application
Server Broker-Fronted Server Secure Session
Fronted Server entity
(prerequisite) Existing secure session
(prerequisite): Establish brokerInfo + public connection information for server and client
(prerequisite) Existing secure session
(setup step c): Enable Proxy (brokerInfo, Client Broker SSID, Server App SSID) (App-AL-EnableProxy)
(c.1.i): datagram (TlsClientMsg1Sapdu)
(c.1.ii): ALPDU (TlsClientMsg1Pdu) (AL-Sess-Data)
(c.2): Check ALPDU type Unwrap ALPDU, extract brokerInfo Check Broker Info is valid Rewrap ALPDU
(c.2.v): ALPDU (TlsClientMsg1Pdu) (AL-Sess-Data)
(c.3): datagram(s) (TlsClientMsg1Sapdu)
(d.5): datagram(s) (TlsServerMsg1Sapdu)
(e.1): ALPDU (TlsServerMsg1Pdu) (AL-Sess-Data)
(e.2): Check ALPDU type Unwrap ALPDU, extract brokerInfo Check Broker Info is valid Rewrap ALPDU
(e.2.v): ALPDU (TlsServerMsg1Pdu) (AL-Sess-Data)
(e.2.vi): Remove Broker Info
ServerHello
(e.3): datagram(s) (TlsServerMsg1Sapdu)
handoff
handoff
Client Handshake Msg 2 + data
TLS control messages and application datagrams

Figure 21 — Sequence diagram for secure connection brokering operations on the Server Broker

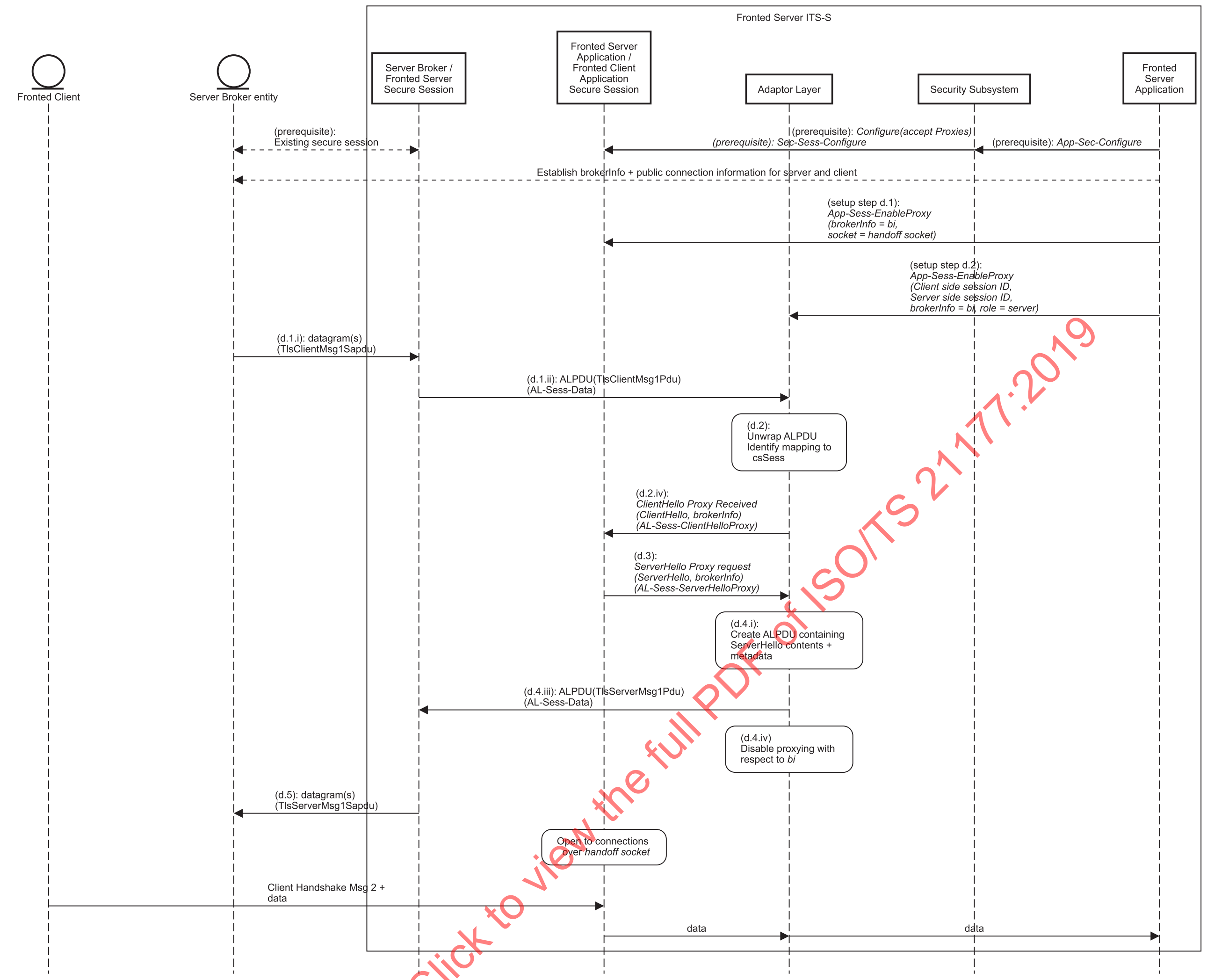

Fronted Server ITS-S
Fronted Client
Server Broker entity
Server Broker / Fronted Server Secure Session
Fronted Server Application / Fronted Client Application Secure Session
Adaptor Layer
Security Subsystem
Fronted Server Application
(prerequisite): Existing secure session
(prerequisite): Configure(accept Proxies)
(prerequisite): Sec-Sess-Configure
(prerequisite): App-Sec-Configure
Establish brokerInfo + public connection information for server and client
(setup step d.1): App-Sess-EnableProxy (brokerInfo = bi, socket = handoff socket)
(setup step d.2): App-Sess-EnableProxy (Client side session ID, Server side session ID, brokerInfo = bi, role = server)
(d.1.i): datagram(s) (TlsClientMsg1Sapdu)
(d.1.ii): ALPDU(TlsClientMsg1Pdu) (AL-Sess-Data)
(d.2): Unwrap ALPDU Identify mapping to csSess
(d.2.iv): ClientHello Proxy Received (ClientHello, brokerInfo) (AL-Sess-ClientHelloProxy)
(d.3): ServerHello Proxy request (ServerHello, brokerInfo) (AL-Sess-ServerHelloProxy)
(d.4.i): Create ALPDU containing ServerHello contents + metadata
(d.4.iii): ALPDU(TlsServerMsg1Pdu) (AL-Sess-Data)
(d.4.iv) Disable proxying with respect to bi
(d.5): datagram(s) (TlsServerMsg1Sapdu)
Open to connections over handoff socket
Client Handshake Msg 2 + data
data
data

Figure 22 — Sequence diagram for secure connection brokering operations on the Fronted Server

6.10 Force end session

In this process flow, the Application or the security subsystem instructs the Secure Session services to force end the current session.

Prerequisites:

— The Secure Session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as defined in 5.2, there has been an initialize step using the methods of 6.5.

The process steps for this process flow are:

a) EITHER of the following happens:

1) The Application determines that the session should be ended:

i) The Application notifies the Security Subsystem using the `App-Sec-EndSession.request` primitive, see 7.6.8.

ii) The Security Subsystem:

I) ACKs the request via `App-Sec-EndSession.confirm` primitive, see 7.6.9.

II) notifies the Adaptor Layer using the `Sec-AL-EndSession.request` primitive, see 8.4.4.

2) The Security Subsystem determines that the session should be ended:

i) The Security Subsystem notifies the Application using the `App-Sec-EndSession.indication` primitive, see 7.6.10.

ii) The Security Subsystem notifies the Adaptor Layer using the `Sec-AL-EndSession.request` primitive, see 8.4.4.

b) The Adaptor Layer:

1) ACKs the request using the `Sec-AL-EndSession.confirm` primitive, see 8.4.5.

2) Notifies the Secure Session services that the session should be terminated via the `AL-Sess-EndSession.request` primitive, see 9.4.4.

c) The Secure Session service:

1) ACKs the request using the `AL-Sess-EndSession.confirm` primitive, see 9.4.5.

2) If session termination for the specific secure session mechanism requires peer datagram exchange, the peer secure session entities carry out the session termination process.

A sequence diagram for this process flow is given in Figure 23.

