INTERNATIONAL STANDARD

ISO 21177

First edition
2023-04

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

Systèmes de transport intelligents — Services de sécurité des stations ITS pour l'établissement et l'authentification des sessions sécurisées entre dispositifs de confiance

Reference number
ISO 21177:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

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*, in collaboration with the European Committee for Standardization (CEN) Technical Committee CEN/TC 278, *Intelligent transport systems,* in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

This first edition of ISO 21177 cancels and replaces ISO/TS 21177:2019, which has been technically revised.

The main changes are as follows:

— change proposals presented in ISO/TR 21186-3:2021 have been incorporated, including:
  — CRL request functionality added;
  — session extension functionality added;
— editorial improvements to improve readability and clarity have been made, including:
  — revision of Figure 7, renumbered to Figure 8;
  — insertion of new Figure 7.

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 specifies ITS station security services that provide authenticity of the source and confidentiality and integrity of application activities taking place between trusted devices. The two devices taking part in a data exchange establish a cryptographically secure session; as part of establishing this session, each device [or, more precisely, each end entity (EE) which is an application on the device] is sent one or more digital certificates that are cryptographically bound to the other EE and contain statements, made by a trusted third party, about the EE's capabilities, properties and permissions. This allows each EE to have assurance about the properties of the other EE in the session, and this in turn allows each EE to make trust and access control decisions about data that the other EE can access, commands that the other EE can execute, states that the other EE can change, and other types of access that the other EE can request. In other words, the two EEs establish a trust relationship where each EE is trusted by the other EE to carry out specific actions, without requiring one EE to allow the other EE to have arbitrary access.

The mechanisms specified in this document allow each EE to establish trusted facts about the other EE. For these mechanisms to be used, the EE specification needs to include an access control policy, indicating which properties are required to be known to be true about the other EE for that other EE to be allowed to carry out particular actions. In other words, this document provides a means to obtain security-relevant information, but the use of that security-relevant information is to be specified in the specification of the EE.

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.

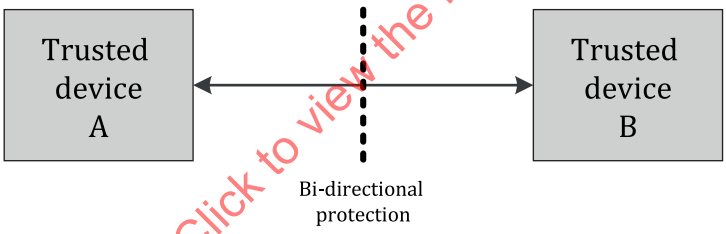


Figure 1 — Interconnection of trusted devices

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-SCUs) 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.

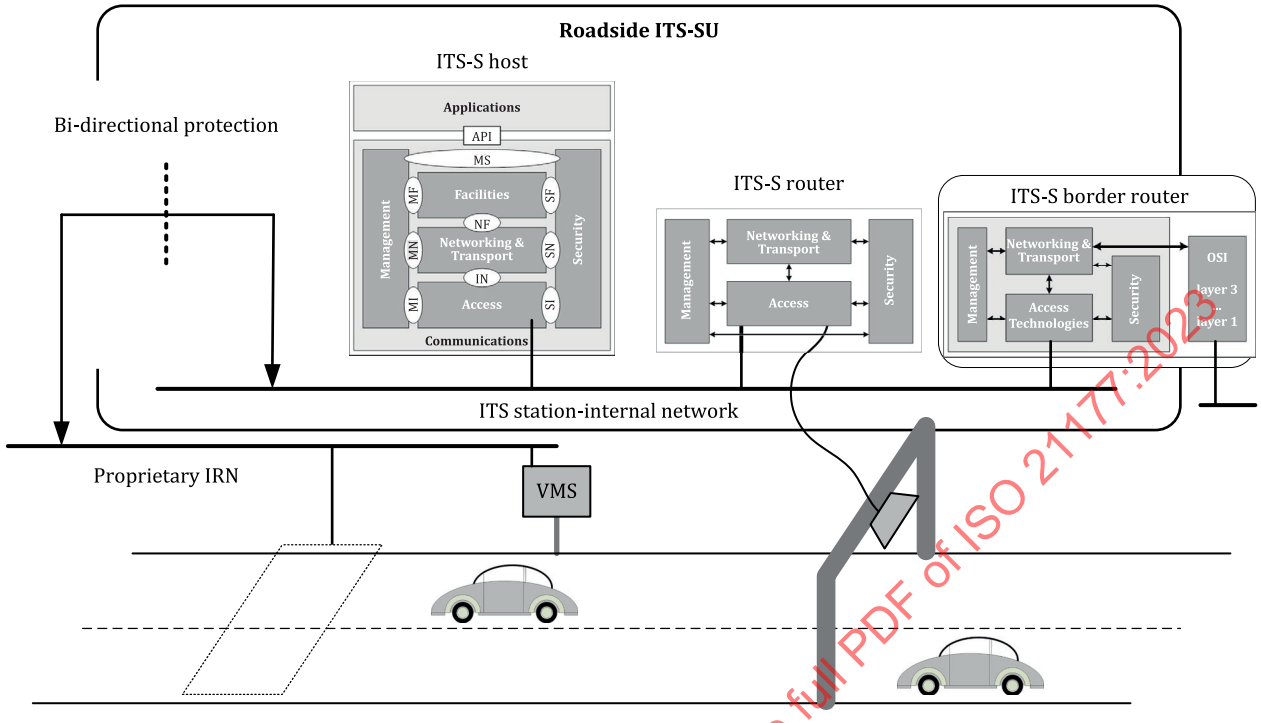
NOTE 1 ISO 21217 fully covers the functionality of EN 302 665,[16] which is a predecessor of ISO 21217.

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 and ISO 17419. Station-internal management communications between ITS-SCUs of the same ITS-SU are specified in ISO 24102-4. The 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 ITSstations.

Four implementation contexts of communication nodes in ITS communications networks are identified in the ITS station and communication architecture of ISO 21217, each comprised of ITS-SUs 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, for example, sustainability, road safety and transportation efficiency. See also Figure 2, Figure 3, Figure 4 and Figure 5.

Over the last decade, ITS services have arisen that require secure access to data from sensor and control networks (SCN), for example, from in-vehicle networks (IVN) and from infrastructure/roadside

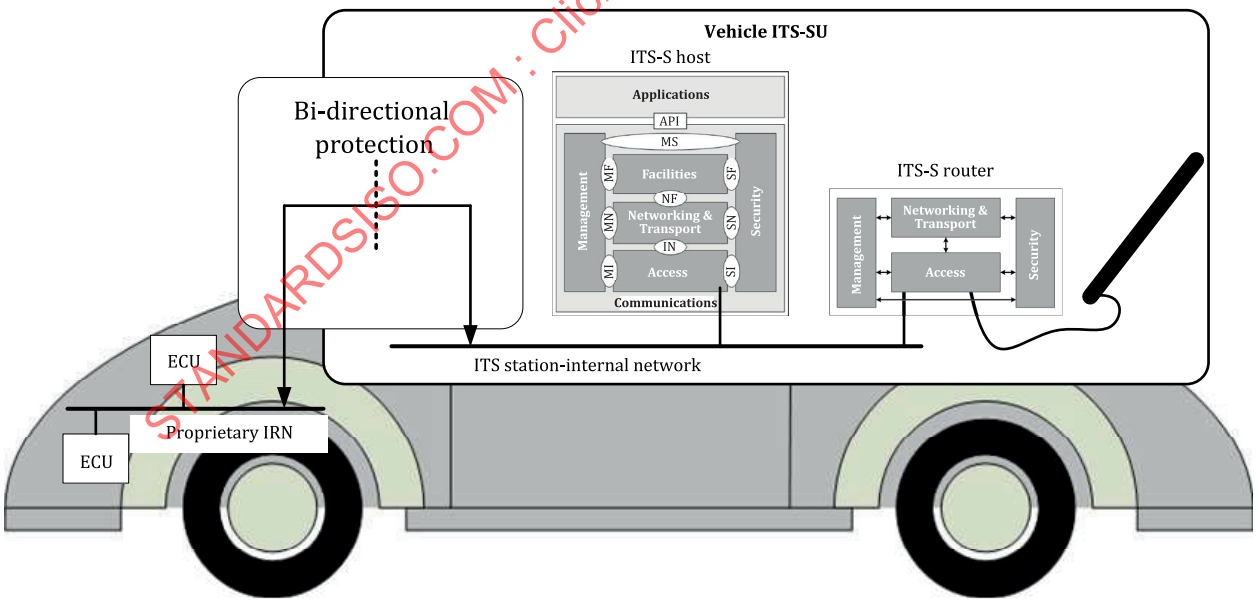
networks (IRN), some of which require secure local access to time-critical information; see Figure 2 and Figure 3.



Key

VMS Variable Message Sign

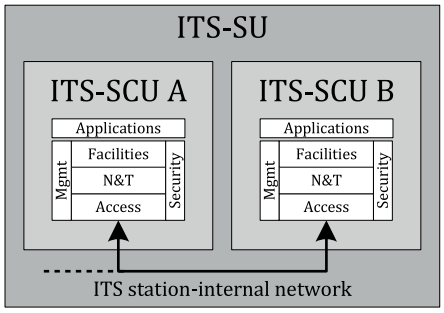
Figure 2 — Example of a roadside ITS-SU connected with proprietary IRN



Key

ECU Electronic control unit

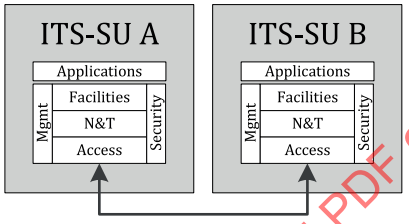
Figure 3 — Example of a vehicle ITS-SU connected with proprietary IVN



Key

N&T Networking & Transport

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



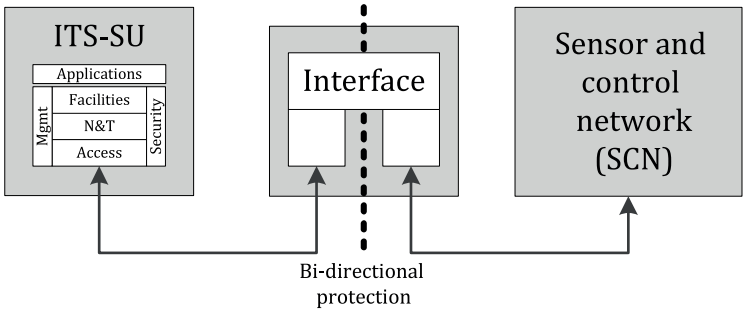
Key

N&T Networking & Transport

Figure 5 — Interconnection of ITS-SUs

By applying basic security means specified in this document, the ITS-SUs can establish secure application sessions. Establishment of sessions either requires prior knowledge about a session partner or can be achieved by means of a service announcement as specified in ISO 22418. Further on, the broadcasting of messages is secured by means of authenticating the sender of such a message, applicable for the service advertisement message (SAM) specified in ISO 16460 and used in ISO 22418. 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.



Key

N&T Networking & Transport

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 and dynamic priorities;

— protection of private data, e.g. in compliance with the European "General Data Protection Regulation" (GDPR);[18]

— local access to certified real-time data for sustainability applications, e.g. dynamic emission zones (controlled zones as currently standardized by CEN/TC 278), intersection priorities based on emissions, and 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 this number will grow (see the US National ITS Reference Architecture,[19] for example). It is critical that ultimately all ITS-SUs in a given area be 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 to be 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 need to 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) Specification of 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 prioritization and restricted access.

2) Specification of ITS station security services for enabling trust between ITS-S application processes running on the same ITS-SU.

3) Extension of these ITS security services for enabling trust between an ITS-SCU and devices being part of a sensor and control network.

Such security services include, for example, the basic security features of:

a) authentication and authorization;

b) confidentiality and privacy;

c) data integrity;

d) non-repudiation.

Tasks related to communications are:

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

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

g) encrypting messages.

NOTE 3 Tasks f) and g) above related to communications are already specified in other standards, see IEEE 1609.2 and several related standards from ETSI, for example.

The International Organization for Standardization (ISO) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO shall not be held responsible for identifying any or all such patent rights.

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, i.e.:

— between devices operated as bounded secured managed entities, i.e. "ITS Station Communication Units" (ITS-SCU) and "ITS station units" (ITS-SU) as 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 the authentication and secure session establishment which are required to exchange information in a trusted and secure manner.

These services are essential for many intelligent transport system (ITS) applications and services including time-critical safety applications, automated driving, remote management of ITS stations (ISO 24102-2), 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.

ETSI TS 102 941, *Intelligent Transport Systems (ITS); Security; Trust and privacy management*

ETSI TS 103 097, *Intelligent Transport Systems (ITS); Security; Security header and certificate formats*

IEEE 1609.2*including Amendment 1, IEEE Standard for Wireless Access in Vehicular Environments—Security Services for Applications and Management Messages*

IEEE 1003.1:2017 *IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7*

3 Terms and definitions

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

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

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

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

3.1
access control PDU
protocol data unit (PDU) generated by the security subsystem for purposes of establishing the authorization 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

Note 1 to entry: The natural language term "application" is used in this document as a synonym to the term "ITS-S application process" specified in ISO 21217.

3.4
cryptomaterial
cryptographic keys and associated material

Note 1 to entry: Cryptomaterial refers to either a secret key for a symmetric algorithm, or a private key for an asymmetric algorithm and the associated public key or certificate.