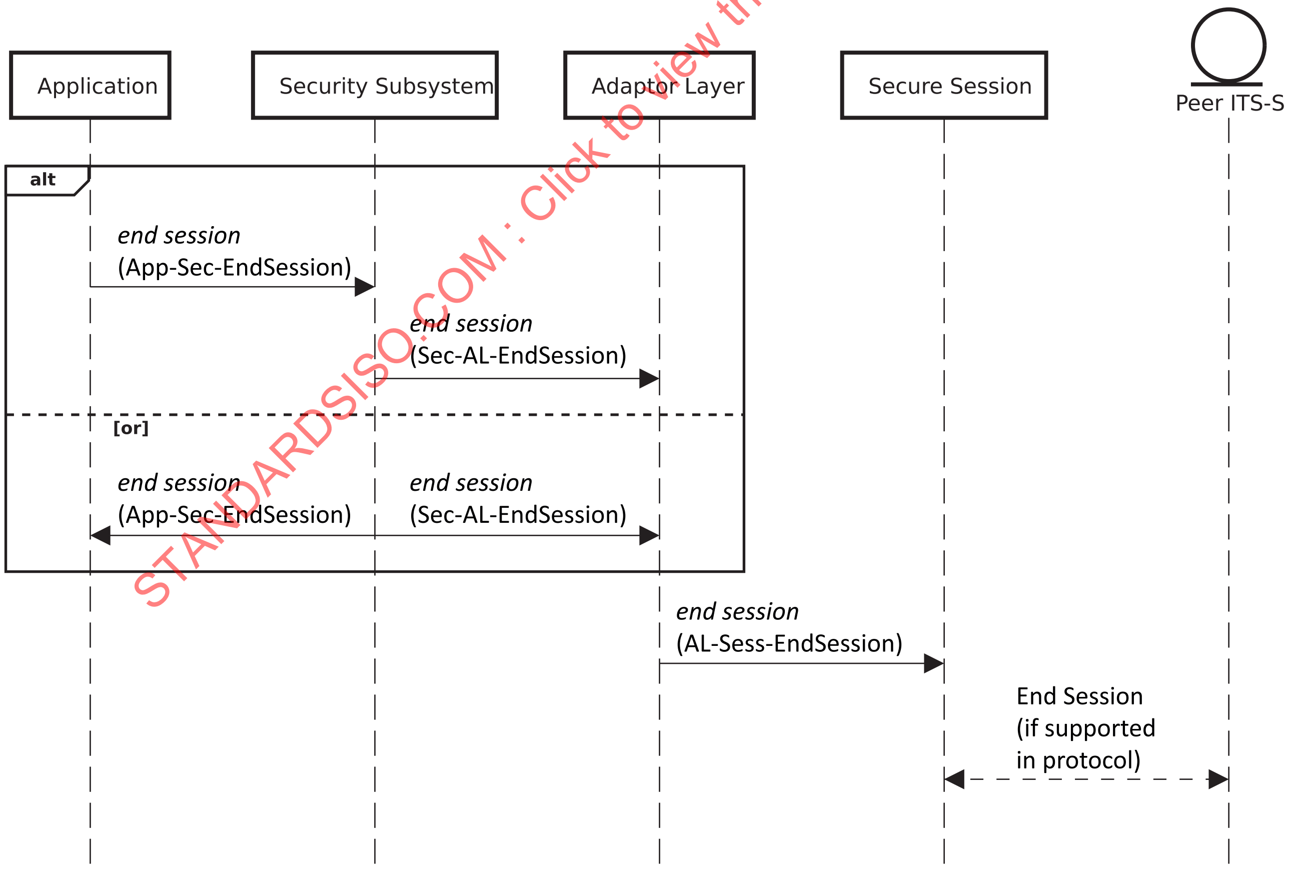


Figure 23 — Sequence diagram for Force End Session

6.11 Session terminated at session layer

In this process flow, the peer trusted entity terminates the session at the session layer, connectivity to the peer is lost, or the secure session times out. The Secure Session services become aware that the session has terminated and inform the higher layers.

Prerequisites:

— The Secure Session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as defined in 5.2, there has been an initialize step using the methods of 6.4.

The process steps for this process flow are:

a) The Secure Session service informs the Security Subsystem that the session has terminated using the `Sec-Sess-EndSession.indication` primitive, see 9.3.4.

b) The Security Subsystem:

1) Notifies the Application using the `App-Sec-EndSession.indication` primitive, see 7.6.10.

2) Notifies the Adaptor Layer using the `Sec-AL-EndSession.request` primitive, see 8.4.4.

c) The Adaptor Layer ACKs the request using the `Sec-AL-EndSession.confirm` primitive, see 8.4.5.

A sequence diagram for this process flow is given in Figure 24.

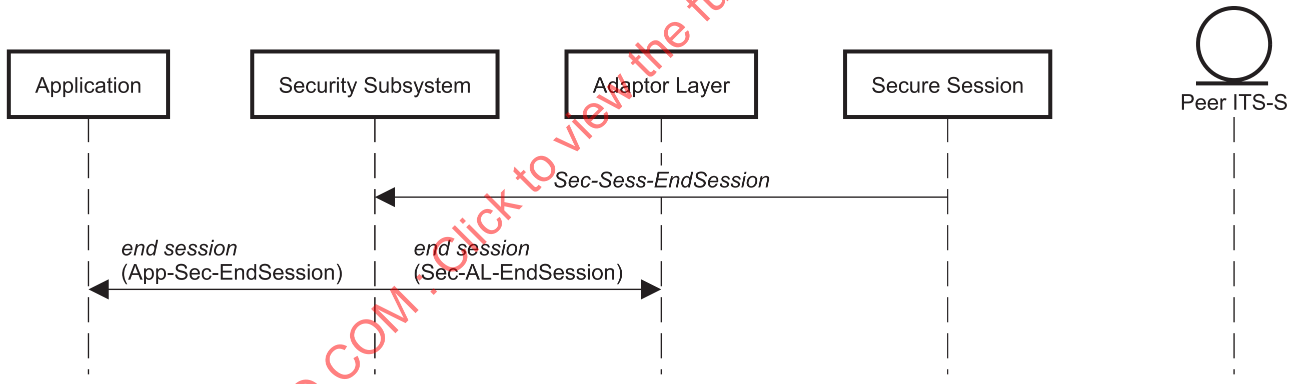


Figure 24 — Sequence diagram for Session Terminated at Session Layer

6.12 Deactivate

In this process flow the Secure Session services are instructed to deactivate, i.e. to stop using the credentials that were proved in the corresponding Configure process flow and (if in a server role) to stop accepting incoming connections. This process flow has no effect on existing sessions.

Prerequisites:

The Secure Session service has been configured using the methods of 6.4.

The process steps for this process flow are:

a) EITHER of the following happens:

1) The Application determines that the Secure Session Services should be deactivated:

i) The Application notifies the Security Subsystem using the `App-Sec-Deactivate.request` primitive, see 7.6.11).

ii) The Security Subsystem:

I) ACKs the request via `App-Sec-Deactivate.confirm` primitive, see 7.6.12.

II) notifies the Secure Session services using the `Sec-Sess-Deactivate.request` primitive, see 9.3.5.

2) The Security Subsystem determines that the session should be ended:

i) The Security Subsystem notifies the Application using the `App-Sec-Deactivate.indication` primitive, see 7.6.13.

ii) The Security Subsystem notifies the Secure Session services using the `Sec-Sess-Deactivate.request` primitive, see 9.3.5.

b) The Secure Session service:

1) ACKs the request using the `Sec-Sess-Deactivate.confirm` primitive, see 9.3.6.

2) Stop accepting new incoming connections (if in a server role) or attempting to start new outgoing connections (if in a client role).

3) Deletes all state relevant to new sessions (while maintaining state relevant to existing sessions).

NOTE The secure session instance does not delete the Cryptomaterial Handle contents, just the local reference to the Cryptomaterial Handle held by the secure session.

A sequence diagram for this process flow is given in Figure 25.

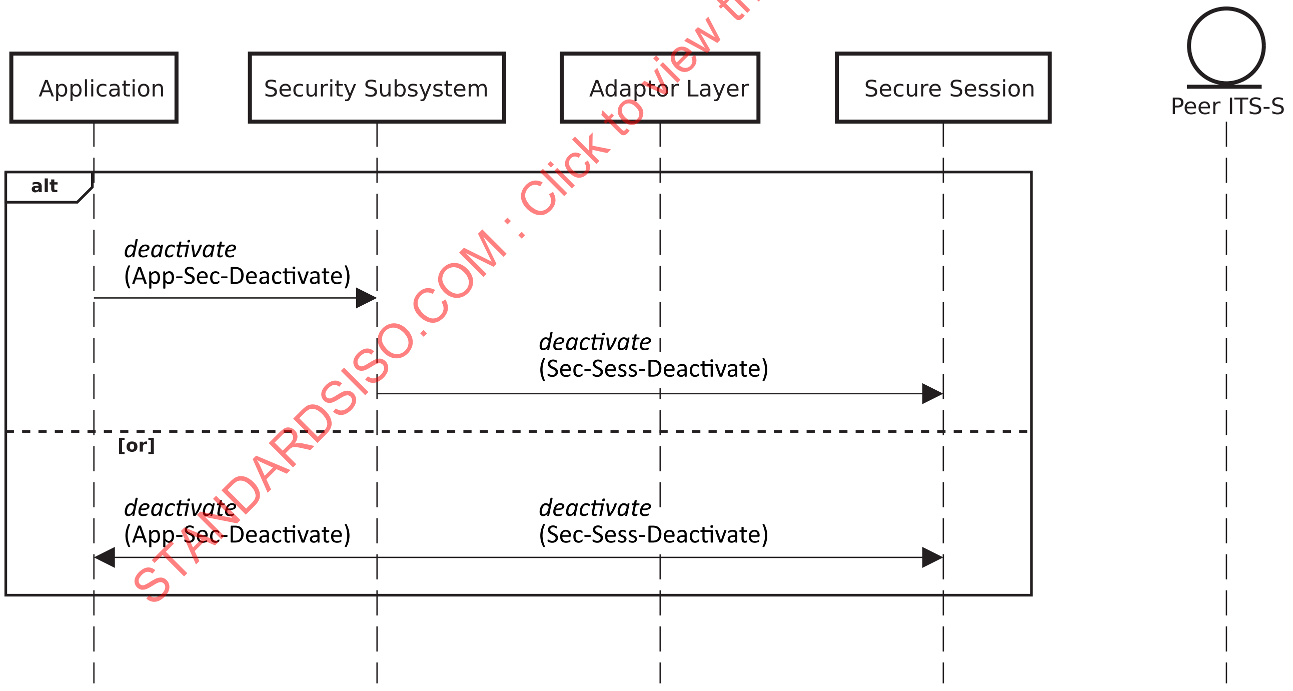


Figure 25 — Sequence diagram for Deactivate

6.13 Secure session example

This subclause illustrates how the process flows from the previous section may be put together to full execution of an entire secure session instance. This illustration does not show connection brokering.

The session runs between two ITS Application Processes referred to as the Initiator and the Responder. Optional data flows are in italics in the figure.

— First, both the Initiator and the Responder application processes configure the security subsystem and the secure session services as specified in 6.4.

— The Initiator application process sends data to its Adaptor layer. This prompts the Initiator secure session services to connect to the Responder secure session services, resulting in the Start Session process flow as specified in 6.5. In this flow the credentials exchanged in the handshake are provided to the receiving Security Subsystem. The credential exchange is shown as optional in the figure because the credentials are only exchanged if the session is a cryptographic secure session, not if it is a physical secure session.

— Following the Start Session process flow, the Responder secure session services provide the data to the Responder application process via the Responder adaptor layer as specified in 6.8.

— The Initiator and Responder send and receive data. Sending data is specified in 6.6 and receiving data is specified in 6.8.

— During the exchange, the Initiator or Responder may determine that additional access control information is needed. Additional access control requests and responses are sent via the Send Access Control PDU, process flow see 6.7 Receiving access control PDUs is specified in 6.8. In the figure, the Responder is shown as requesting the access control PDUs, but in practice either or both of the Initiator and the Responder may send a request.

— One party — in this illustration, the Initiator — force ends the session, as specified in 6.10. As a consequence, the Responder secure session instance informs the other functional entities associated with it that the session has been terminated at the session layer, as specified in 6.11.

— Both parties deactivate the secure session services as specified in 6.12.

Figure 26 shows all the information flows associated with this illustration.

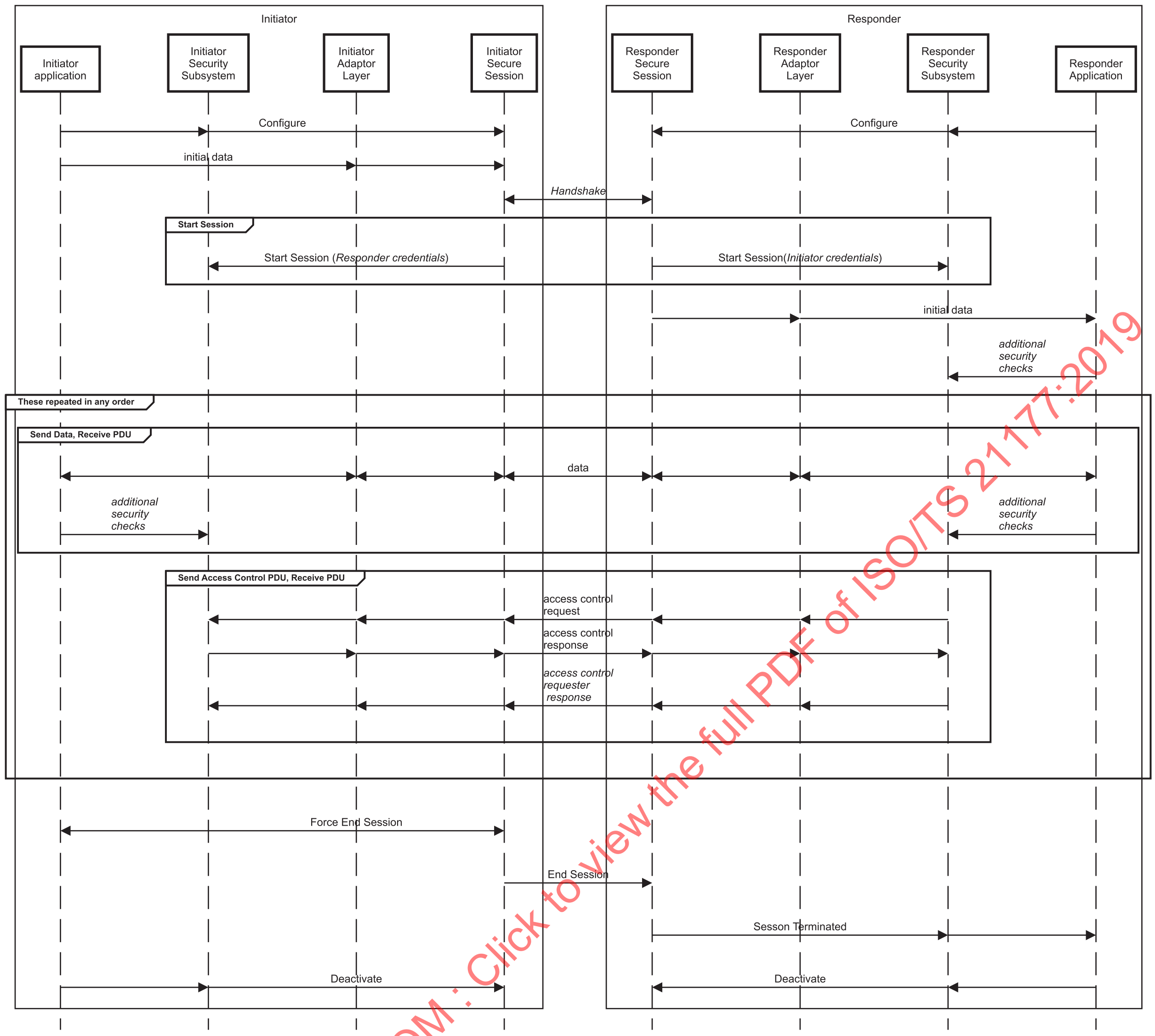


Figure 26 — Sequence diagram for complete secure session example

7 Security Subsystem: interfaces and data types

7.1 General

The Security Subsystem:

— Implements access control processing on incoming data when so requested using `App-Sec-Incoming.request`, see 7.6.6.

— Signs outgoing APDUs if requested by the Application using `App-Sec-Data.request`, see 7.6.4.

— Verifies incoming APDUs if requested by the Application using `App-Sec-Incoming.request`, see 7.6.6.

— Generates requests for, and responses to requests for, additional access control information.

— Updates its state in response to received Access Control PDUs.

— Participates in configuration activities associated with:

  — Configure, see 6.4,

— Start Session, see 6.5,

— Force end session, see 6.10,

— Session terminated at session layer, see 6.11, and

— Deactivate, see 6.12.

The Application uses the Security Subsystem to determine whether an APDU is valid for use. An Application that is conformant to this specification shall invoke `App-Sec-Incoming.request`, see 7.6.6, on every incoming APDU before “making use” of it, i.e. before using that APDU to alter the state of the Application or a resource and before generating a response APDU. An Application that is conformant to this specification may use Application logic to “reject” an incoming APDU without invoking `App-Sec-Incoming.request` on that APDU, i.e. it may use Application logic to determine not to alter the state of an Application or resource or not to generate a response APDU.

7.2 Access control policy and state

The Security Subsystem determines whether an APDU passed to it via `App-Sec-Incoming.request`, see 7.6.6, is valid, i.e. whether the access control policy permits the Application to make use of the APDU. In order to make this determination, it uses the following inputs:

— Its internal state, i.e. information that it has learned about the peer trusted entity during communications with the peer trusted entity.

NOTE If the peer trusted entity uses the same certificate across multiple sessions, the home Security Subsystem can use information learned in previous sessions in making access control decisions, if permitted by the access control policy.

— The state of the home ITS-SU: for example, for an ITS-SU in a vehicle, the access control policy may state that access to certain resources is only permitted if the vehicle is not moving.

— The APDU itself.

— The current state of the application, based on APDUs sent and received and on the configuration of the application.

This document does not provide interfaces for the Security Subsystem to learn about the state of the home ITS-SU: the assumption is that if an ITS-SU supports access control policies that involve the use of the properties of the ITS-SU, then the implementation of the Security Subsystem will have access to those properties.

If the access control policy requires it, the Security Subsystem can generate Access Control PDUs to request additional information from the peer trusted entity and use the responses to update its sate. The Security Subsystem can also generate responses to requests received as Access Control PDUs from the peer Security Subsystem. The two forms of additional information supported in this document are Enhanced Authentication, specified in 7.3, and Extended Authentication, specified in 7.4.

This document does not provide a specification for the access control policy language. An access control policy may specify many different conditions for controlling access, such as:

— The PSID and (optionally) SSP that must appear in the received certificate for the Secure Session handshake.

— Whether enhanced or extended authentication is required for access to specific resources or for particular activities.

— Whether there are particular conditions on the state of the home ITS-SU that must be fulfilled to carry out particular activities.

Since the Security Subsystem applies the access control policy directly to APDUs, the Security Subsystem associated with a particular Application must be able to interpret APDUs. The Security Subsystem implementation for a particular Application will therefore in general be Application-specific. This document specifies an internal Security Subsystem interface consisting of the `Sec-AuthState.request` and `Sec-AuthState.confirm` primitives to provide information about authorization state from the Authorization State Subsystem to the Access Control Subsystem. The Authorization State Subsystem, and this interface, are not application-specific.

7.3 Enhanced authentication

7.3.1 Definition and possible states

This subclause specifies *enhanced authentication*. In enhanced authentication, one party acts as the owner and the other party acts as the accessor. The owner stores or obtains a secret value, referred to as the *enhanced authentication secret*. Information derived from the enhanced authentication secret value is provided to the accessor. The means for providing this information are part of the specification of an application that uses the mechanisms given in this document; possible examples are given below to help understanding. The owner then requests that the accessor proves knowledge of the secret by sending an Enhanced Authentication Request Access Control PDU over the secure session, and the accessor proves knowledge of the secret by sending an Enhanced Authentication Response Access Control PDU to the owner.

Examples of enhanced authentication include:

— The accessed device displays a PIN which is entered into the accessing device.

— The owner comes up with a PIN and enters it into both the accessed and the accessing device.

— The accessed device has a secret which is also stored on a token such as a *keyfob* associated to the accessed device. The owner taps the *keyfob* against the accessing device to transfer either the secret itself or a value derived from it.

— The accessed device has a seed for generating one-time passwords (OTPs). The owner has an app that generates the same set of one-time passwords. The owner enters the current OTP into the accessing device.

This document supports multiple mechanisms for Enhanced Authentication. In each mechanism, the Request and Response both include values derived from the secret, rather than the secret itself; the value in the Request is referred to as the identifier and the value in the Response is referred to as the verifier. The specific mechanisms are specified in 7.3.6.

Enhanced authentication is provided interactively: one party to the communication requests the enhanced authentication and the other provides it. Enhanced authentication may be requested by either party to the communication, whether that party played the role of the client or the server in the original TLS handshake.

7.3.2 States for owner role enhanced authentication

For any individual secure session and any enhanced authentication secret, the *owner role enhanced authentication state* with respect to that secret may be one of the following:

— *Unestablished* — no request has been sent.

— *Pending* — a request has been sent but no valid response has been received.

— *Established* — a valid response has been received.