3.5
cryptomaterial handle
reference to cryptomaterial allowing that cryptomaterial to be used in cryptographic operations (sign, verify, encrypt, decrypt)

3.6
handshake ITS-AID
ITS application identifier (ITS-AID) that appeared in the SignedData HeaderInfo in the signed transport layer security (TLS) messages within the TLS handshake that started a secure session

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

3.8
security subsystem
functional entity providing security functionality for use by an ITS-S application process

3.9
security adaptor layer
functional entity providing multiplexing and demultiplexing functionality for data and session control commands

3.10
secure session service
functional entity providing confidentiality, integrity, authentication, guaranteed in-order delivery and replay protection on the datagrams that are passed over it

4 Abbreviated terms

ACK	acknowledge
ALPDU	adaptor layer protocol data unit
APDU	application protocol data unit
ASD	aftermarket safety device
ASN.1	abstract syntax notation 1
BSM	basic safety message

CA	certification authority
CAM	cooperative awareness message
C-ITS	cooperative intelligent transport systems
C-OER	canonical octet encoding rules
CRL	certificate revocation list
DENM	decentralized environmental notification message
DTLS	datagram transport layer security
ECU	electronic control unit
EE	end entity
FSAM	fast service advertisement message
ID	identifier
IRN	infrastructure/roadside network
ITS	intelligent transport systems
ITS-AID	ITS application identifier (PSID and ITS-AID share a common number space)
ITS-S	ITS station
ITS-SCU	ITS station communication unit
ITS-SU	ITS station unit
IVN	in-vehicle network
MAP	(digital) map (message)
NYC	New York city
OCSP	online certificate status protocol
OSI	open system interconnection
OTP	one-time password
PB-AKE	password based authenticated key exchange
PDU	protocol data unit
PIN	personal identification number
PSID	provider service identifier (PSID and ITS-AID share a common number space)
RSU	roadside unit
SAM	service advertisement message
SAP	service access point
SCN	sensor and control network

SDEE	secure data exchange entity
SPAKE2	secure password authenticated key exchange 2
SPaT	signal phase and timing (message)
SPDU	signed protocol data unit
SRM	service response message
SSP	service specific permission
SSTD	secure session between trusted devices
TCP	transmission control protocol
TIM	traveller information message
TLM	trust list manager
TLS	transport layer security
TMC	traffic management centre
VMS	variable message sign
WAVE	wireless access in vehicular environment
WSA	WAVE service announcement

5 Overview

5.1 General description, relationship to transport layer security (TLS) and relationship to application specifications

This document specifies a secure communications mechanism for secure sessions between ITS station units (ITS-SUs). The sessions are secure in that data within the session is protected against eavesdroppers (encryption) and protected against modification (integrity) and that the participants in the session know cryptographically-assured information about each other that can be used by one participant to determine which access privileges to grant to the other participant.

The secure communications sessions are based on transport layer security (TLS) 1.3, where digital certificates and secured protocol data units as specified in IEEE 1609.2 are incorporated into the TLS handshake as specified in RFC 8942.

TLS is a secure session protocol above the transport layer in the OSI model, and the endpoints of the ISO 21177 session are applications on the endpoint ITS-SUs. From the point of view of the applications, the use of ISO 21177 for secure communications is identical to the use of TLS over TCP and PDU handling for transport over the ISO 21177 session is the same as PDU handling for transport over a TLS session. The main differences between ISO 21177 and TLS are that ISO 21177:

— integrates the use of IEEE 1609.2 certificates, and

— provides mechanisms for one party to request additional certificates from the other party.

The mechanisms specified in this document allow each end entity (EE) to establish trusted facts about the other EE. These trusted facts are known as the authorization state of the other EE (in the context of that application session). However, the use of the authorization state is out of scope for this document, as this use will be specific to the specific application. The application specification itself is the appropriate home for a specification of the access control policy that maps known facts about

the other party in the session to permissions of that other party. The application specification is also the home for specifications of customizations of the IEEE 1609.2 certificates used by instances of that application, via the use of the service specific permissions (SSP) field associated with the ITS-AID in the certificate. The approach anticipated is that:

— the SSP specification for the application will specify semantics of the SSP (for example, "if bit 1 is set the sender can take action A", or "if bit 1 is set the sender is a police car", or "if bit 1 is set the sender is blue");

— the access control policy will map from the semantics of the SSP to permitted activities (for example, "if the SSP says the holder can take action A, then the holder can take action A", or "if the SSP says the holder is a police car, then the holder can take action B", or "if the SSP says the holder is blue, then the holder can take action C".

The access control policy may also key off other properties in the certificate, for example in the certificateId field. It may also key off other information, such as whether the party granting access is in motion or at rest. All of this is to be included in the application specification for that application.

5.2 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.

5.3 Architecture and functional entities

The protocols specified in this document are for secure communications between two peer applications on different ITS-Ss. The applied model is that each application on an ITS-S has access to resources, some of which are specific to that application, and some of which are shared with (i.e. can be accessed and potentially modified by) other processes on the ITS-S. The two application instances on different ITS-Ss that communicate using thee protocol specified in this document are referred to as the home and the peer application instances. The secure communications between the peer and the home application

instances can be considered a series of requests by the peer application to access the local resources: for example, to read a value, or to update the local state, or to carry out a financial transaction by accessing and modifying financial information and metadata stored by the home application. The goal of the protocols is to give the home application instance the information it needs about the peer application instance to make a decision as to whether or not to allow the requested access to the local resources.

This document uses the following definitions; see also Clause 3:

— **Resources**: These are endpoints of ITS-S application process activity and may be inside or outside the ITS-SU or ITS-SCU 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 To simplify reading of this document, the term "application" is used as a replacement for "ITS-S application process", where appropriate.

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

— 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.

The description in this document refers in general to activities on the “home trusted entity" and the “peer trusted entity". Details of functional blocks in an ITS-S are shown in Figure 8.

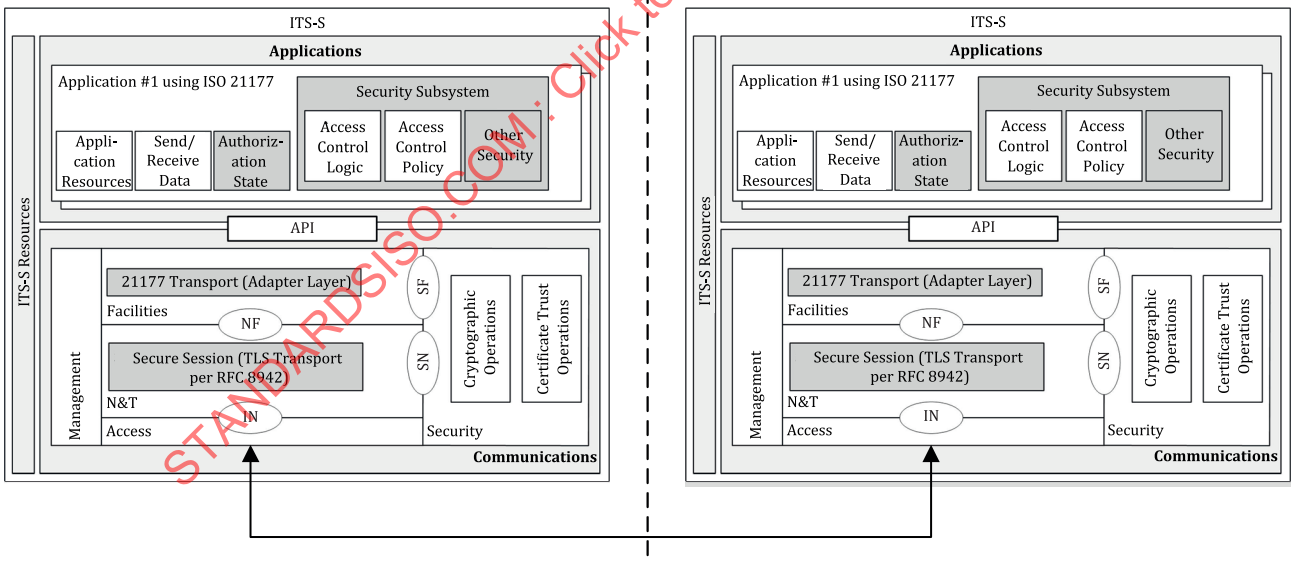


Figure 7 — Logical Architecture with two ITS-SUs

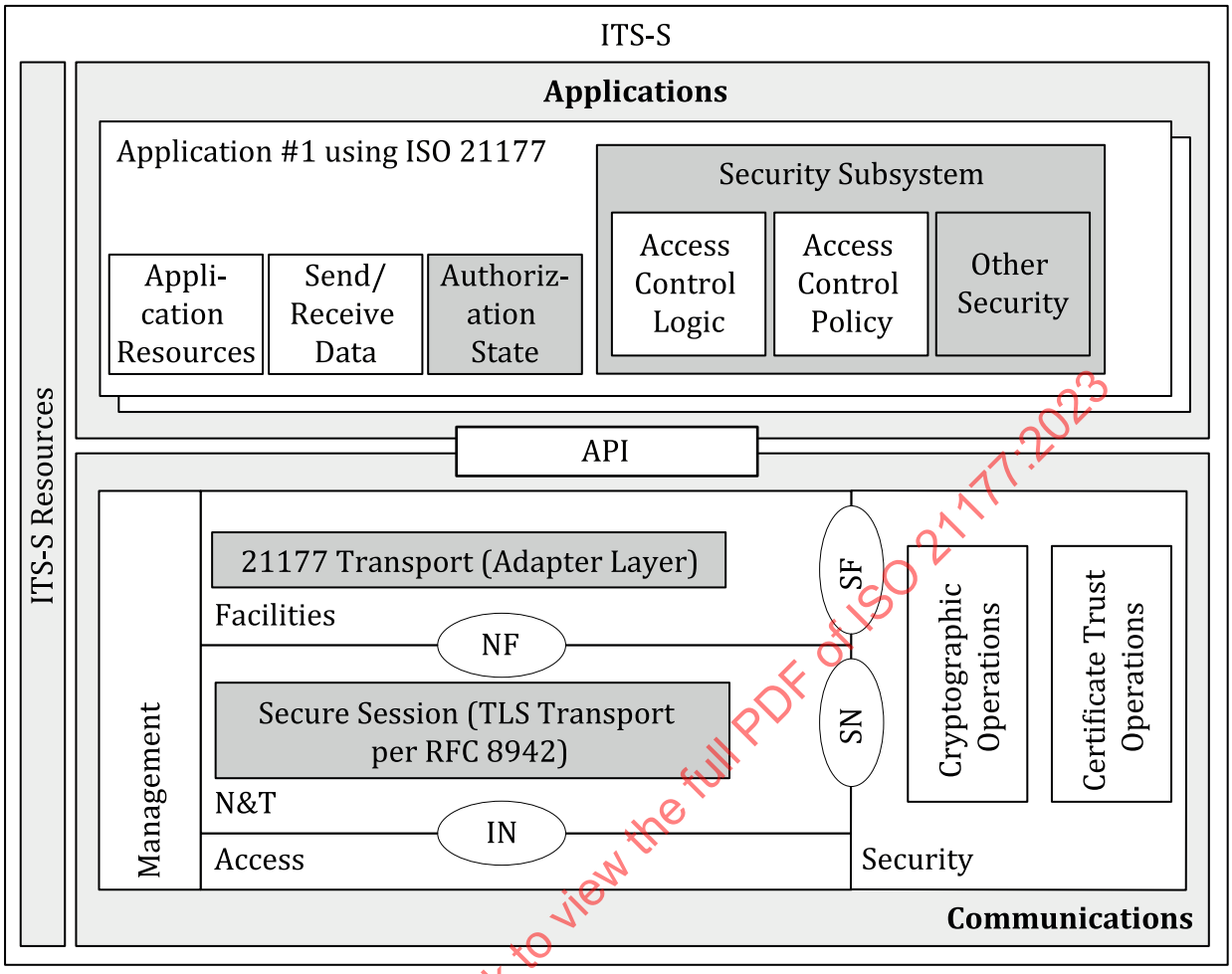


Figure 8 — Logical Architecture within the ITS-S architecture

Figure 8 shows the allocation of functional entities involved in communicating using the mechanisms of this document in the ITS station architecture specified in ISO 21217. The functional entities shaded in grey are the subject of this document. It is to be noted that several applications may use simultaneously the functionalities specified in this document; to simplify the graphical presentation, only one such application is fully presented.

The ITS-S resources are those resources of an ITS-SU that the home application can access and that are also accessed by other applications or processes on the same ITS-SU. Note that the term ITS-SU indicates a physical implementation of the ITS-S functionality; see ISO 21217.

There may also be application-specific resources that are not necessarily shared with other applications.

The functionalities "21177 Transport" and "Secure Session" specified in this document allow secure communication of the authorization properties of the peer application instances

— The functionality of "Secure Sessions (TLS Transport as per RFC 8942)" is located in the ITS-S networking & transport layer.

— The functionality "21177 Transport (Adapter Layer) is located in the ITS-S facilities layer.

The functionality "Authorization State" stores those authorization properties.

— The functionality "Authorization State" is located inside the application using the functionalities of this document. An equally suited alternative location would be in the IST-S facilities layer; there serving all applications that use the functionalities of this document.

Note that for an implementation of the functionality of this document, the location of functionalities in the architecture is not at all important, and thus, the presented location is only for illustration purposes.

The functional entities have the following responsibilities:

— **Authorization State**: Stores authorization information about the peer entity as well as metadata such as when the authorization information was received.

— **ISO 21177 Transport (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 Sessions (TLS Transport as per RFC 8942)**: 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 document the only supported cryptographic secure session mechanism is TLS 1.3 with certificates specified in IEEE 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 components of the application are the application resources, the “Send/Receive Data” functional entity, and the functional entities within the “Security subsystem” component of the application. These functional entities play the following roles:

— **Application Resources**: resources accessed only by that application instance.