In this subclause the state for a (application, session, secret) tuple is denoted by O(i,j,k).

O(i,j,k) for a particular session identifier j is set to Unestablished when the session is created (by `App-Sec-Configure.request` if the session role is client, and by `Sec-Sess-StartSession.indication` if the session role is server). When the owner role enhanced authentication state is *Unestablished*, incoming Enhanced Authentication Response Access Control PDUs with respect to that secret are ignored.

O(i,j,k) transitions from *Unestablished to Pending* when the Security Subsystem sends an Enhanced Authentication Request Access Control PDU with respect to k. This may happen when prompted by the application, by the access control policy, by a timer, in response to an incoming APDU, or by other events or triggers to be specified in the specification of an application that uses the mechanisms provided by this document.

When O(i,j,k) is *Pending*, the Security Subsystem may send additional Enhanced Authentication Request Access Control PDUs with respect to that enhanced authentication secret. As with the initial Enhanced Authentication Request Access Control PDU, the trigger for sending additional Enhanced Authentication Request Access Control PDUs is to be specified in the specification of an application that uses the mechanisms provided by this document.

O(i,j,k) transitions from *Pending* to *Unestablished* on the end of session *j*, on access control policy-defined timeout, or in response to other events or triggers are to be specified in the specification of an application that uses the mechanisms provided by this document.

O(i,j,k) transitions from *Pending* to *Established* when the Security System receives a valid Enhanced Authentication Response Access Control PDU corresponding to an Enhanced Authentication Request Access Control PDU that was sent over session *j* with respect to enhanced authentication secret *k* and that has not timed out. The definition of a valid response depends on the enhanced authentication mechanism in use and is given in the individual enhanced authentication mechanism section, see 7.3.6.

When O(i,j,k) is *Established*, the Security Subsystem shall not send an Enhanced Authentication Request for *k*. Incoming Enhanced Authentication Response Access Control PDU with respect to *k* are ignored.

The owner role enhanced authentication state transitions from *Established* to *Unestablished* on the end of session *j*, on access control policy-defined timeout, or in response to other events or triggers are to be specified in the specification of an application that uses the mechanisms provided by this document.

A state machine diagram for enhanced authentication is provided in Figure 27.

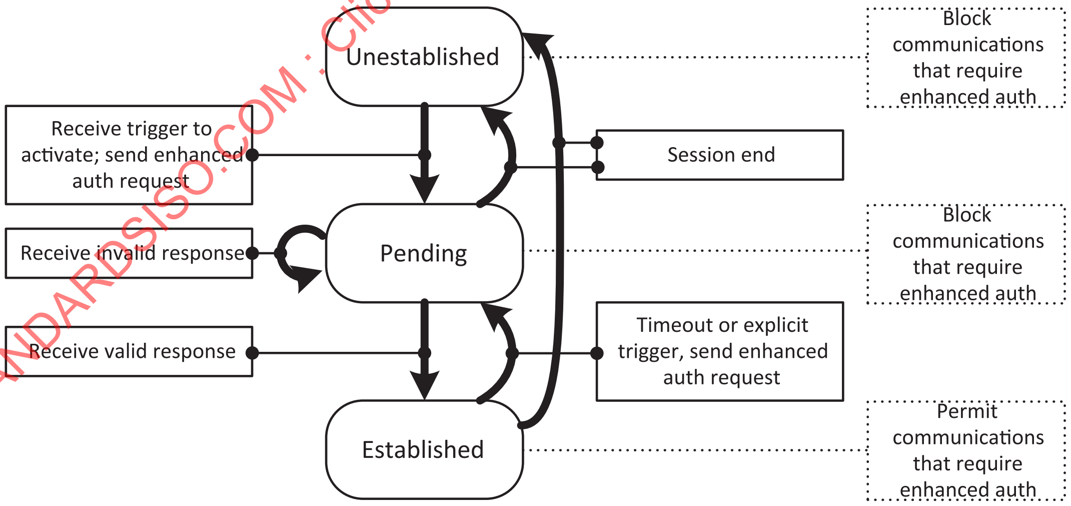

Unestablished
Pending
Established
Receive trigger to activate; send enhanced auth request
Receive invalid response
Receive valid response
Session end
Timeout or explicit trigger, send enhanced auth request
Block communications that require enhanced auth
Block communications that require enhanced auth
Permit communications that require enhanced auth

Figure 27 — Informal state machine for the Enhanced Authentication module

7.3.3 State for accessor role enhanced authentication

For any individual secure session and any enhanced authentication secret identifier, the *accessor role enhanced authentication state* may take one of the following values:

— *Established* — a request has been received and a valid response has been sent;

— *Unestablished* — otherwise.

The state with respect to an unknown identifier is automatically Unestablished.

If an Enhanced Authentication Request Access Control PDU is received containing a particular identifier is received and the Security Subsystem can construct a valid response per the mechanisms specified below in 7.3.6, then:

— The Security Subsystem constructs that response and sends it via `Sec-AL-AccessControl.request`.

— The state with respect to that enhanced authentication secret is transitioned to Established.

The accessor role enhanced authentication state with respect to a particular secret transitions from *Established to Unestablished* on session end, on access control policy-defined timeout, or in response to other events or triggers is to be specified in the specification of an application that uses the mechanisms provided by this document.

7.3.4 Use by Access Control

The access control policy for any resource or action may require a particular state for owner role or accessor role before the action may be taken. The design supports multiple enhanced authentication secrets. If there are multiple enhanced authentication secrets the policy may be specific for each secret or may be more general, for example requiring that owner role enhanced authentication state is *Established* for at least one of the secrets.

For owner role, each enhanced authentication secret can be identified within the policy.

For accessor role, the secrets cannot be known in advance and so the access control policy can only address the number of secrets for which the state is *Established*.

The access control policy may also specify action to be taken if a peer provides a certain number of invalid responses to an authentication request, e.g. it may require that the session is terminated.

7.3.5 Methods for providing enhanced authentication

Enhanced authentication is established with the following process flow:

— An instance of the Security Subsystem acting in the owner role sends an Enhanced Authentication Request Access Control PDU.

— An instance of the Security Subsystem acting in the accessor role sends an Enhanced Authentication Response Access Control PDU containing a valid response to the corresponding request.

The following mechanism is supported:

— Enhanced authentication with "Secure Password Authenticated Key Exchange 2" (SPAKE2), see 7.3.6;

7.3.6 Enhanced authentication using SPAKE2

SPAKE2 is carried out as specified in [13] with the following configuration choices:

— The group used is the elliptic curve NISTp256.

— H is SHA-256.

— *A*, “Alice’s identifier”, is the certificate presented by the TLS server in the handshake and provided to the client via `Sec-Sess-StartSession.indication`.

— *B*, “Bob’s identifier”, is the certificate presented by the TLS client in the handshake and provided to the server via `Sec-Sess-StartSession.indication`.

The request contains *T*, calculated as specified in 3.2 of [13]. It may also contain a password hint to allow the responder to select between multiple recently received secrets.

The response contains *S*, calculated as specified in 3.2 of [13]. It also contains a field `hashOfKPrimeRespId`, calculated as follows:

— The quantity *K* is the 32-byte output of the SHA-256 hash of the session variables as specified in 3.2 of [13].

— ID is the certificate out of *A* and *B* as defined above that is owned by the responder.

— The quantity `hashOfKPrimeRespId` is calculated as the SHA-256 hash of [len(*K*), *K*, len(ID), ID].

The response may request proof of knowledge from the original requester. In this case, the requester sends a "requester response" that contains the field `hashOfKPrimeRelId`, calculated as follows:

— The quantity *K* is the 32-byte output of the SHA-256 hash of the session variables as specified in 3.2 of [13].

— ID is the certificate out of *A* and *B* as defined above that is owned by the requester.

— The quantity `hashOfKPrimeRespId` is calculated as the SHA-256 hash of [len(*K*), *K*, len(ID), ID].

With regard to enhanced authentication, the requester may be only in one of the following states:

— No enhanced authentication,

— Responder authenticated.

The responder may be in only one of the following states:

— No enhanced authentication,

— Responder believed authenticated, no requester authentication requested,

— Responder believed authenticated, no requester authentication received,

— Mutually authenticated.

The Request data type is defined in 7.5.12. The Response data type is defined in 7.6.13.

7.4 Extended authentication

This subclause specifies *extended authentication*. In extended authentication, one party acts as the owner and the other party acts as the accessor. The owner requests that the accessor provides information about their authorisation in addition to the information provided in the handshake of the secure session by sending an Extended Authentication Request Access Control PDU over the secure session, and the accessor may respond by sending an Extended Authentication Response Access Control PDU to the owner to provide this additional information. The Extended Authorisation Response Access Control PDU is signed by a 1609.2 certificate which indicates the certificate holder's permissions.

The Extended Authentication Request Access Control PDUs are sent based on triggers that are to be specified in the specification of an application that uses the mechanisms provided by this document.

The Request PDU includes a structured statement of the authorisations that the owner is requesting the accessor to demonstrate. It is specified in 7.5.6 to 7.5.8. The structure supports combining atomic authorisation requests via AND and OR operations. The structure includes a random element to provide assurance that responses are freshly generated.

The Response PDU is sent in response to a Request PDU. It is specified in 7.5.9. It consists of an Ieee1609Dot2Data of type signed, containing the signature of the accessor on a value derived from the request.

The accessor should create a response that demonstrates the authorisations requested in the Request. However, the accessor may create a response that does not match the authorisations requested in the Request. In this case the owner may choose to store the received authorisations from the Response, may choose to ignore the Response, or may take another action, to be specified in the specification of an application that uses the mechanisms provided by this document.

The accessor may create a multipart Response to a Request, if the Request indicates (by setting `multipartAccepted` to True) that a multipart Response is acceptable. This enables an accessor to respond to a Request even if the requested authorisations are in multiple certificates belonging to the accessor. A multipart response consists of a series of individual Responses, each signed by a different certificate, sent as separate Access Control PDUs.

7.5 Data types

7.5.1 General

Access Control PDUs are defined in ASN.1 and encoded with the Canonical Octet Encoding Rules (COER). The ASN .1 basic notation is specified in ISO/IEC 8824-1[11]. The COER are specified in [12].

7.5.2 Imports

The data type definitions import the following definitions from IEEE 1609.2: `Psid`, `PsidSspRange`, `Ieee1609Dot2Data`.

7.5.3 Iso21177AccessControlPdu

The ASN.1 type of Access Control PDUs is `Iso21177AccessControlPdu`.

```
Iso21177AccessControlPdu ::= SEQUENCE {
   messageId   ISO-21177-ACCESS-CONTROL-ID-
        TYPE.&id({Iso21177AccessControlPduTypes }),
   value       ISO-21177-ACCESS-CONTROL-ID-
        TYPE.&Type({Iso21177AccessControlPduTypes}{@.messageId}), ...
}
ISO-21177-ACCESS-CONTROL-ID-TYPE ::= CLASS {
   &id Iso21177AccessCtrlPduId UNIQUE,
   &Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}
Iso21177AccessControlPduTypes ISO-21177-ACCESS-CONTROL-ID-TYPE ::= {
   { AccessControlResult  IDENTIFIED BY accessControlResultId } |
   { ExtendedAuthPdu      IDENTIFIED BY extendedAuthId } |
   { EnhancedAuthPdu      IDENTIFIED BY enhancedAuthId } ,
   ...
}
Iso21177AccessCtrlPduId ::= INTEGER (0..255)
   iso21177AccessCtrlPduId-reserved   Iso21177AccessCtrlPduId ::= 0 --'00'H
   accessControlResultId              Iso21177AccessCtrlPduId ::= 1 --'01'H
   extendedAuthId                     Iso21177AccessCtrlPduId ::= 2 --'01'H
   enhancedAuthId                     Iso21177AccessCtrlPduId ::= 3 --'02'H
```

7.5.4 AccessControlResult

The ASN.1 type `AccessControlResult` is used to communicate the result of an access control activity.

```
AccessControlResult::= INTEGER {
   success    (0),
   authorisation-failure    (1)
} (0..255)
```

7.5.5 ExtendedAuthPdu

The ASN.1 type of the Access Control PDUs associated with extended authentication as specified in 7.4 is `ExtendedAuthPdu`.

```
ExtendedAuthPdu ::= SEQUENCE {
   messageId EXTENDED-AUTH-ID-TYPE.&id({ExtendedAuthPduTypes}),
   value     EXTENDED-AUTH-ID-TYPE.&Type({ExtendedAuthPduTypes}{@.messageId}), ...
}
EXTENDED-AUTH-ID-TYPE ::= CLASS {
   &id ExtendedAuthPduId UNIQUE,
   &Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}
ExtendedAuthPduTypes EXTENDED-AUTH-ID-TYPE ::= {
   { ExtendedAuthRequest     IDENTIFIED BY extendedAuthRequestId } |
   { ExtendedAuthResponse    IDENTIFIED BY extendedAuthResponseId } ,
   ...
}
ExtendedAuthPduId ::= INTEGER (0..255)
   extendedAuthPduId-reserved   ExtendedAuthPduId ::= 0 --'00'H
   extendedAuthRequestId        ExtendedAuthPduId ::= 1 --'01'H
   extendedAuthResponseId       ExtendedAuthPduId ::= 2 --'02'H
```

7.5.6 ExtendedAuthRequest

The ASN.1 type `ExtendedAuthRequest` contains an extended authentication request and associated metadata. In this ASN.1 type

— the component `multipartAccepted` indicates that the response may consist of multiple parts, i.e. that the responder may use multiple different credentials to demonstrate their authorisation;

— the component nonce is a freshly-generated strong random number, used in the generation of the response to protect against replay attacks;

— the component inner contains the details of the permissions that are being requested.

```
ExtendedAuthRequest ::= SEQUENCE {
   multipartAccepted   BOOLEAN,
   nonce               OCTET STRING (SIZE(32)),
   inner               InnerExtendedAuthRequest
}
```

7.5.7 InnerExtendedAuthRequest

The ASN.1 type `InnerExtendedAuthRequest` contains an extended authentication request. In this ASN.1 type

— the component and indicates that the responder is being requested to demonstrate that they have all of the authorisations indicated within the field;

— the component or indicates that the responder is being requested to demonstrate that they have at least one of the authorisations indicated within the field;

— the component contents contains a single authorisation request.

Since `InnerExtendedAuthRequest` may contain an instance of itself, in principle there may be an arbitrary number of levels of nesting in a final extended authentication request. A conformant implementation shall support at least four levels of nesting, i.e. shall support an `InnerExtendedAuthRequest` containing an `InnerExtendedAuthRequest` containing an `InnerExtendedAuthRequest` containing an `InnerExtendedAuthRequest`, and may support more.

```
InnerExtendedAuthRequest ::= CHOICE {
   and         SEQUENCE OF InnerExtendedAuthRequest,
   or          SEQUENCE OF InnerExtendedAuthRequest,
   contents    AtomicExtendedAuthRequest
}
```

7.5.8 AtomicExtendedAuthRequest

The ASN.1 type `AtomicExtendedAuthRequest` contains a single authentication request. In this ASN.1 type

— the component `psid` indicates that the responder is being requested to demonstrate that they have permissions related to the indicated PSID;

— the component `psidSspRange` indicates that the responder is being requested to demonstrate that they have permissions consistent with PSID and SSP Range, as defined in IEEE Std. 1609.2™;

— the component `idRegExp` indicates that the responder is being requested to provide an IEEE Std. 1609.2™ certificate whose name matches the indicated regular expression. The regular expression is conformant to IEEE 1003, Clause 9;

— the component `issuer` indicates that the responder is being requested to provide an IEEE Std. 1609.2™ certificate such that one of the issuing Certificate Authority certificates in the certificate chain, see IEEE Std. 1609.2™ for definitions of certificate authorities and certificate chain, has the indicated eight-byte hash value.

```
AtomicExtendedAuthRequest ::= CHOICE {
   psid            Psid,
   psidSspRange    PsidSspRange,
   idRegExp        IA5String,
   issuer          OCTET STRING (SIZE(8)),
...
}
```

7.5.9 ExtendedAuthResponse

The ASN.1 type `ExtendedAuthResponse` contains an extended authentication response. It is an IEEE 1609.2 Signed SPDU in which the authType field defined in IEEE 1609.2b[8] is present and set to the value iso21177ExtendedAuth.

```
ExtendedAuthResponse ::= Ieee1609Dot2Data (WITH COMPONENTS {...,
   content (WITH COMPONENTS {...,
      signedData (WITH COMPONENTS {...,
         tbsData (WITH COMPONENTS {...,
            payload (WITH COMPONENTS {...,
               data (WITH COMPONENTS {...,
                  content (WITH COMPONENTS {
                     unsecuredData (CONTAINING ExtendedAuthResponsePayload)
                  })
               })
            }),
            headerInfo (WITH COMPONENTS {...,
               generationTime PRESENT,
               expiryTime ABSENT,
               generationLocation ABSENT,
               p2pcdLearningRequest ABSENT,
               missingCrlIdentifier ABSENT,
               encryptionKey ABSENT,
               authType (iso21177ExtendedAuth) PRESENT
            })
         })
      })
   })
})
```

7.5.10 ExtendedAuthResponsePayload

The ASN.1 type `ExtendedAuthResponsePayload` contains the payload of an extended authentication response. In this ASN.1 type

— the component `part` indicates the part number of this response within a multipart response. It shall be between 1 and `maxPart`. If the response is not multipart this field shall take the value 1.

— the component `maxPart` indicates the number of parts in a multipart response. If the response is to multipart this component shall take the value 1.

— the component `requestHash` is the SHA-256 hash of the entire I`so21177AccessControlPdu` that contained the corresponding request.

```
ExtendedAuthResponsePayload ::= SEQUENCE {
   part           INTEGER,
   maxPart        INTEGER,
   requestHash    OCTET STRING(SIZE(32))
}
```

7.5.11 EnhancedAuthPdu

The ASN.1 type `EnhancedAuthPdu` is the container type for Access Control PDUs associated with enhanced authentication as specified in 7.4.

```
EnhancedAuthPdu ::= SEQUENCE {
   messageId   ENHANCED-AUTH-ID-TYPE.&id({EnhancedAuthPduTypes}),
   value       ENHANCED-AUTH-ID-TYPE.&Type({EnhancedAuthPduTypes}{@.messageId}), ...
}
ENHANCED-AUTH-ID-TYPE ::= CLASS {
   &id   EnhancedAuthPduId UNIQUE,
   &Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}
EnhancedAuthPduTypes ENHANCED-AUTH-ID-TYPE ::= {
   { SpakeRequest              IDENTIFIED BY spakeRequestId } |
   { SpakeResponse             IDENTIFIED BY spakeResponseId } |
   { SpakeRequesterResponse    IDENTIFIED BY spakeRequesterResponseId } ,
   ...
}
EnhancedAuthPduId ::= INTEGER (0..255)
   enhancedAuthPduId-reserved    EnhancedAuthPduId ::= 0 --'00'H
   spakeRequestId                EnhancedAuthPduId ::= 1 --'01'H
   spakeResponseId               EnhancedAuthPduId ::= 2 --'02'H
   spakeRequesterResponseId      EnhancedAuthPduId ::= 3 --'03'H
```

7.5.12 SpakeRequest

The ASN.1 type `SpakeRequest` is the request for enhanced authentication using SPAKE2. The component t is set as specified in 7.3.6. The component hint takes one of two forms:

— If it is 0, no hint is provided.