— **Send/Receive Data**: Functionality that sends and receives data.

— **Security Subsystem**: Security functionality that includes the following functional entities:

  — **Access Control Logic**: Functionality that determines whether a particular resource access request by the peer application instance will be granted.

  — **Access Control Policy**: The specification of the criteria used by access control Logic to make an access decision.

  — **Other Security**: Other security operations, for example signing or verifying individual PDUs.

— **ITS-S Resources**: general resources of an ITS station unit accessible by different applications.

The relationships between instances of these functional elements and data sources on a single trusted entity are shown in Figure 9. In Figure 9, 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.

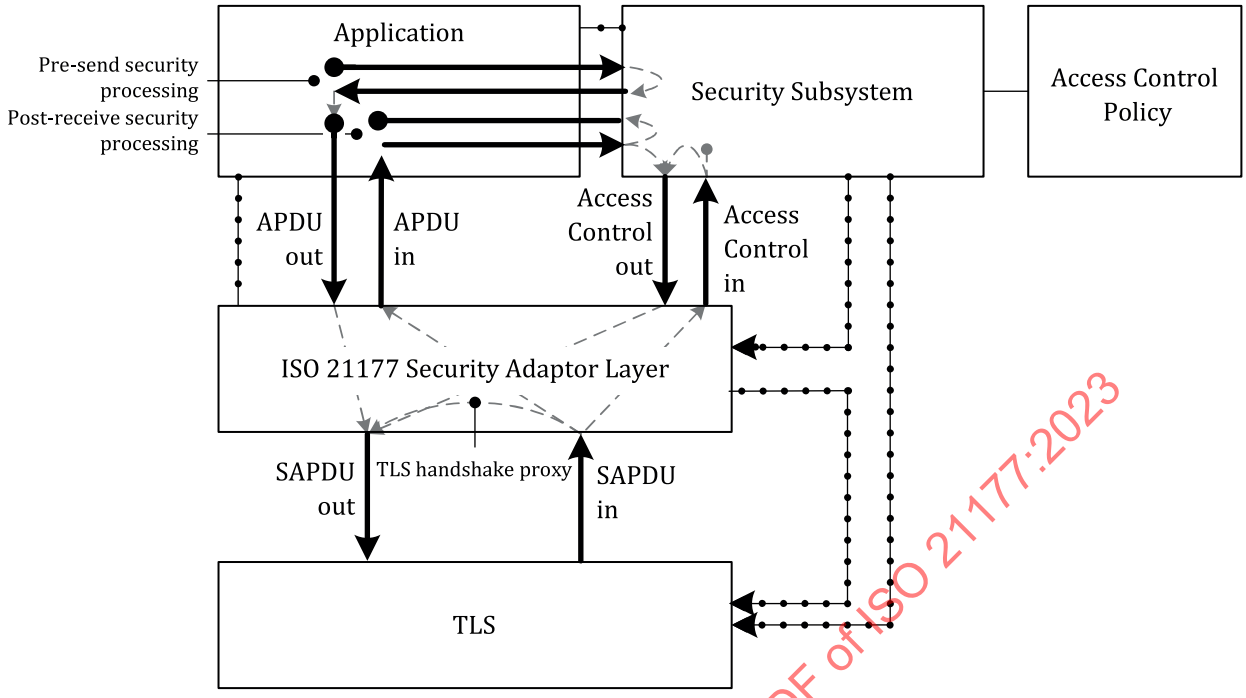


Figure 9 — Interactions between local functional elements

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

— The **application-security subsystem interface** is specified in 7.7. It is used:

  — optionally, by the application, to request IEEE 1609.2 application-level processing for outgoing APDUs specified in IEEE 1609.2 as described in 6.6;

  — by the application, to request that access control services are applied to activities that can potenially 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 can result in the security subsystem updating its state or in the security subsystem generating 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 **ISO 21177 Security 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 6.10;

  — 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 for TLS brokering as specified in 6.10.4.

— 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 9.3. 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.

5.4 Cryptomaterial handles

This document uses the concept of cryptomaterial handles specified in IEEE 1609.2 to represent logical access to cryptographic keys and, if appropriate, their associated certificates. See IEEE 1609.2 for more details.

5.5 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 specified 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.

5.6 Access control and authorization state

The security subsystem maintains the authorization state of the session. The authorization state is the collection of all “authorization 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 "authorization statement" is any PDU sent by the peer party that contains cryptographically protected information about the authorization of that peer party. Specifically, in this document, the authorization statements presented in Table 1 are specified.

Table 1 — Authorization statements

Type	Description	Related sub-clauses	Service primitives for provision to security subsystem
Initial authorization	The credentials presented during the handshake of the secure session, if one occurs	6.5	`Sec-Sess-StartSession.indication`
Enhanced authorization	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 7.6.12	6.7, 6.8,7.3	`Sec-AL-AccessControl.indication`
Extended authorization	Additional credentials presented during the secure session via an access control PDU of type `ExtendedAuthPdu` as specified in 7.6.6.	6.7, 6.8, 7.4	`Sec-AL-AccessControl.indication`

This document specifies access control request PDUs and access control response PDUs in 7.6.

An access control PDU containing an extended authorization 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 authorization 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 authorization 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.

5.7 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.

5.8 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 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.

Service primitives are not testable but indicate the information that one functional entity is obliged to 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, in this document, a message from a higher layer to a lower layer 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 to a higher layer is specified with an:

— *XXX*`.indication` service primitive,

and the response from the layer entity to the lower layer 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. The list of parameters of a service primitive function does not contain any information allowing to map `.confirm` service primitives on respective `.request` service primitive, and `.indication` service primitives on respective `.response` service primitive; such mapping parameters are specified in ISO 24102-3 for the service primitives. Thus, this document parameterizes only service primitive functions.

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.

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

Owner authorization shall be as specified in Annex D.

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.

— Extend Session, see 6.9: Mechanism for one party to indicate to the other that it has a new certificate with which it wishes the session to be associated.

— Secure connection brokering, see 6.10: 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 6.11: 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 6.12.

— Deactivate, see 6.13: 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 6.14.

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 intended 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 10, the message contains an APDU and is sent using the primitive `App-Sec-Data.request`, see 7.7.4.

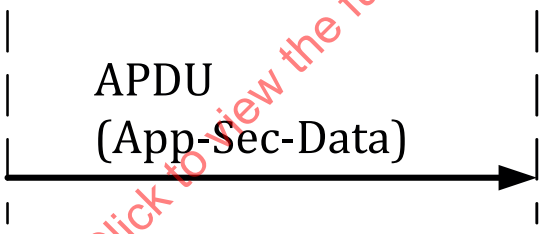


Figure 10 — Convention for sequence diagram - 1

— 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 11, the message configure contains only local information and is sent using the `App-Sec-Configure.request` service primitive.

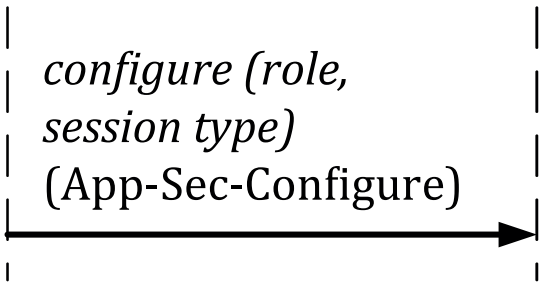


Figure 11 — Convention for sequence diagram - 2

— Mapping to primitives: See 5.8 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. At the end of this process flow, if the security subsystem has been configured to take the client role in the secure session, the security subsystem initiates the handshake for the secure session. If the security subsystem has been configured to take the server role, it awaits incoming connections.

A prerequisite for this step is that the application has a socket identifier, i.e. an identifier for the communications session to which security is to be applied. This document does not specify socket functionality other than to note that all functional entities shall interpret a socket identifier consistently.

The process steps for this process flow are as follows.

a) The application uses the `App-Sec-Configure.request` primitive (see 7.7.1) to inform the security subsystem of:

   1) the role it has in the secure session (i.e. 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 edition of this document, three secure session types are permitted:

      i) TLS for reliable transport mechanisms;

      ii) DTLS for unreliable transport mechanisms;

      iii) 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 7.7.2, (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:

         a) role,

         b) cryptomaterial handle,

c) socket type;

ii) from the access control policy:

a) the permissions constraints to apply to incoming IEEE 1609.2 certificates, i.e. the acceptable ITS-AID and SSP values,

b) the SDEE Identifier used to identify the application to the IEEE 1609.2 secure data service, as specified in IEEE 1609.2,

c) 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;

If the parameter *role* is "Client", the instance attempts to carry out the handshake.

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 12.

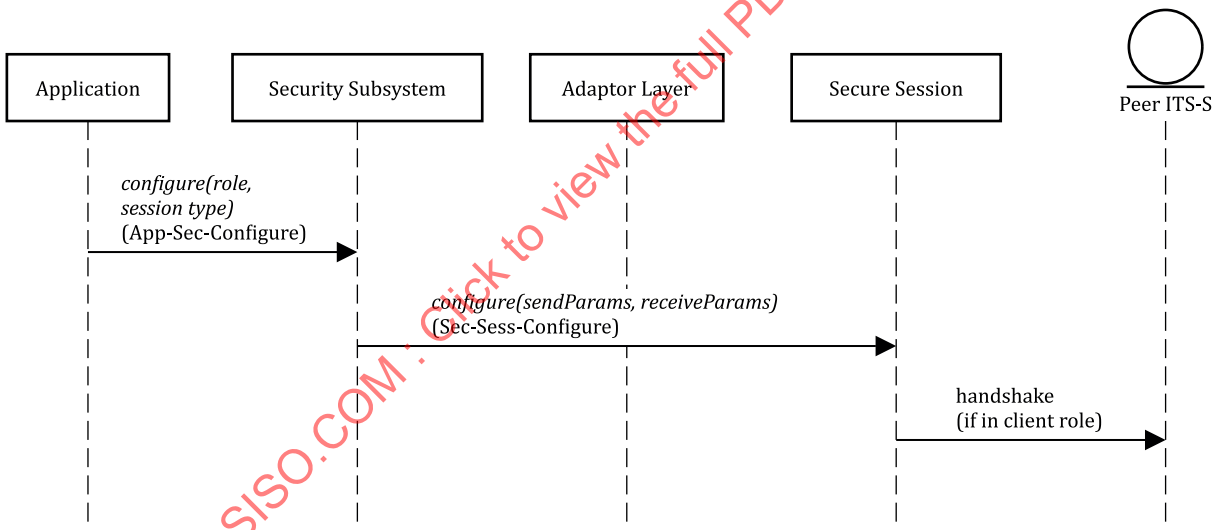


Figure 12 — Sequence diagram for Configure process flow

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 6.13 and Configure a new secure session instance using the mechanisms of this subclause.

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 is required to 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 specified in 5.3.

— The secure session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session, a handshake has taken place (initiated by the client in the secure session as the final step in "Configure").

— 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 as follows.

a) The local secure session service uses the `Sec-Sess-Start.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 (in the case of a cryptographic secure session);

   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 is 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. (Note that the security subsystem will only be notified of incoming connection if the certificate in the handshake has met the conditions provided in the Configure step, and so the security subsystem has assurance that the certificate meets those conditions; as such, it is permissible for there to be no additional access control processing at this step). The status may be:

   1) Success;

   2) Unauthorized, communication rejected. In this case the security subsystem ends the secure session as specified in 6.12;

   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 7.6.5,

      ii) in the case where failure is bing communicated to the peer, this is an access control PDU of type `AccessControlResult.failure`, as specified in 7.6.5,

      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 this as a datagram 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 7.7.3) to provide the session ID obtained in step a.3) above to the application. (If the secure session services are acting in the client role, the successful initiation is indicated by the return code from Configure).

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 13.

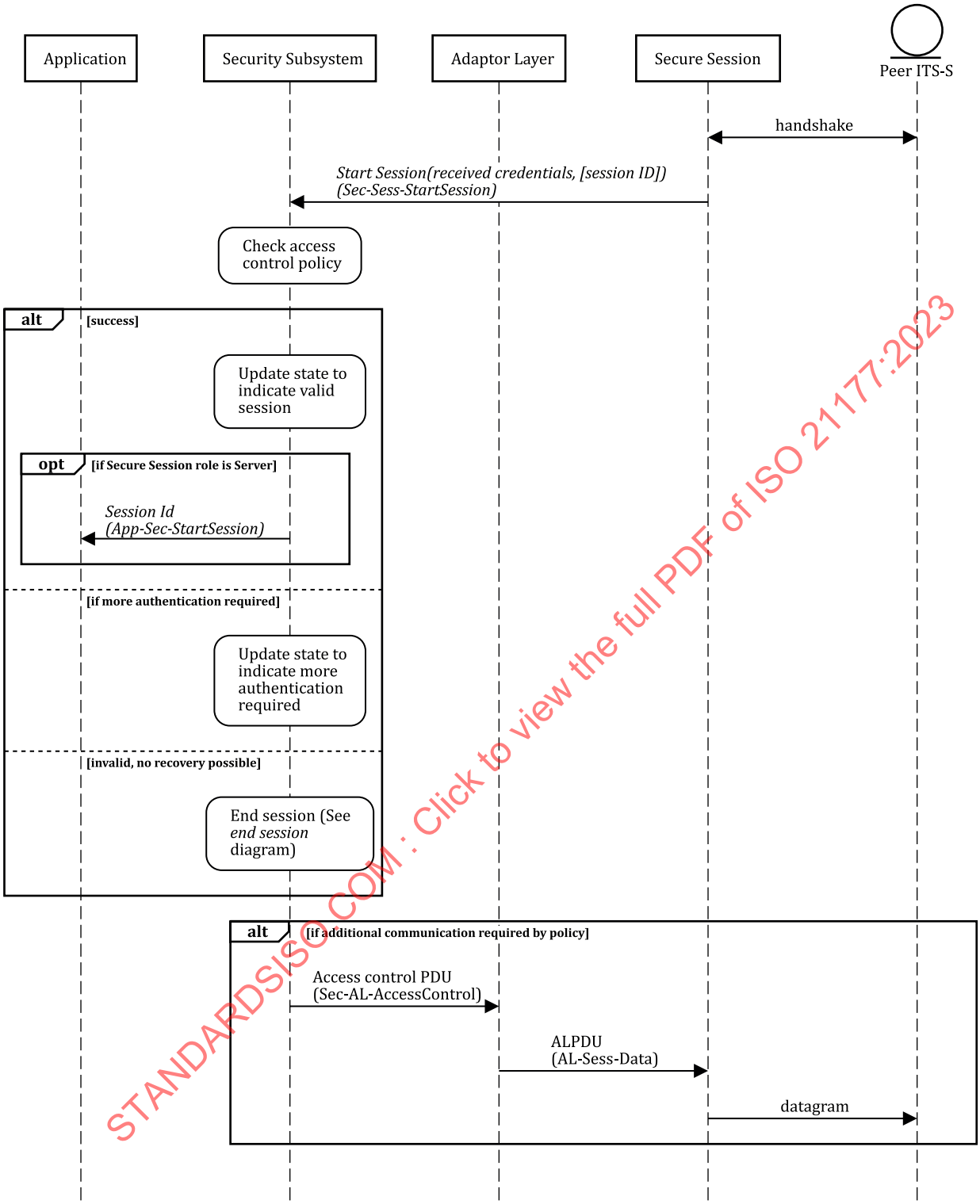


Figure 13 — Sequence diagram for Start Session

6.6 Send data

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

Prequisites:

— The secure session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as specified in 5.3, 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. If data is a stream rather than individual PDUs, the application determines how to split the stream into PDUs for submission to the secure session in the same way as is done for TLS.

The process steps for this process flow are as follows.

a) The APDU is encapsulated as an Ieee1609Dot2Data in a way that depends on whether 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 7.7.4, to send the APDU to the security subsystem for signing. The security subsystem returns the signed APDU, which is an `Ieee1609Dot2Data` of type signed as specified in IEEE 1609.2 (or, if appropriate, an error message) via `App-Sec-Data.confirm`, see 7.7.5;

   2) otherwise, the application creates an Ieee1609Dot2Data as specified 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 7.7.5;

   2) adds the session non-repudiation service if so configured. This service is not specified in this edition of this document but is planned for future revisions;

   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 14.

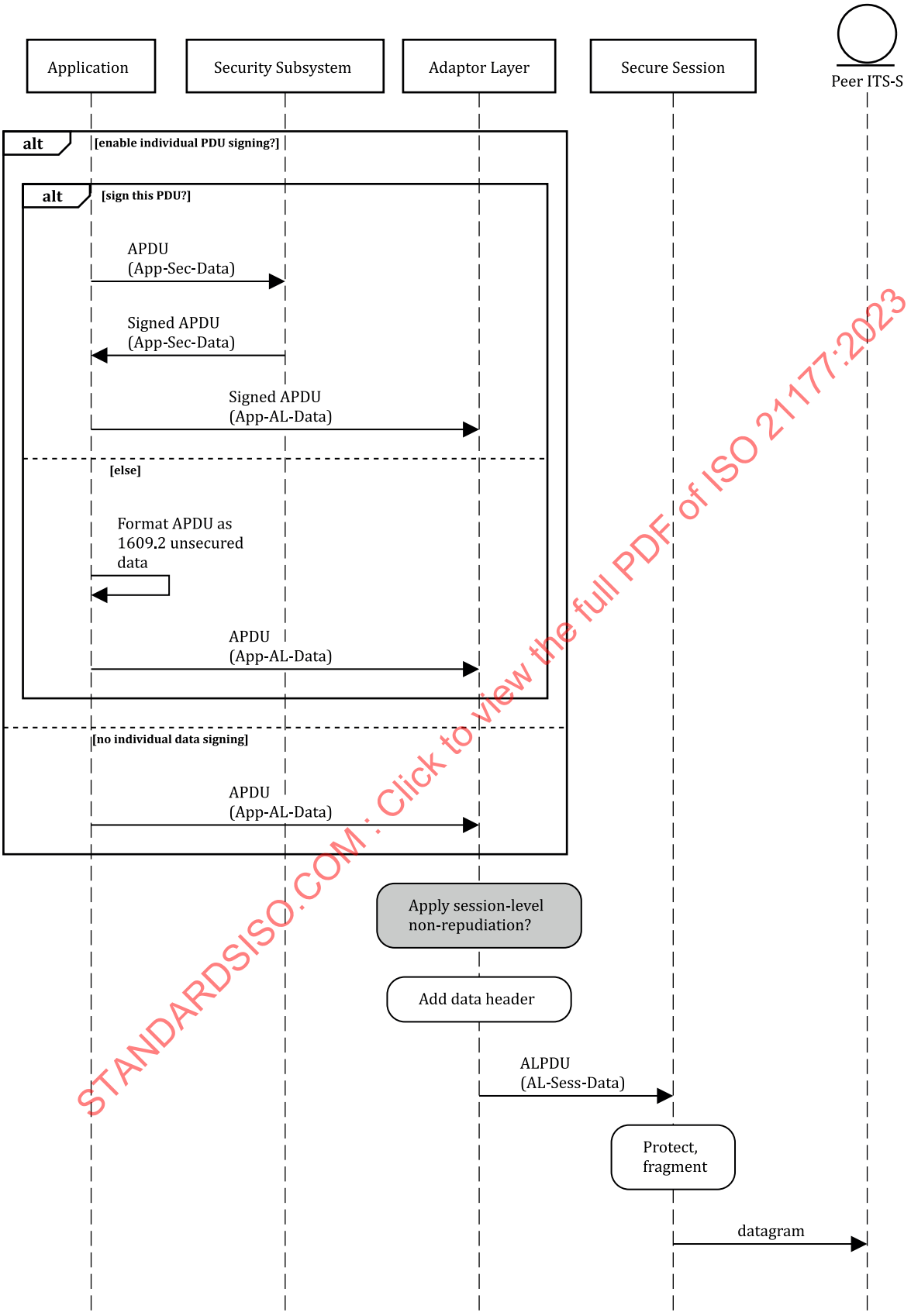


Figure 14 — Sequence diagram for sending data

6.7 Send access control PDU

In this process flow the security subsystem creates an access control PDU to update the authorization state on the peer security subsystem, i.e. to update the peer security subsystem's knowledge of the permissions and other authenticated properties of the home application. The access control PDU can be created as result of a local access policy decision, or as 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 subclause describes sending activities. The corresponding receive-side sequence is specified in 6.8.

Prequisites:

— The secure session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as specified in 5.3, the session has been started using the methods of 6.5.

The process steps for this process flow are as follows.

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 can be individually signed or not. In this edition 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 15.

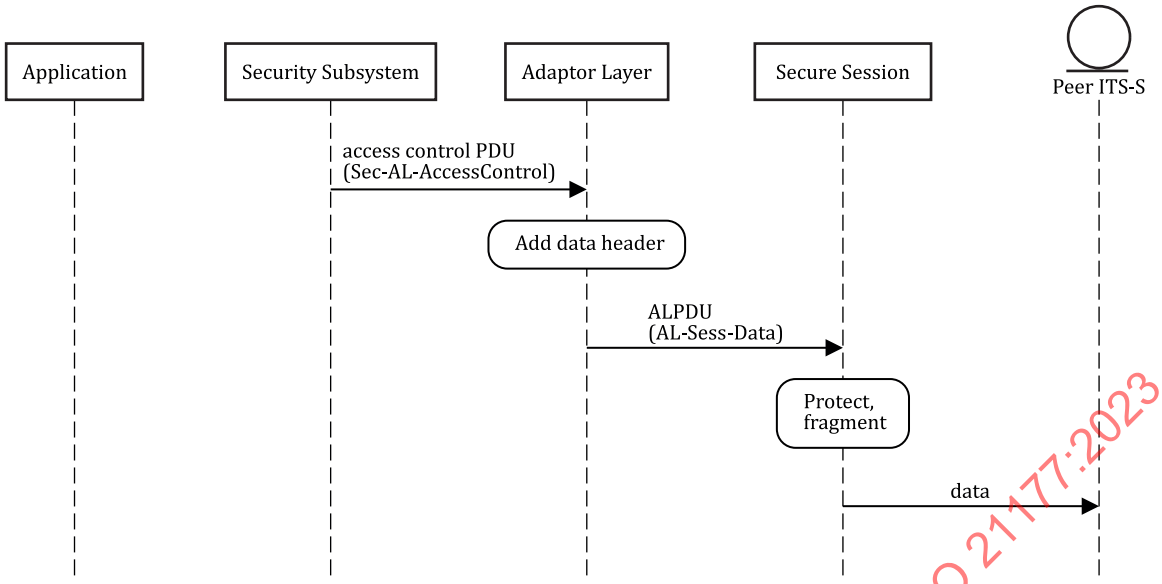


Figure 15 — Sequence diagram for sending access control PDU

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 subclause describes receiving activities. The send-side sequences are specified in 6.6 and 6.7.

Prequisites:

— The secure session service has been configured using the methods of 6.4.

— If the secure session is a cryptographic secure session as specified in 5.3, the session has been started using the methods of 6.5.

The process steps for this process flow are as follows.

a) The secure session service receives a datagram from the peer trusted entity and checks that the session has not timed out as per the parameters set in b.3.ii.c) 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 received 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 edition of this document but is planned for future revisions;

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 6.10.4.

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 5.6 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:

a) generate and send an access control PDU in response as specified in 6.7;

b) 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 and the content of the APDU 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:

a) 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;

b) 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 7.7.6. 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.

a) If the APDU is an `Ieee1609Dot2Data` of type signed, the security subsystem attempts to verify it as per IEEE 1609.2.

If verification fails, the security subsystem returns that indication via `App-Sec-Incoming.confirm` (see 7.7.7) and the process flow ends.

If verification succeeds, processing continues.

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

c) 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 of the access control decision process is one of the following:

a) the APDU is not acceptable as 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;

b) the APDU is acceptable as 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 16 (overview), Figure 17 (APDUs detail), Figure 18 (Access control PDUs detail).

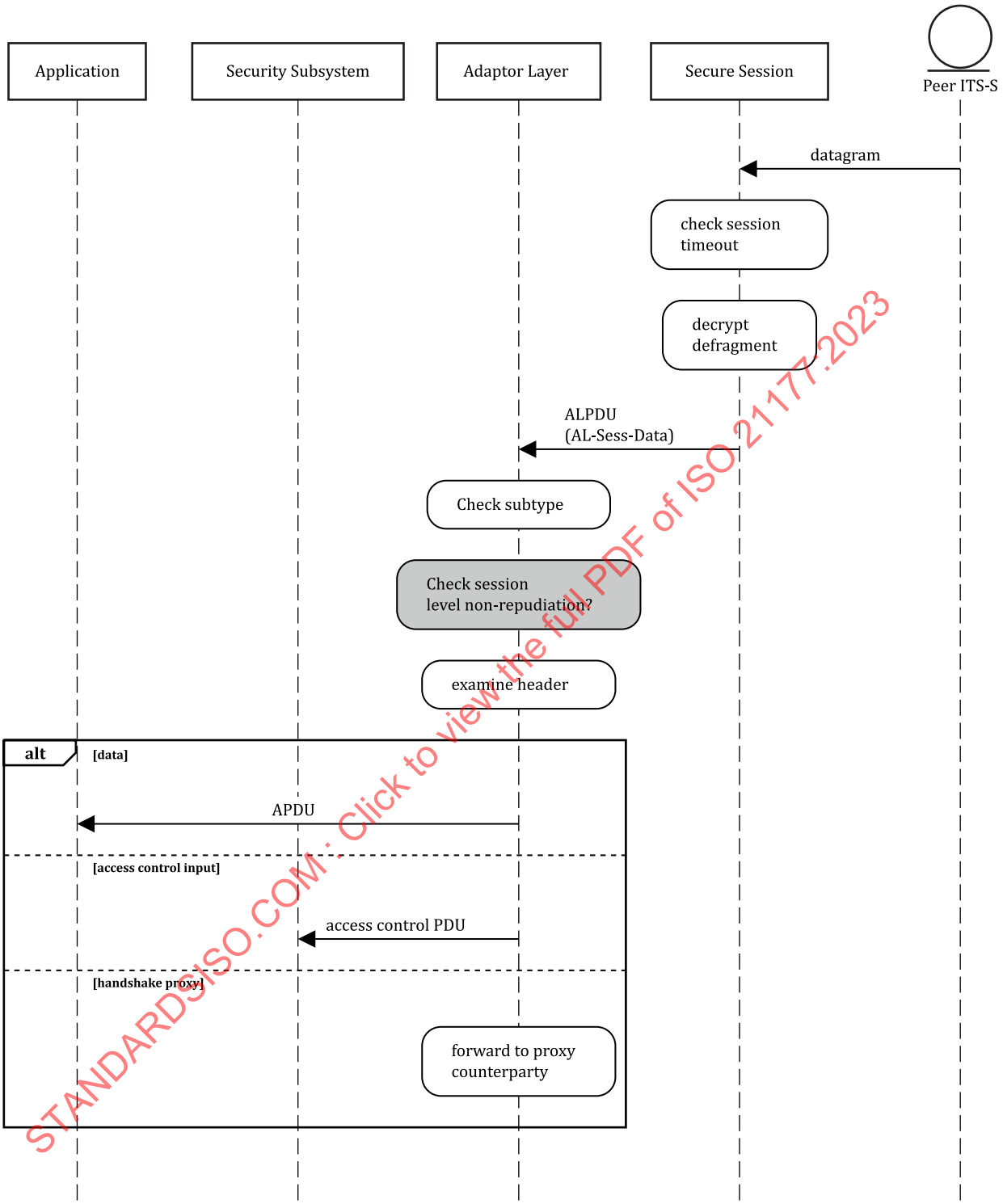


Figure 16 — Sequence diagram for incoming datagrams (overview)