— If the top bit is 1, then the last 3 bits are the last 3 bits of the SHA_256 hash of the shared secret.

```
SpakeRequest ::= SEQUENCE {
   t       EccP256CurvePoint,
   hint    OCTET STRING (SIZE(1))
}
```

7.5.13 SpakeResponse

The ASN.1 type `SpakeResponse` is the response to a request for enhanced authentication using SPAKE2. The components are set as specified in 7.3.6.

```
SpakeResponse ::= SEQUENCE {
   s                    EccP256CurvePoint,
   hashOfKPrimeRespId   OCTET STRING (SIZE(32)),
   responseRequest      BOOLEAN
}
```

7.5.14 SpakeRequesterResponse

The ASN.1 type `SpakeRequesterResponse` is the original requester's response to a request to provide mutual authentication for enhanced authentication using SPAKE2. The content is set as specified in 7.3.6.

```
SpakeRequesterResponse ::= OCTET STRING (SIZE(32))
```

7.6 App-Sec Interface

7.6.1 App-Sec-Configure.request

This service primitive instructs the Security Subsystem to activate an instance of the Secure Session Services.

The parameters of the service primitive are as follows:

```
App-Sec-Configure.request (
   Application ID,
   Role,
   Socket,
   Session Type,
   Proxied,
   Session ID,
   Transport Mechanism Type (optional),
   Cryptomaterial Handle (optional),
   Broker Info,
   Proxying Session ID
)
```

Table 2 — Parameters for `App-Sec-Configure.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Role	Enumerated	Client, Server	The role that the home trusted entity (ITS-SU) will play in the secure session.
Socket	Socket Instance	Any	A socket instance for a communications session without cryptographic security.
Session Type	Enumerated	Internal, External	Used to determine whether cryptographic security is necessary for the communications session.
Proxied	Integer	True, False	Used to indicate whether the secure session handshake should be brokered as specified in 6.9.
— The following parameter is supplied only if *Role* is Client.			
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
— The following parameters are supplied only if *Session Type* is External.			
Transport Mechanism Type	Enumerated	Reliable, Unreliable	Indicates whether a secure session protocol for reliable or unreliable transport shall be used (in this case, DTLS or TLS).
Cryptomaterial Handle	Cryptomaterial Handle	A valid cryptomaterial handle associated with a certificate as defined in IEEE Std. 1609.2™ and containing permissions consistent with the access control policy.	The Cryptomaterial handle to be used for signatures in the handshake of the cryptographic secure session.
— The following parameter is supplied only if *Proxied* is True.			

Table 2 *(continued)*

Name	Type	Valid range	Description
Broker Info	Octet String	Any	The Broker Info used to identify the session while it is being brokered. The inner semantics of the Broker Info string are not used by the services defined in this standard; it is simply used as an identifier.
— The following parameter is supplied only if *Proxied* is True and *Role* is Client.			
Proxying Session ID	Integer	Any	The ID to be provided as Session ID to `AL-Sess-ClientHelloProxy`.

On receipt, the Security Subsystem carries out the steps specified in 6.4 step b).

7.6.2 App-Sec-Configure.confirm

This service primitive returns the results of the corresponding request primitive.

The parameters of the service primitive are as follows:

```
App-Sec-Configure.confirm (
   Result
)
```

Table 3 — Parameter for `App-Sec-Configure.confirm`

Name	Type	Valid range	Description
Result	Enumerated	Success, Secure session type not available, Secure session type not permitted for this Application, Cryptomaterial Handle not permitted	The result of the request.

7.6.3 App-Sec-StartSession.indication

This service primitive provides the Session ID when a new session is started with a Secure Session instance in the Server role. It is generated in response to `Sec-Sess-Start.indication`, see 9.3.3.

The parameters of the service primitive are as follows:

```
App-Sec-StartSession.indication (
   Application ID,
   Session ID
)
```

Table 4 — Parameters for `App-Sec-StartSession.indication`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.

7.6.4 App-Sec-Data.request

This service primitive is used to request the Security Subsystem to sign an APDU before it is sent over the security session, to provide non-repudiation to the individual APDU.

The parameters of the service primitive are as follows:

```
App-Sec-Data.request (
   Application ID,
   Session ID,
   Cryptomaterial Handle,
   Data,
   Signing Parameters
)
```

Table 5 — Parameters for `App-Sec-Data.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Cryptomaterial Handle	Cryptomaterial Handle	A valid cryptomaterial handle per 1609.2 associated with a certificate. See IEEE Std. 1609.2™ for further details.	The cryptomaterial handle to be used to sign the APDU.
Data	Octet String	Any Octet String	The APDU to be signed.
Signing Parameters	Structured	The set of parameters to be provided to the 1609.2 `Sec-SignedData.request` primitive. See IEEE Std. 1609.2™ for further details.	The set of parameters to be provided to the IEEE 1609.2 `Sec-SignedData.request` primitive.

On receipt, the Security Subsystem attempts to sign the indicated Data with the indicated Cryptomaterial handle and Signing Parameters.

7.6.5 App-Sec-Data.confirm

This service primitive returns the result of the corresponding request primitive.

The parameters of the service primitive are as follows:

```
App-Sec-Data.confirm (
   Result Code,
   Signed Data (optional)
)
```

Table 6 — Parameters for `App-Sec-Data.confirm`

Name	Type	Valid range	Description
Result Code	Enumerated	Any Result Code that may be returned by the IEEE 1609.2 primitive `Sec-SignedData.confirm`. See IEEE Std. 1609.2™ for further details.	The result of the operation.
Signed Data	Octet String	An Ieee1609Dot2Data of type signedData. See IEEE Std. 1609.2™ for further details.	If Result Code is Success, the signed APDU. Otherwise, omitted.

On receipt of this primitive, if the parameter Result Code is Success, the Application invokes `App-AL-Data.request` to submit the data for sending. Otherwise, no effect is specified.

7.6.6 App-Sec-Incoming.request

This service primitive is generated by the Application to request that the Security Subsystem applies the access control policy to an incoming APDU.

The parameters of the service primitive are as follows:

```
App-Sec-Incoming.request (
   Application ID,
   Session ID,
   Data,
   Is Ieee1609Dot2Data,
   Signed Data Verification Parameters (optional)
)
```

Table 7 — Parameters for `App-Sec-Incoming.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Data	Octet String	If *Is Ieee1609Dot2Data* is true, an Ieee1609Dot2Data. If *Is Ieee1609Dot2Data* is false, any Octet String. See IEEE Std. 1609.2™ for further details.	The received APDU.
Is Ieee1609Dot2Data	Boolean	True, False	Whether the received APDU is an Ieee1609Dot2Data.
Signed Data Verification Parameters	Structured	The set of parameters to be provided to the IEEE 1609.2 `Sec-SignedDataVerification.request` primitive. See IEEE Std. 1609.2™ for further details.	Provided only if *Is Ieee1609Dot2Data* is True. The set of parameters to be provided to the IEEE 1609.2 `Sec-SignedDataVerification.request` primitive

On receipt, the Security Subsystem carries out the activities specified in 6.8, step e) 3) The result of the operation is returned via `App-Sec-Incoming.confirm`.

7.6.7 App-Sec-Incoming.confirm

This service primitive returns the result of the corresponding request primitive.

The parameters of the service primitive are as follows:

```
App-Sec-Incoming.confirm (
   Result
)
```

Table 8 — Parameter for `App-Sec-Incoming.confirm`

Name	Type	Valid range	Description
Result	Enumerated	Success, Invalid Ieee1609Dot2Data Type, Invalid Signed Ieee1609Dot2Data, Invalid APDU per access control policy/request sent, Invalid APDU per access control policy/no request sent	The result of the request.

No effect of receipt is specified.

NOTE Similarly to the primitive `Sec-Signed-Data-Verification.request` specified in IEEE Std. 1609.2™, this primitive does not return the APDU contents, as the Application is assumed to be able to parse the IEEE 1609.2 headers if present to recover the Application payload.

7.6.8 App-Sec-EndSession.request

This service primitive is used to request that a particular session is ended.

The parameters of the service primitive are as follows:

```
App-Sec-EndSession.request (
   Application ID,
   Session ID
)
```

Table 9 — Parameter for `App-Sec-EndSession.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any existing session ID value associated with the *Application ID*.	An identifier for the session to be ended.

On receipt, the Security Subsystem carries out the activities specified in 6.10, step g) 1) ii).

7.6.9 App-Sec-EndSession.confirm

This service primitive confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

7.6.10 App-Sec-EndSession.indication

This service primitive is used to indicate that the Secure Session services have detected or determined that a particular secure session has ended. It is generated when the Security Subsystem determines that a session should be ended per the access control policy, or when the Security Subsystem receives a `Sec-Sess-EndSession.indication` primitive from the Secure Session Services.

If the Security Subsystem has determined that the session should be ended, the Security Subsystem also generates a `Sec-AL-EndSession.request`.

The parameters of the service primitive are as follows:

```
App-Sec-EndSession.indication (
   Application ID,
    Session ID,
    Originating Layer
)
```

Table 10 — Parameters for `App-Sec-EndSession.indication`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any existing session ID value associated with the *Application ID*.	An identifier for the session to be ended.
Originating Layer	Enumerated	Security Subsystem, Secure Session Service	The layer at which the determination was made that the session should be or had been ended.

On receipt, the Application takes any actions associated with the end of the secure session.

7.6.11 App-Sec-Deactivate.request

This service primitive is used to request that a particular instance of the Secure Session service is deactivated as specified in 6.12.

The parameters of the service primitive are as follows:

```
App-Sec-Deactivate.request (
    Application ID,
    Secure Session Instance ID
)
```

Table 11 — Parameters for `App-Sec-Deactivate.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Secure Session Instance ID	Integer	Any existing secure session instance ID value associated with the *Application ID*.	An identifier for the secure session instance to be deactivated.

On receipt, the Security Subsystem carries out the activities specified in 6.12, step g) 1) ii).

7.6.12 App-Sec-Deactivate.confirm

This service primitive confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

None specified.

7.6.13 App-Sec-Deactivate.indication

This service primitive is used by the Security Subsystem to notify the Application that, per the access control policy, the indicated secure session instance shall be deactivated. When the Security Subsystem generates this primitive, it also generates a corresponding `Sec-Sess-Deactivate.request`.

The parameters of the service primitive are as follows:

```
App-Sec-Deactivate.indication (
   Application ID,
   Secure Session Instance ID
)
```

Table 12 — Parameters for `App-Sec-Deactivate.indication`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Secure Session Instance ID	Integer	Any existing secure session instance ID value associated with the *Application ID*.	An identifier for the secure session instance to be deactivated.