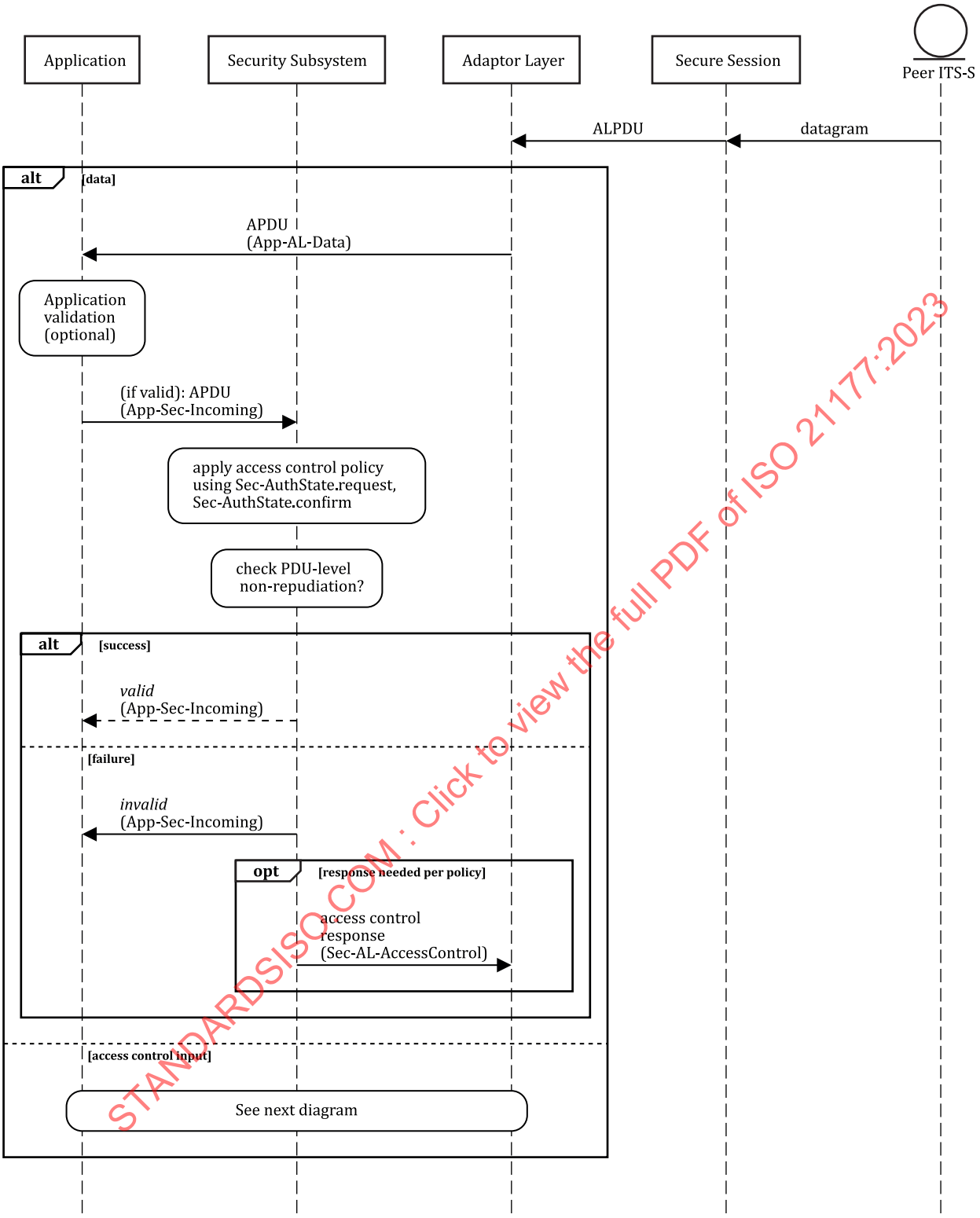


Figure 17 — Sequence diagram for handling of incoming APDUs

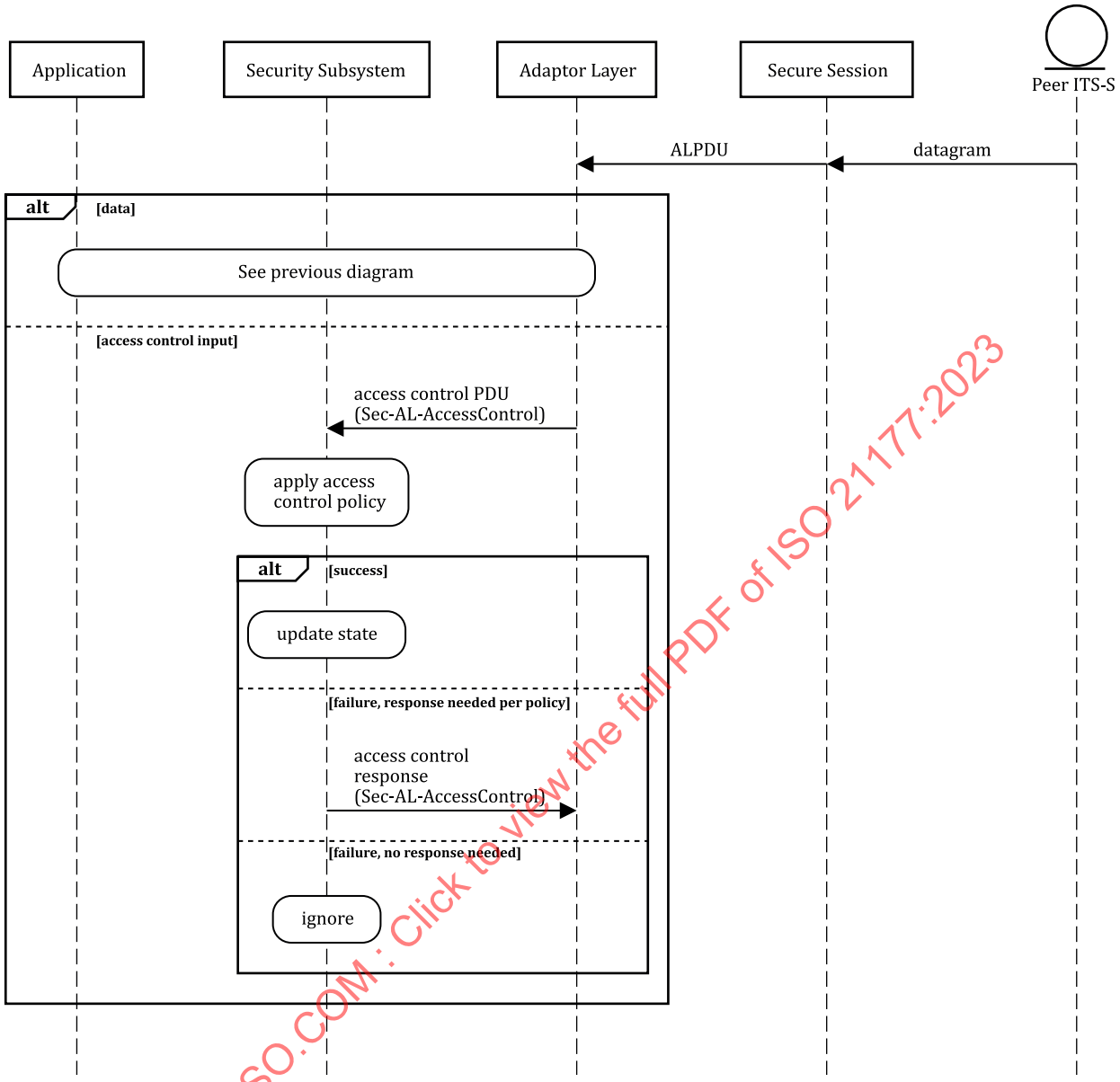


Figure 18 — Sequence diagram for handling of incoming access control PDUs

6.9 Extend session

6.9.1 Goals

A secure session is based on certificates presented in the TLS handshake (see Reference [16] for more details). The definition in Reference [16] recommends, but does not require, that a session expires when one of the certificates in the session expires. This can be inconvenient for a long-lived association between two devices, e.g. a traffic signal controller and an RSU. This subclause proposes a mechanism for one party to indicate to the other that it has a new certificate with which it wishes the session to be associated.

The intent of the mechanism is that a party to a communications session can provide a signed SPDU, signed by a currently valid certificate, indicating that a new certificate is a successor certificate to the original certificate. In this design, the signed SPDU includes a pduFunctionalType indicator as specified in IEEE 1609.2b with a new value of sessionExtension (i.e. the PduFunctionalType data type of IEEE 1609.2b is to be extended under this design).

The mechanism presented in this document is not the only possible approach. In particular, an approach based on the native TLS mechanism known as exported authenticators can be effective. However, to use an approach based on exported authenticators requires additional work within IETF to adapt exported authenticators for IEEE 1609.2 certificates. The approach presented in 6.9.2 is also more consistent with the IEEE 1609.2 security model.

6.9.2 Processing

Either party to a secure session can send a SessionExtensionPdu at any time before the expiry time of the signing certificate that was sent in the handshake to the session. The SessionExtensionPdu is sent without being requested. A SessionExtensionPdu contains nested SPDUs. The inner one contains the proposed new certificate, signed by the old certificate. The outer one contains the inner one, signed by the new certificate. The timestamp in the HeaderInfo is used to prevent replay attacks.

In more detail, when Party B receives a SessionExtensionPdu from Party A, the following steps occur.

a) Party B checks that SessionExtensionPdus are permitted under the access control policy. If they are, Party B continues. If not, Party B aborts.

b) Party B verifies the two signatures in the SessionExtensionPdu as per IEEE 1609.2, including a check that the generation time is within a tolerance threshold of the current time. The tolerance threshold is to be specified for each specific ITS-AID. If the checks pass, Party B continues. If not, Party B aborts.

c) Party B sends a SessionExtensionPdu with type identified by SessionExtensionAckId as specified in 7.6.26.

d) Party B extends the lifetime of the session to the minimum of:

   1) the maximum extension permitted by the access control policy;

   2) the expiry time of the new certificate from Party A;

   3) the expiry time of Party B's current certificate.

6.10 Secure connection brokering

6.10.1 Goals

"Secure connection brokering" is the process by which two applications, each fronted by a broker, can 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.

6.10.2 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, as 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, as per the secure session requirements of Clause 9.

— The server broker and client broker are also securely connected, as per the secure session requirements of Clause 9.

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

  — the fronted client has demonstrated to the client broker that it is authorized 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 authorized to act as a client broker;

  — the server broker has demonstrated to the fronted server and the client broker that is it authorized to act as a server broker;

  — the fronted server has demonstrated to the server broker that it is authorized 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.

6.10.3 Overview

An overview of the process, omitting some details and primitive invocations, is shown in Figure 19. 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 (respectively 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.

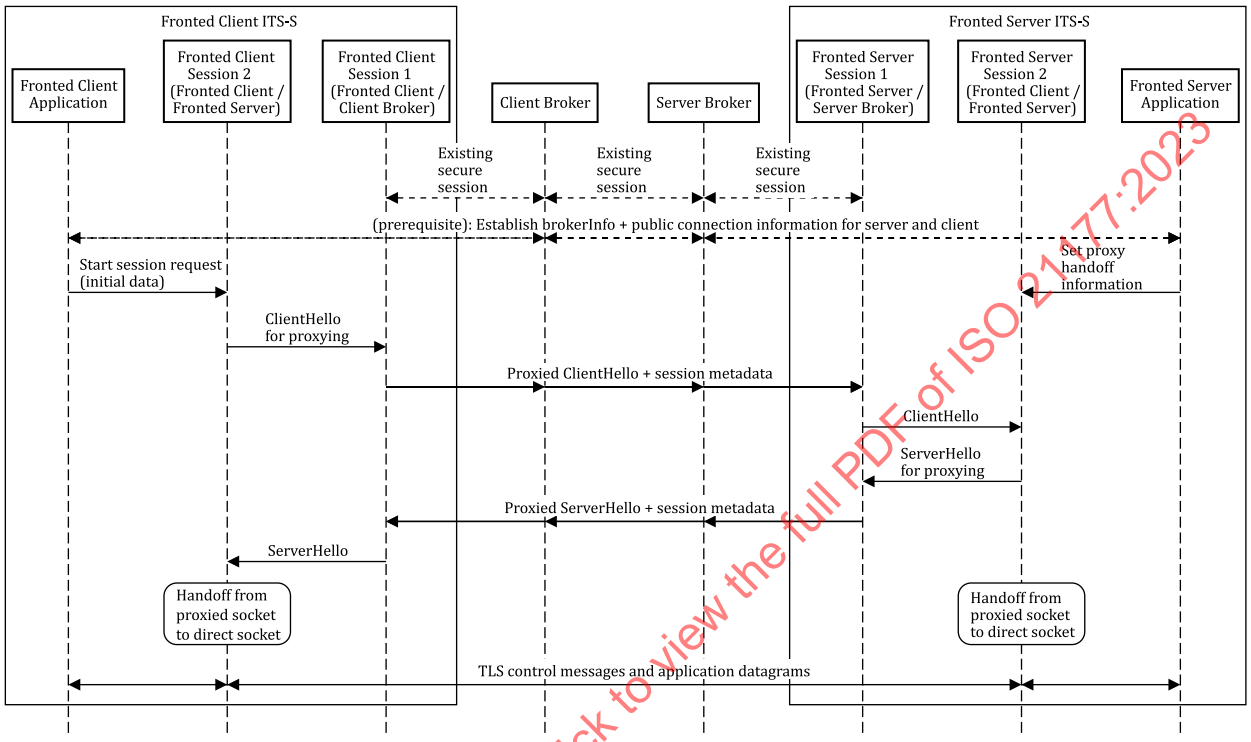


Figure 19 — Overview sequence diagram for secure connection brokering

6.10.4 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 7.7.1, 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 ClientHello 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 Figure 20, Figure 21, Figure 22 and Figure 23, 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 ClientHello 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 client 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) as 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 ServerHello 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.

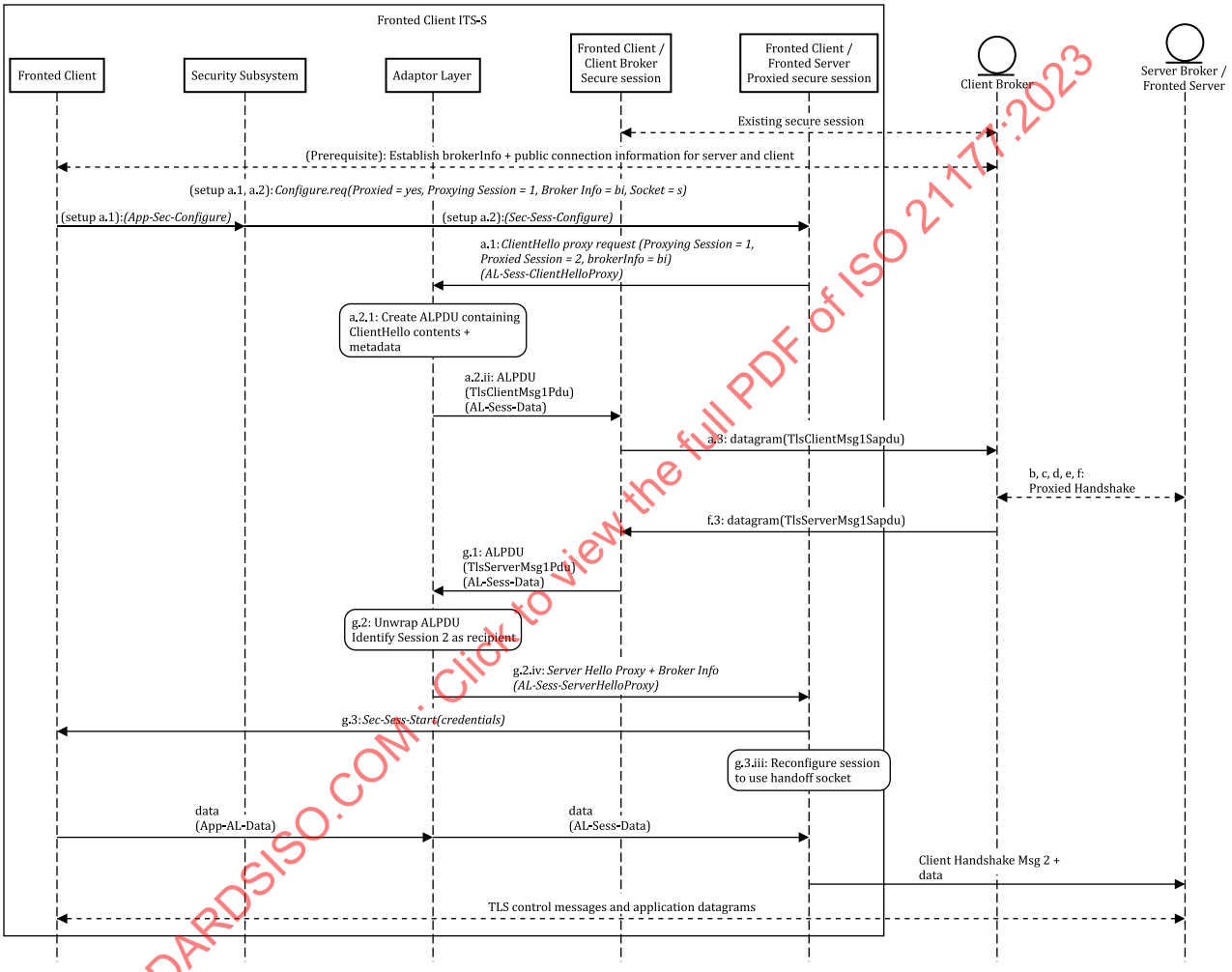


Figure 20 — Sequence diagram for secure connection brokering operations on the fronted client

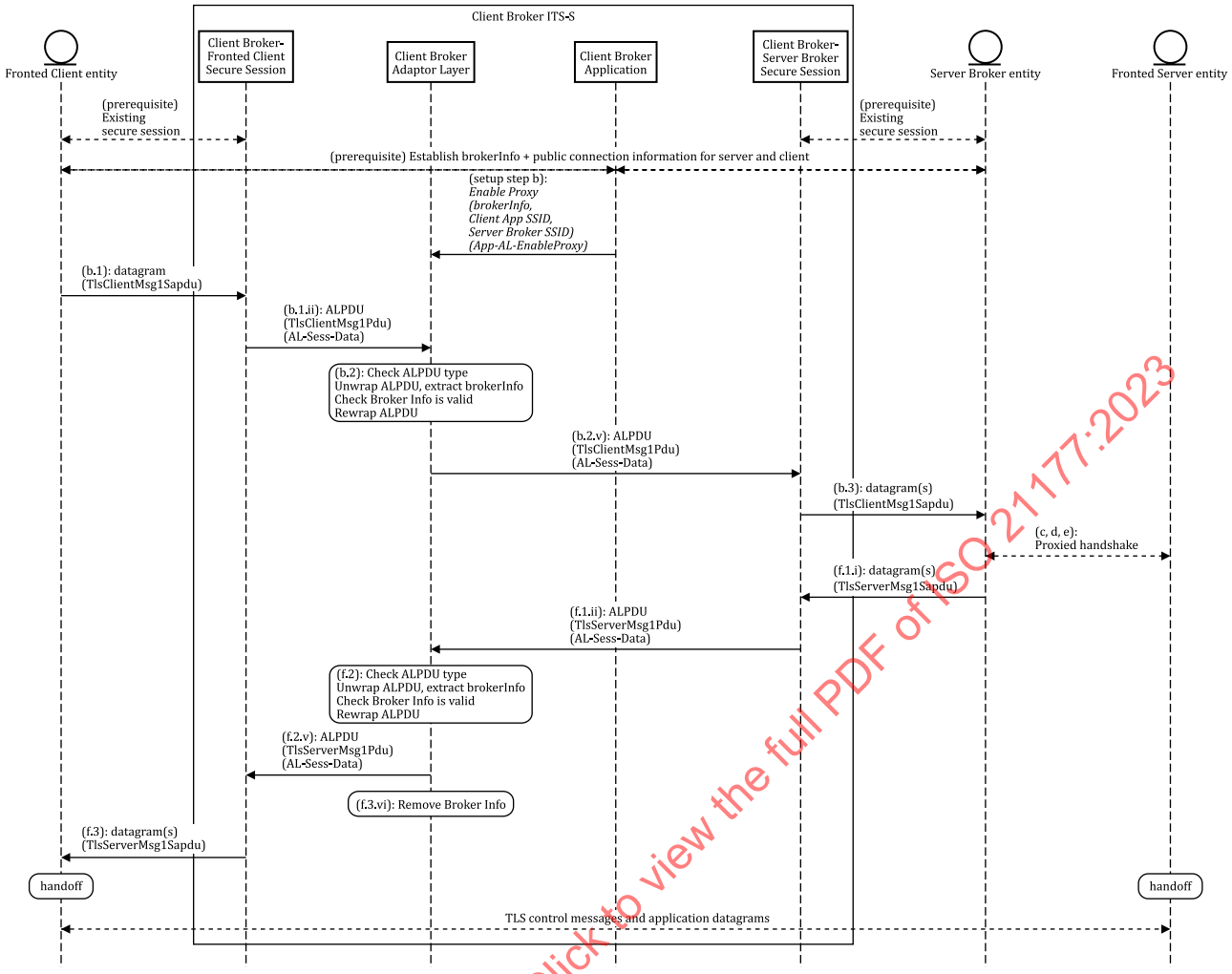


Figure 21 — Sequence diagram for secure connection brokering operations on the client broker

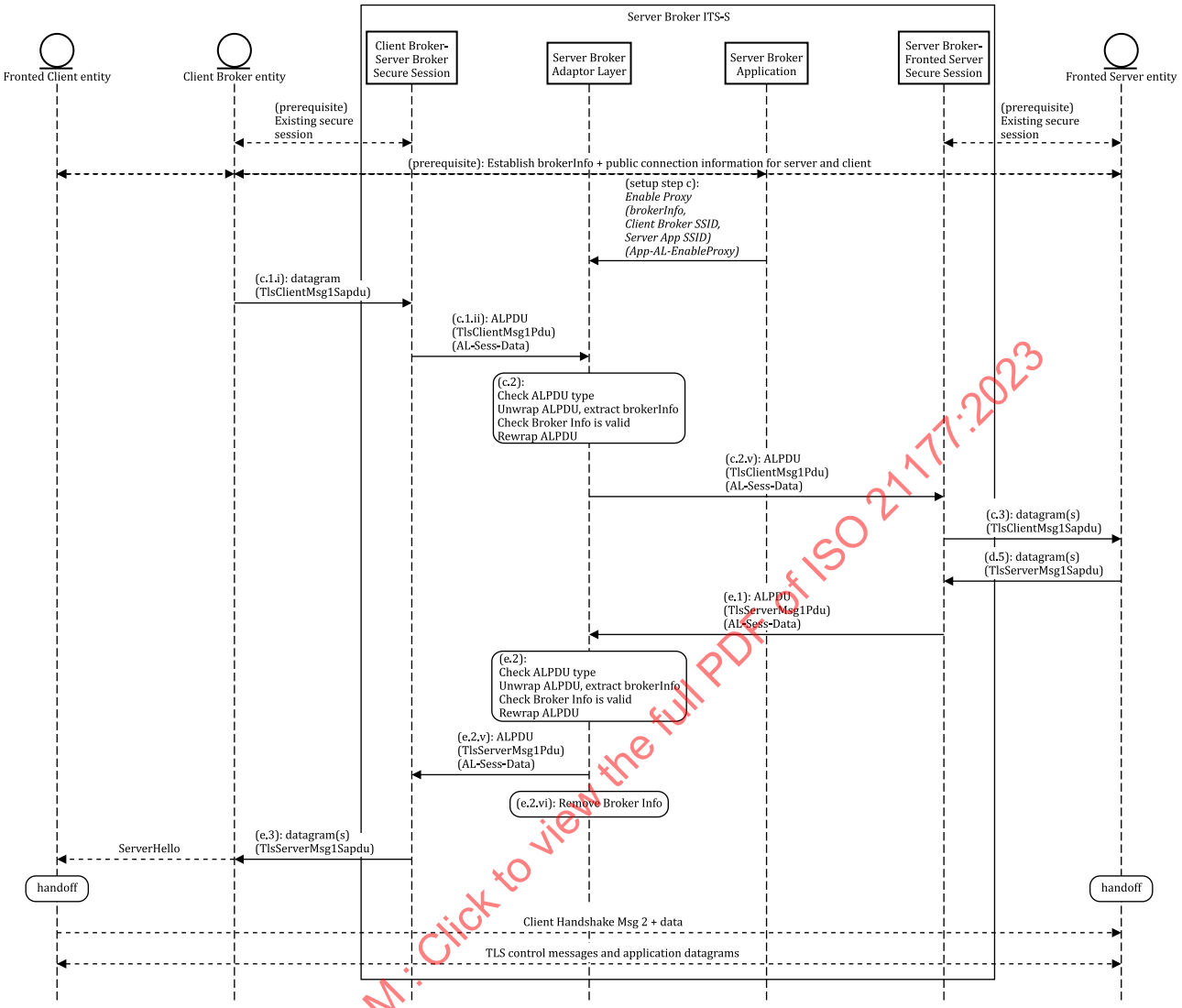


Figure 22 — Sequence diagram for secure connection brokering operations on the server broker