On receipt, the Application takes any steps associated with the end of a secure session instance.

7.7 Security Subsystem internal interface

7.7.1 General

This subclause defines the interfaces between the Access Control Subsystem, which is application-specific, and the Authorization State Subsystem, which is not application-specific.

7.7.2 Sec-AuthState.request

This service primitive allows the Access Control Subsystem to request the current authorization state from the Authorization State Subsystem.

The parameters of the service primitive are as follows:

```
Sec-AuthState.request (
   Application ID,
   Session ID,
   Not Before,
   Location (optional)
)
```

Table 13 — Parameters for `Sec-AuthState.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application
Not Before	Date and Time	Any Date and Time in the past at the time the primitive is generated	The Authorization Subsystem returns only authorization state information that was received after the *Not Before* date and time.
Location	A 3D Location	Any	The current location of the ITS-S, indicating that authorization statements should be provided only if they are valid at that location.

7.7.3 Sec-AuthState.confirm

This service primitive allows the Authorization State Subsystem to provide the current authorization state to the Access Control Subsystem. The primitive is generated in response to the corresponding request primitive.

The parameters of the service primitive are as follows:

```
Sec-AuthState.confirm (
   Application ID,
   Session ID,
   Credential Based Authorization State,
   Enhanced Authorization State
)
```

Table 14 — Parameters for `Sec-AuthState.confirm`

Name		Type	Valid range	Description
Application ID		Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID		Integer	Any	An identifier for the session, unique within sessions for the Application
Credential Based Authorization State		Sequence of 0 or more of the following		
	AID	ITS-AID	Any ITS-AID	An ITS-AID contained in a certificate that (a) was received after the *Not Before* date and time in the corresponding request primitive (b) has not expired (c) is valid at the location provided at *Location* in the corresponding request primitive
	SSP	SSP	Any SSP valid in the context of the ITS-AID provided in *AID*	The SSP associated with *AID* in the certificate in which *AID* was received
	CertId	A HashedId8 as defined in 1609.2	Any	The HashedId8 of the certificate in which *AID* was received
	Reception Time	A date and time	Any date and time in the past at the time the service primitive was generated	The time at which the certificate indicated by *CertId* was received
Enhanced Authorization State		Sequence of 0 or more of the following		
	Reception Time	A date and time	Any date and time in the past at the time the service primitive was generated	The time at which the enhanced authorization operation was completed

8 Adaptor Layer: Interfaces and data types

8.1 General

The Adaptor Layer is a protocol layer similar to the Record Protocol in TLS. The purpose of the design is to allow a single secured session to carry both Application datagrams and (security) control datagrams that affect the configuration of that session. The Adaptor Layer creates and consumes Adaptor Layer Protocol Data Units (ALPDUs) as defined in 8.2; the ALPDU type field indicates the type of the ALPDU and so the action for the Adaptor Layer to take. The supported data types are defined in Table 15.

Table 15 — Data types

Content type	Direction	Received via	Action
Data	Outgoing	`App-AL-Data.request`	Create ALPDU of type Apdu, send via `AL-Sess-Data.request`.
	Incoming	`AL-Sess-Data.indication`	Determine that ALPDU is of type Apdu, send payload to application via `App-AL-Data.indication`.
Access Control	Outgoing	`Sec-AL-AccessControl.request`	Create ALPDU of type Access Control, send via `AL-Sess-Data.request`.
	Incoming	`AL-Sess-Data.indication`	Determine that ALPDU is of type Access-Control, send payload to Security Services via `Sec-AL-AccessControl.indication`.
TLS Client Message 1	Outgoing	`AL-Sess-ClientHello.indication`	Create ALPDU of type `TlsClientMsg1`, send over indicated secure session via `AL-Sess-Data.request`.
	Incoming	`AL-Sess-Data.indication`	Determine that ALPDU is of type `TlsClientMsg1`. Check configuration properties related to the BrokerInfo field. If they indicate that the ALPDU shall be proxied, send the received ALPDU exactly as received over the indicated secure session via `AL-Sess-Data.request`. If they indicate that the proxied server session is resident on this ITS-S, extract the information and provide to the secure session instance via `AL-Sess-ClientHello.request`.
TLS Server Message 1	Outgoing	`AL-Sess-ServerHello.indication`	Create ALPDU of type `TlsServerMsg1`, send over indicated secure session via `AL-Sess-Data.request`.
	Incoming	`AL-Sess-Data.indication`	Determine that ALPDU is of type `TlsServerMsg1`. Check configuration properties related to the BrokerInfo field. If they indicate that the ALPDU shall be proxied, send the received ALPDU exactly as received over the indicated secure session via `AL-Sess-Data.request`. If they indicate that the proxied client session is resident on this ITS-S, extract the information and provide to the secure session instance via `AL-Sess-ServerHello.request`.

8.2 Data types

8.2.1 General

Adaptor Layer PDUs are defined in ASN.1 and encoded with the Canonical Octet Encoding Rules (COER). The ASN .1 basic notation is specified in ISO/IEC 8824-1[11]. The COER are specified in [12].

8.2.2 Iso21177AdaptorLayerPDU

The ASN.1 type `Iso21177AdaptorLayerPDU` is the container type for Adaptor Layer PDUs.

```
Iso21177AdaptorLayerPDU ::= SEQUENCE {
   messageId   ISO-21177-ADAPTOR-LAYER-ID-
         TYPE.&id({Iso21177AdaptorLayerPduTypes }),
   value       ISO-21177-ADAPTOR-LAYER-ID-
         TYPE.&Type({Iso21177AdaptorLayerPduTypes}{@.messageId}),
   ...}
ISO-21177-ADAPTOR-LAYER-ID-TYPE ::= CLASS {
   &id Iso21177AdaptorLayerPduId UNIQUE,
   &Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}
Iso21177AdaptorLayerPduTypes ISO-21177-ADAPTOR-LAYER-ID-TYPE ::= {
   { Apdu   IDENTIFIED BY apduId } |
   { AccessControl   IDENTIFIED BY accessControlId } |
   { TlsClientMsg1   IDENTIFIED BY tlsClientMsg1Id } |
   { TlsServerMsg1   IDENTIFIED BY tlsServerMsg1Id } ,
...
}
Iso21177AdaptorLayerPduId ::= INTEGER (0..255)
iso21177AdaptorLayerPduId-reserved   Iso21177AdaptorLayerPduId ::= 0 --'00'H
apduId             Iso21177AdaptorLayerPduId ::= 1 --'01'H
accessControlId    Iso21177AdaptorLayerPduId ::= 2 --'01'H
tlsClientMsg1Id    Iso21177AdaptorLayerPduId ::= 3 --'01'H
tlsServerMsg1Id    Iso21177AdaptorLayerPduId ::= 4 --'01'H
```

8.2.3 Apdu

The ASN.1 type Apdu is created from the Data parameter from the `App-AL-Data.request` service primitive, see 8.3.1.

```
Apdu ::= OCTET STRING
```

8.2.4 Access Control

The ASN.1 type `AccessControl` contains a C-OER encoded Iso21177AccessControlPdu as created by the Security Subsystem and provided to the Adaptor Layer as the Data parameter from the `Sec-AL-AccessControl.request` primitive, see 8.4.1.

```
AccessControl ::= Iso21177AccessControlPdu
```

8.2.5 TlsClientMsg1

The ASN.1 type `TlsClientMsg1` is created by the Adaptor Layer from the `ClientHello` and Broker Info parameters of `AL-Sess-ClientHelloProxy.indication`, see 9.4.7.

```
TlsClientMsg1 ::= SEQUENCE {
   clientHelloMsg    OCTET STRING,
   brokerInfo        OCTET STRING
}
```

8.2.6 TlsServerMsg1

The ASN.1 type `TlsServerMsg1` is created by the Adaptor Layer from the `ServerHello` and Broker Info parameters of `AL-Sess-ClientHelloProxy.indication`, see 9.4.7.

```
TlsServerMsg1 ::= SEQUENCE {
   serverHelloMsg    OCTET STRING,
   brokerInfo        OCTET STRING
}
```

8.3 App-AL Interface

8.3.1 App-AL-Data.request

This service primitive is used by the Application to submit an APDU for transmission across the secure session.

The parameters of the service primitive are as follows:

```
App-AL-Data.request (
   Application ID,
   Session ID,
   Data
)
```

Table 16 — Parameters of `App-AL-Data.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Data	Octet String	Any	The APDU to be sent.

On receipt, the Adaptor Layer:

— Generates an `App-AL-Data.confirm` service primitive to confirm receipt of the `App-AL-Data.request` service primitive;

— Creates ALPDU, an Iso21177AdaptorLayerPdu with the component `messageId` equal to `apduId` and Data parameter from this service primitive;

— Generates an `AL-Sess-Data.request` primitive with (Application ID, Session ID, ALPDU) as the Application ID, Session ID, Data parameters.

8.3.2 App-AL-Data.confirm

This service primitive confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

8.3.3 App-AL-Data.indication

This service primitive passes a received APDU from the Adaptor Layer to the Application. It is generated when the Adaptor Layer receives an `AL-Sess-Data.indication` service primitive, see 9.4.3, containing an Iso21177AdaptorLayerPdu with component `messageId` equal to `apduId`.

The parameters of the service primitive are as follows:

```
App-AL-Data.indication (
   Application ID,
   Session ID,
   Data
)
```

Table 17 — Parameters for `App-AL-Data.indication`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Data	Octet String	Any	The XXX field from the received ALPDU.

On receipt, the Application carries out the activities specified in 6.8, step e) 2).

8.3.4 App-AL-EnableProxy.request

This service primitive is used by the Application to configure the Adaptor Layer to proxy TLS handshake messages between two Secure Session instances.