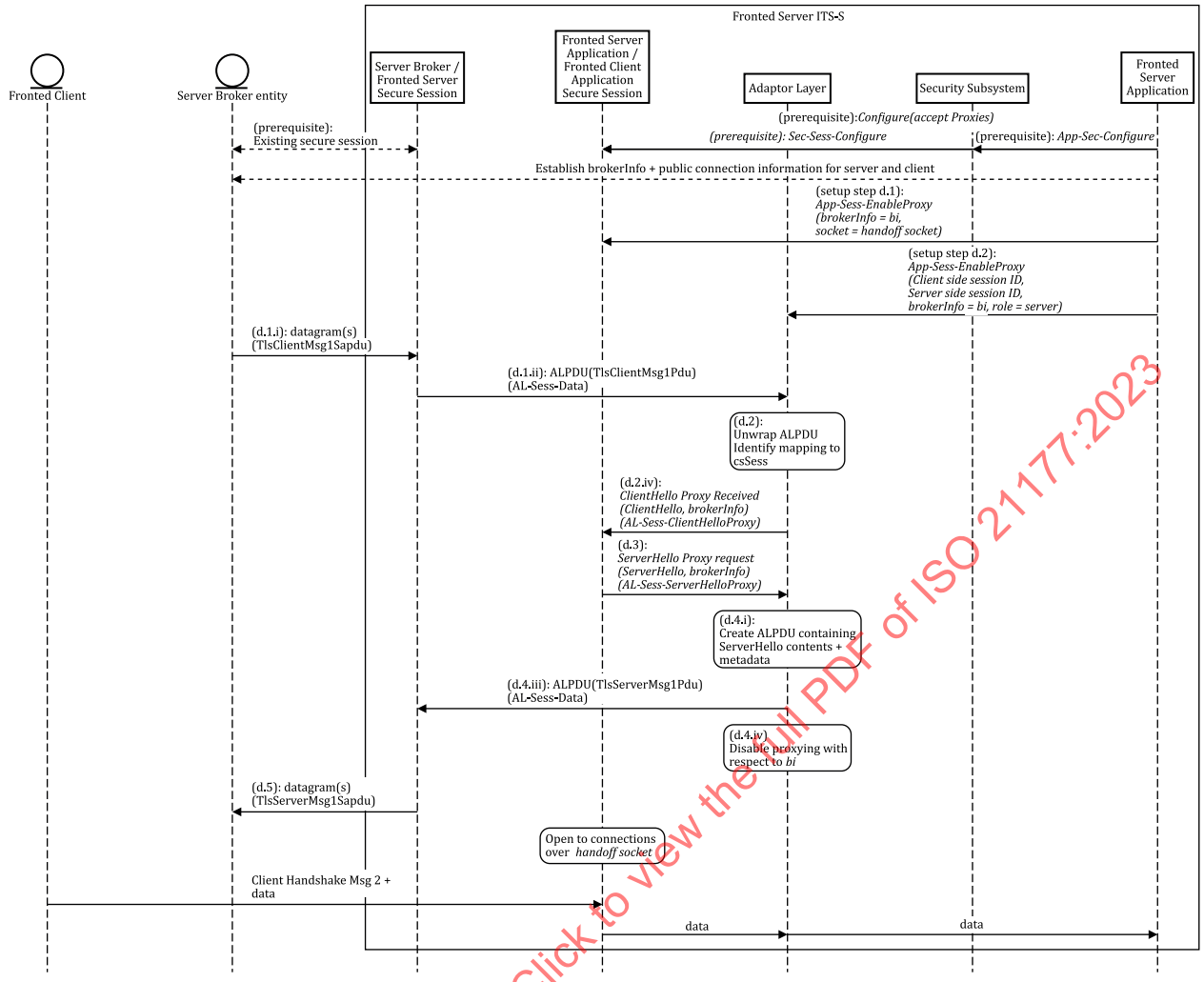


Figure 23 — Sequence diagram for secure connection brokering operations on the fronted server

6.11 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 specified in 5.3, there has been an initialize step using the methods of 6.5.

The process steps for this process flow are as follows.

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 7.7.8,

ii) the security subsystem:

a) ACKs the request via `App-Sec-EndSession.indication` primitive; see 7.7.9,

b) 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 7.7.9,

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 24.

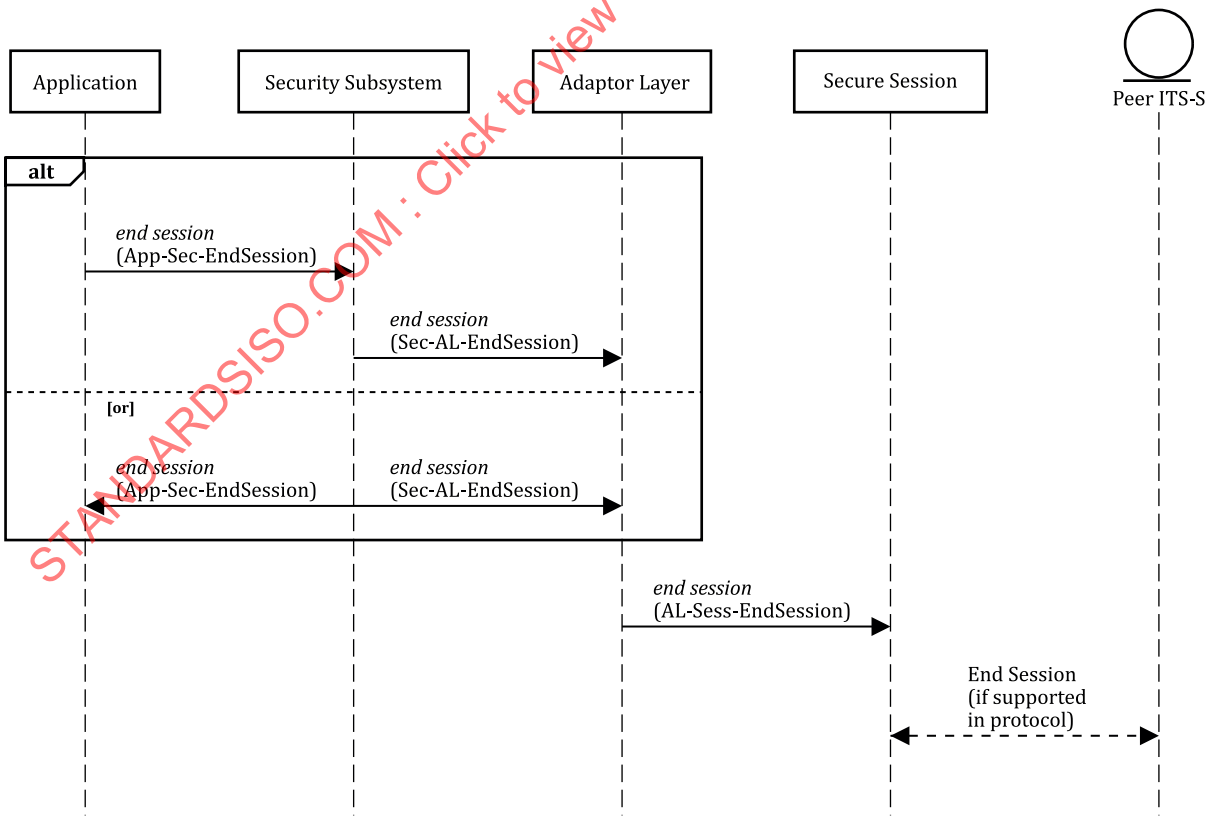


Figure 24 — Sequence diagram for Force End Session

6.12 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 specified in 5.3, there has been an initialize step using the methods of 6.5.

The process steps for this process flow are as follows.

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 7.7.9;

   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 25.

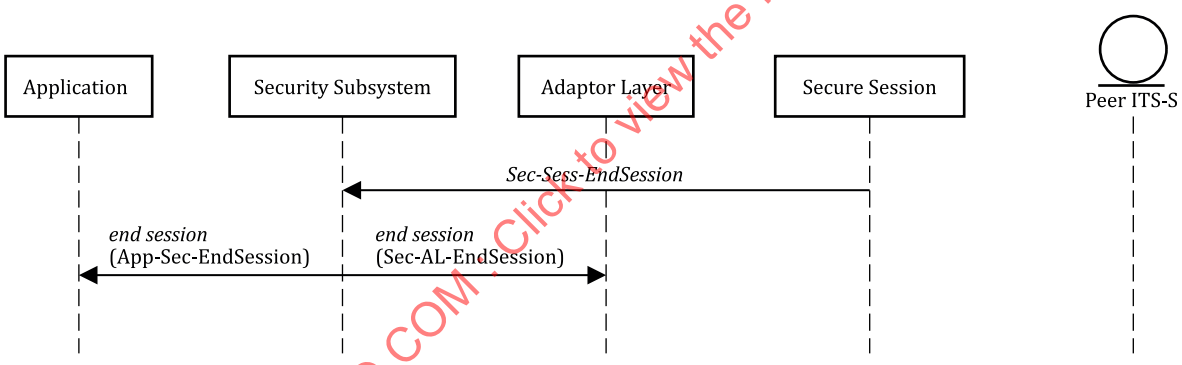


Figure 25 — Sequence diagram for Session Terminated at Session Layer

6.13 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 as follows.

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 7.7.10,

ii) the security subsystem:

a) ACKs the request via `App-Sec-Deactivate.confirm` primitive; see 7.7.11,

b) 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 7.7.12,

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) stops 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 26.

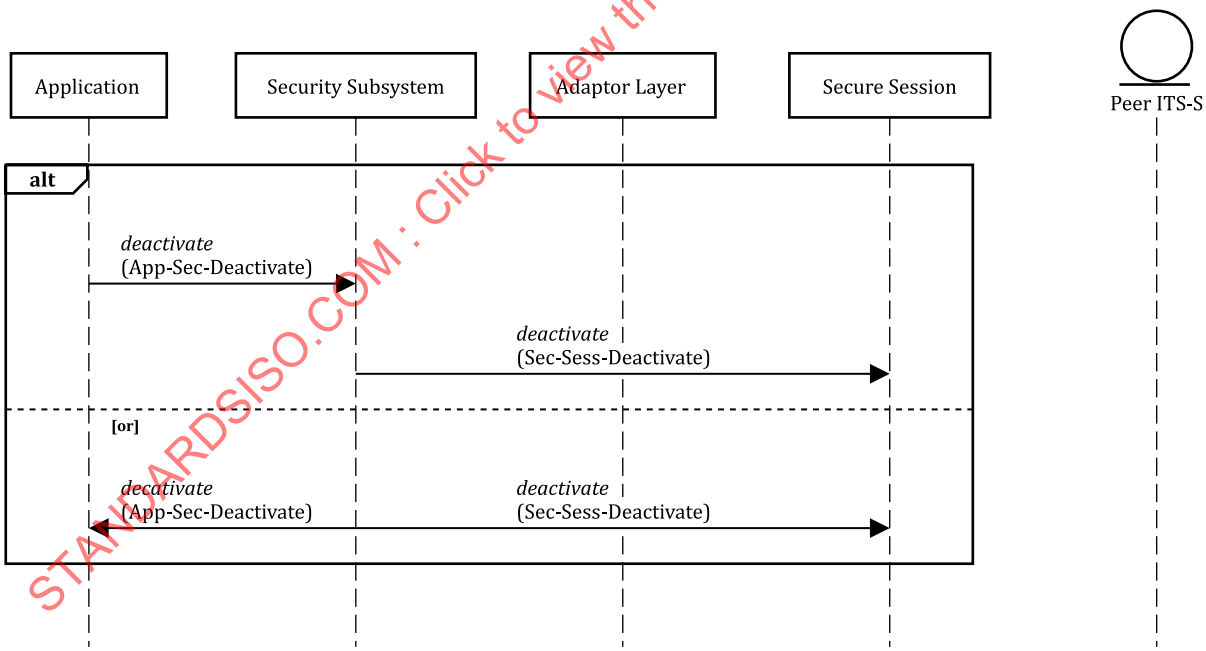


Figure 26 — Sequence diagram for Deactivate

6.14 Secure session example

This subclause illustrates how the process flows from the previous section can 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.

— 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 6.11. 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 6.12.

— Both parties deactivate the secure session services as specified in 6.13.

Figure 27 shows all the information flows associated with this illustration. Optional data flows are in italics.

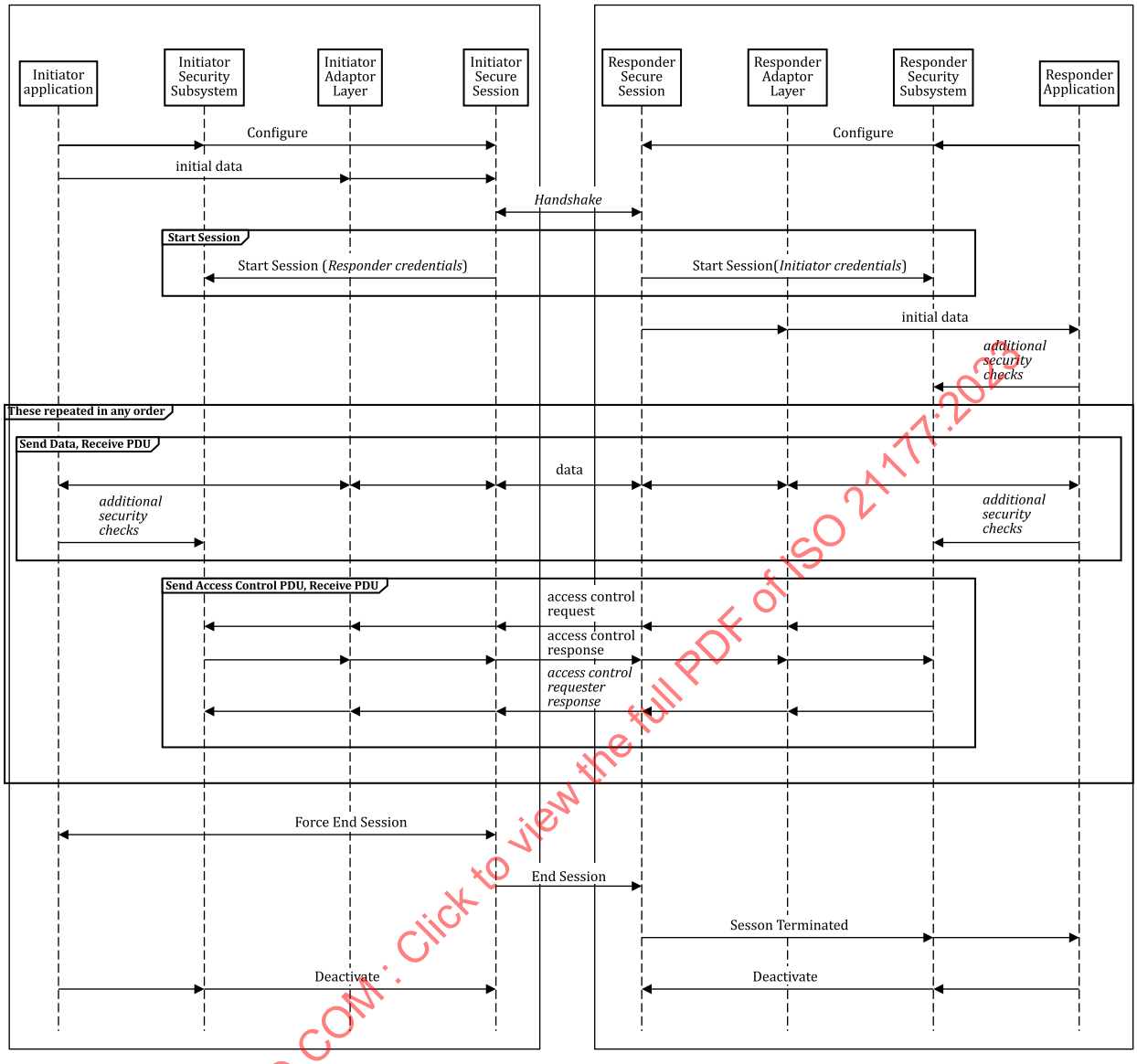


Figure 27 — Sequence diagram for complete secure session example

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 7.7.6;