The parameters of the service primitive are as follows:

```
App-AL-EnableProxy.request (
   Application ID,
   Client Side Session ID,
   Server Side Session ID (optional),
   Broker Info,
   Role
)
```

Table 18 — Parameters for `App-AL-EnableProxy.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Client Side Session ID	Integer	Any	An identifier for the session that will receive the incoming `TlsClientMsg1` ALPDU and will be sent the outgoing `TlsServerMsg1` ALPDU.
Server Side Session ID	Integer	Any	An identifier for the session that will be sent the outgoing `TlsClientMsg1` ALPDU and will receive the `TlsServerMsg1` ALPDU. Included only if *Role* is Broker.
Broker Info	Octet String	Any	The Broker Info used to identify the session while it is being brokered. The inner semantics of the Broker Info string are not used by the services defined in this document; it is simply used as an identifier.
Role	Enumerated	Server, Broker	If Broker, then all incoming and outgoing handshake messages are sent via `AL-Sess-Data.request` and Server Side Session ID must be provided. If Server, then the Client Hello is passed to the Secure Session via `AL-Sess-ClientHello.request` and the Server Hello and other server handshake messages are received via `AL-Sess-ServerHello.indication`, and the Server Side Session ID is not used but instead the Broker Info is used to associate the Client Hello with the Server Hello.

If *Role* is Broker, then on receipt of this primitive, the Adaptor Layer stores the following information:

— On receipt of an `AL-Sess-Data.indication`, see 9.4.3, where all the following hold:

  — Data is an `Iso21177AdaptorLayerPdu` with:

    — the component `messageId` equal to `tlsClientMsg1Id`;

    — the `brokerInfo` component in `TlsClientMsg1` equal to the Broker Info parameter to this primitive;

    - Session ID equal to the Client Side Session ID.
  - Then the received Iso21177AdaptorLayerPdu shall be forwarded unaltered to the secure session instance identified by Server Side Session ID.
- On receipt of an `AL-Sess-Data.indication`, see 9.4.3, where all the following hold:
  - Data is an `Iso21177AdaptorLayerPdu` with:
    - the component `messageId` equal to `tlsClientMsg1Id`;
    - the `brokerInfo` component in `TlsClientMsg1` equal to the Broker Info parameter to this primitive;
    - Session ID equal to the Server Side Session ID.
  - Then the received `Iso21177AdaptorLayerPdu` shall be forwarded unaltered to the secure session instance identified by Client Side Session ID.

If *Role* is Server, then on receipt of this primitive, the Adaptor Layer stores the following information:

- On receipt of an `AL-Sess-Data.indication`, see 9.4.3, where all the following hold:
  - Data is an `Iso21177AdaptorLayerPdu` with:
    - the component `messageId` equal to `tlsClientMsg1Id`;
    - the `brokerInfo` component in `TlsClientMsg1` equal to the Broker Info parameter to this primitive;
    - Session ID equal to the Client Side Session ID.
  - Then the Adaptor Layer shall create an `AL-Sess-ClientHelloProxy.request` service primitive, see 9.4.6, where:
    - Application ID is the Application ID field from this service primitive;
    - `ClientHello` is the `clientHello` component in the `TlsClientMsg1`;
    - Broker Info is the brokerInfo component in the `TlsClientMsg1`.
- On receipt of an `AL-Sess-ServerHelloProxy.indication`, see 9.4.9, where all the following hold:
  - Application ID is the Application ID field from this primitive;
  - Broker Info is the `brokerInfo` component from this primitive;
  - then the Adaptor layer shall create an `Iso21177AdaptorLayerPdu` with:
    - the component `messageId` equal to `tlsServerMsg1Id`;
    - the `serverHello` field in the `TlsServerMsg1` equal to the Broker Info parameter to this primitive;
    - the `brokerInfo` field in the `TlsServerMsg1` equal to the Broker Info parameter to this primitive.
  - The Adaptor Layer is then to forward the `Iso21177AdaptorLayerPdu` to the secure session instance identified by Client Side Session ID.

8.4 Sec-AL Interface

8.4.1 Sec-AL-AccessControl.request

This service primitive passes an Access Control PDU from the Security Subsystem to the Adaptor Layer for transmission over a secure session. The Access Control PDU format is specified in 7.5 but is opaque to the Adaptor Layer.

The parameters of the service primitive are as follows:

```
Sec-AL-AccessControl.request (
   Application ID,
   Session ID,
   Data

 )
```

Table 19 — Parameters for `Sec-AL-AccessControl.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Data	Octet String	A C-OER encoded `Iso21177AccessControlPdu`	The Access Control PDU.

On receipt, the Adaptor Layer:

— Generates a `Sec-AL-AccessControl.confirm` to confirm receipt of this primitive;

— Creates ALPDU, an `Iso21177AdaptorLayerPdu` with the component `messageId` equal to `accessControlId` and value an OCTET STRING containing the Data parameter from this primitive;

— Generates an `AL-Sess-Data.request` primitive with (Application ID, Session ID, ALPDU) as the *Application ID*, *Session ID*, *Data* parameters.

8.4.2 Sec-AL-AccessControl.confirm

This service primitive confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

No effect on receipt is specified.

8.4.3 Sec-AL-AccessControl.indication

This service primitive passes a received Access Control PDU from the Adaptor Layer to the Security Subsystem. It is generated when the Adaptor Layer receives an `AL-Sess-Data.indication` containing an ALPDU with `messageId` equal to `accessControlId`.

The parameters of the service primitive are as follows:

```
Sec-AL-AccessControl.indication (
   Application ID,
   Session ID,
   Data
)
```

Table 20 — Parameters for `Sec-AL-AccessControl.indication`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
Data	Octet String	Any	The value component from the received ALPDU.

On receipt, the Security Subsystem carries out the activities specified in 6.8, step d) 2).

8.4.4 Sec-AL-EndSession.request

This service primitive is used to request that a particular session is ended.

The parameters of the service primitive are as follows:

```
    Sec-AL-EndSession.request (
    Application ID,
    Session ID
)
```

Table 21 — Parameters for `Sec-AL-EndSession.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Session ID	Integer	Any existing session ID value associated with the *Application ID*.	An identifier for the session to be ended.

On receipt of this primitive, the Adaptor Layer carries out the activities specified in 6.10, step b).

8.4.5 Sec-AL-EndSession.confirm

This service primitive confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

9 Secure Session services

9.1 General

The Secure Session services maintain security state for each session.

9.2 App-Sess interfaces

9.2.1 App-Sess-EnableProxy.request

This service primitive is used by the Application to configure an instance of the Secure Session acting in the Server role to accept an incoming proxied client connection.

The parameters of the service primitive are as follows:

```
App-Sess-EnableProxy.request (
   Application ID,
   Broker Info,
   Socket
)
```

Table 22 — Parameters for `App-Sess-EnableProxy.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Broker Info	Octet String	Any	The Broker Info used to identify the session while it is being brokered. The inner semantics of the Broker Info string are not used by the services defined in this standard; it is simply used as an identifier.
Socket	Socket Instance	Any	A socket instance for a communications session without cryptographic security.

On receipt, the Secure Session instance configures itself so that when it receives an `AL-Sess-ClientHelloProxy.request` service primitive with the indicated value of Broker Info, it takes the actions specified in 6.9.4 step d) 3).

9.3 Sec-Sess interface

9.3.1 Sec-Sess-Configure.request

9.3.1.1 Function

This service primitive instructs the Secure Session Services to configure a Secure Session instance.

9.3.1.2 Semantics

The parameters of the service primitive are as follows:

```
Sec-Sess-Configure.request (
   Application ID,
   Role,
   Socket,
   Session Type,
   Proxied,
   Session ID (optional),
   Transport Mechanism Type (optional),
   Cryptomaterial Handle (optional),
   Certificate Permissions Pattern (optional),
   Inactivity Timeout (optional),
   Session Timeout (optional),
   Require Client Authentication (optional),
   Incoming Request Timeout (optional),
   Max Incoming Sessions (optional),
   Name Constraints (optional),
   Issuer Constraints (optional),
   Proxying Session ID (optional),
   Broker Info (optional)
)
```

Table 23 — Parameters for `Sec-Sess-Configure.request`

Name	Type	Valid range	Description
Application ID	Structure. See 5.4 (`ITSsapiid`)	See ISO 17419	An identifier for the ITS-S application process, unique within the home trusted entity (ITS-SU).
Role	Enumerated	Client, Server	The role that the home ITS-SU will play in the secure session.
Socket	Socket Instance	Any	A socket instance for a communications session without cryptographic security.
Session Type	Enumerated	Internal, External	Used to determine whether cryptographic security is necessary for the communications session.
Proxied	Boolean	True, False	Used to indicate whether the secure session handshake should be brokered as specified in 6.9.
— The following parameter is provided only if *Role* is Client.			
Session ID	Integer	Any	An identifier for the session, unique within sessions for the Application.
— The following parameters are provided only if *Session Type* is External.			
Transport Mechanism Type	Enumerated	Reliable, Unreliable	Indicates whether a secure session protocol for reliable or unreliable transport shall be used (in this case, DTLS or TLS).
Cryptomaterial Handle	Cryptomaterial Handle	A valid cryptomaterial handle associated with a certificate as specified in IEEE Std. 1609.2™ and containing permissions consistent with the access control policy.	The Cryptomaterial handle to be used for signatures in the handshake of the cryptographic secure session.
Certificate Permissions Pattern	An array of (PSID, SSP)	Any	The PSID and SSP pattern that must be matched by the peer's 1609.2 certificate presented in the handshake. A match of any entry in this array is acceptable. See IEEE Std. 1609.2™ for further details.
Inactivity Timeout	A time period	Any positive time period	How long any session associated with this secure session instance may be inactive before a fresh handshake is required.
Session Timeout	A time period	Any positive time period	How much time may have passed since the most recent handshake of a session associated with this secure session instance before a fresh handshake is required.
— The following parameter is provided only if Session Type is External and Role is Server and are required in that case.			
Require Client Authentication	Boolean	True, False	Whether to require client authentication during the TLS handshake. If this parameter is not provided, cie.
— The following parameters are provided only if Session Type is External and Role is Server and are optional in that case.			
Incoming Request Timeout	A time period	Any positive time period	The length of time after the Secure Session instance receives this primitive in which it will accept incoming connections. After this time has passed, the Secure Session instance will not accept incoming connections.