— signs outgoing APDUs if so requested by the application using `App-Sec-Data.request`; see 7.7.4;

— verifies incoming APDUs if so requested by the application using `App-Sec-Incoming.request`; see 7.7.6;

— 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 6.11,

  — "Session" terminated at session layer, see 6.12, and

  — "Deactivate", see 6.13.

The application uses the security subsystem to determine whether an APDU is valid for use. An application that is conformant to this document shall invoke `App-Sec-Incoming.request` (see 7.7.6) 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 document 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 and the content of the APDU 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 7.7.6) 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 so 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 so requires, the security subsystem can generate access control PDUs to request additional information from the peer trusted entity and use the responses to update its state. 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 (= ITS-AID) and (optionally) SSP that shall appear in the received certificate for the secure session handshake as obtained via the `Sec-Sess-StartSession.indication` service primitive;

— 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 shall be fulfilled to carry out particular activities.

Since the security subsystem applies the access control policy directly to APDUs, it is necessary for the security subsystem associated with a particular application to 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 in 7.3 to facilitate 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 personal identification number (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. The mechanism used in this document to implement enhanced authentication is based on password based authenticated key exchange (PB-AKE) mechanisms, a well-studied mechanism within cryptography, which allows two parties to establish shared knowledge of a weak secret without allowing offline guessing attacks by an attacker who records or participates in the session. An informative overview of PB-AKE is provided in IEEE 1363.2.

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 an (*i*: application, *j*: session, *k*: 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 28.

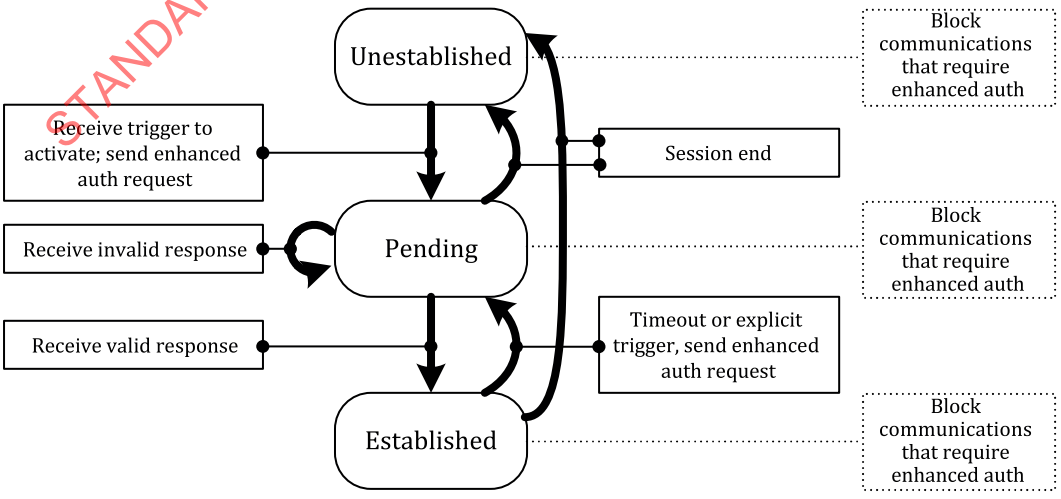


Figure 28 — Informal state machine for the enhanced authentication module

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 as per the mechanisms specified 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 transition 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);[14] see 7.3.6.

7.3.6 Enhanced authentication using SPAKE2

SPAKE2 is carried out as specified in Reference [14] 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 section 3.2 of Reference [14]. 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 section 3.2 of Reference [14]. 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 section 3.2 of Reference [14].

— ID is the certificate out of *A* and *B* as specified 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 can potentially request proof of knowledge from the original requester. In this case, the requester sends a “requester response” that contains the field hashOfKPrimeRe1Id, calculated as follows.

— The quantity *K* is the 32-byte output of the SHA-256 hash of the session variables as specified in section 3.2 of Reference [14].

— ID is the certificate out of *A* and *B* as specified 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 authentication1;

— responder believed authenticated, no requester authentication requested;

— responder believed authenticated, no requester authentication received;

— mutually authenticated.

The Request data type is specified in 7.6.13. The Response data type is specified in 7.6.14.

NOTE At the time of publication, the Internet Research Task Force (IRTF) had decided to deprecate the use of SPAKE2 in favour of a more compact Authenticated Key Exchange algorithm known as CPACE. As of December 2021, the IRTF has not published a stable CPACE specification. It is anticipated that once a stable specification is available, support for CPACE will be added to a future edition of this document.

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 authorization 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 authorization response access control PDU is signed by an IEEE 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 authorizations that the owner is requesting the accessor to demonstrate. It is specified in 7.6.7 through 7.6.9. The structure supports combining atomic authorization 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 7.6.10. 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 authorizations requested in the Request. However, the accessor may create a response that does not match the authorizations requested in the Request. In this case the owner may choose to store the received authorizations 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 authorizations 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 Security Management Information Request

7.5.1 Rationale

A natural piece of information for an access control policy to take into account is the freshness of the authorization of a device that is requesting access. For example, an access control policy could have a requirement that a device has been issued a certificate no more than 24 hours before the access request. This provides assurance to the accessed device that the authorization infrastructure believed the accessing device to be secure at least until very recently.

It is natural for the accessed device to want to apply a similar condition to the CA certificates in the chain of an accessing device. In other words, to ensure that not just the device itself but all CAs involved in making the trust statement were believed secure in the last 24 hours. For EE certificates this is straightforward, because the certificate can simply be reissued periodically. For CA certificates this approach cannot be taken. First, issuing a new CA certificate is currently an expensive operation. Second, all of the certificates issued by a CA certificate need to have validity periods within the validity period of the CA certificate, so reissuing a CA certificate frequently would create a series of requirements to reissue all of the downstream certificates at the same or higher frequency. This could be managed in principle for a tree of CA certificates which only issued end-entity certificates with a short validity period, but in practice it would likely be unwieldy. Additionally, since CAs are expensive to establish, it is useful to be able to use them to issue end-entity certificates for different applications with different natural lifetimes.

These issues could potentially be overcome to create a system based on short-lived CA certificates in order to satisfy applications with a requirement that the trust status of the entire chain is fresh. However, in order to support the deployment of these applications without creating a requirement for short-lived CA certificates all the way up the chain, this document supports an alternative model based on certificate revocation lists (CRLs).

In this alternative model, the approach is that instead of frequently reissuing CA certificates, the system frequently publishes the CRLs that the CAs would appear on. Thus, instead of demanding a recently issued CA certificate, the access-granting device can demand a recently published CRL. If a CA is absent from the recently published CRL, that is equivalent to making a statement that the CA was trusted by the authorization infrastructure at least as recently as the issue date of the CRL. This combination

of not-necessarily-recent CA certificate and recent CRL thus gives the same assurance to an access-granting device that a recently issued CA certificate would.

Note that in this document the approach is for an access-granting device to request, and for an accessing device to provide, the full CRL on which the CA certificate would appear. An alternative approach known as Online Certificate Status Protocol (OCSP) Stapling is specified in RFC 8446 and in Reference [13] for X.509 certificates. Where a CRL is a list of all revoked certificates that are the responsibility of a given CRL signer, an OSCP Response is a statement about a single certificate that can indicate "revoked" or "not revoked". An OCSP Response therefore both is more compact than a CRL and makes a positive statement in the case that the certificate in question has not been revoked, in contrast to the CRL where the fact that a certificate has not been revoked is deduced by its absence from the CRL. In general, therefore, OCSP responses are a better technology than sending entire CRLs. However, in the settings addressed by this document, a CRL-based freshness mechanism is useful to define because:

— the CRLs will not in general be very long. This mechanism is just for CRLs for CAs, and the number of revoked CAs will probably be small;

— there is no definition of OCSP for IEEE 1609.2 certificates, only for X.509 certificates. As such, defining a CRL-based mechanism enables deployment in the near future of applications with high sensitivity to the freshness of trust information. If in the future it becomes clear that a protocol with properties more like OSCP (statements about single certificates and positive statements about non-revocation) is required, that protocol can be developed and deployed at that point.

7.5.2 General

This subclause specifies security management information exchange. In security management information exchange, one party acts as the owner and the other party acts as the accessor. The owner sends a Security Management Information Request access control PDU to request that the accessor provides public security management information relevant to the authorization state of the accessor. The accessor may respond by sending a Security Management Information Response access control PDU to provide this information.

The security management information that may be requested in this version of this document shall be one of the following.

— The ETSI-style Certificate Revocation List (CRL), as specified in ETSI TS 102 941, that is relevant to an AA certificate that issued one of the ATs presented during the handshake or as part of an extended authentication exchange.

— The ETSI-style Certificate Trust List (CTL), as specified in ETSI TS 102 941, that is relevant to a certificate presented during the handshake or as part of an extended authentication exchange.

— The IEEE-style CRL, as specified in IEEE 1609.2, that is relevant to any certificate presented during the handshake or as part of an extended authentication exchange, or any certificate in its chain.

— The certificate chain relevant to any certificate presented during the handshake or as part of an extended authentication exchange.

Security management information provided in a response in this context is already signed by the originator (the Certificate Authority (CA) or CRL Generator) and is not signed by the responder.

The Security Management 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 in this document.

The Request PDU includes a statement of the security management information that the owner is requesting from the accessor. Requests for separate instances of the security management information are made separately and identified by an identifier generated by the owner.

The Response PDU is sent in response to a Request PDU. It consists of an `Ieee1609Dot2Data` of type unsecured, containing the identifier of the corresponding request and either the requested information or an indication that the requested information is unavailable.

7.6 Data types

7.6.1 General

Access control PDUs are specified in ASN.1 and encoded with the Canonical Octet Encoding Rules (C-OER). The ASN .1 basic notation is specified in ISO/IEC 8824-1. The C-OER are specified in ITU-T X.696. This document specifies ASN.1 modules provided in an electronic attachment to this document. Details shall be as specified in Annex B.

7.6.2 Imports

The data type definitions import the following definitions from IEEE 1609.2 ASN.1 modules:

— `CrlSeries, EccP256CurvePoint, HashedId8, HashedId32, Psid, PsidSspRange, Time32` from the IEEE 1609.2 Base Types module;

— `Certificate, Ieee1609Dot2Data, PduFunctionalType, iso21177SessionExtension` from the IEEE 1609.2 Schema module extended by this document; see Annex C;

— `SecuredCrl` from the IEEE 1609.2 CRL Protocol module.

The data type definitions import the following definitions from ETSI TC ITS security ASN.1 modules:

— `CertificateRevocationListMessage, TlmCertificateTrustListMessage` shall be as being part from the CA Message module specified in ETSI TS 102 941;

— `EtsiTs102941CrlRequest, EtsiTs102941CtrlRequest` shall be as being part from the ETSI Originating HeaderInfo Extension module specified in ETSI TS 103 097.

7.6.3 "Helper" data types

The parameterized type `Ieee1609Dot2Data-Signed-PduFunctionalType` is used as shorthand in the ASN.1 below to indicate a signed IEEE 1609.2 SPDU with a specific payload and a specific PduFunctionalType field.

```
Ieee1609Dot2Data-Signed-PduFunctionalType {Tbs, PduFuncType}::=
  Ieee1609Dot2Data (WITH COMPONENTS {...,
    content (WITH COMPONENTS {...,
      signedData (WITH COMPONENTS {...,
        tbsData (WITH COMPONENTS {...,
          payload (WITH COMPONENTS {...,
            data (WITH COMPONENTS {...,
              content (WITH COMPONENTS {
                unsecuredData (CONTAINING Tbs)
              })
            })
          }),
          headerInfo (WITH COMPONENTS {...,
            generationTime PRESENT,
            expiryTime ABSENT,
            generationLocation ABSENT,
            p2pcdLearningRequest ABSENT,
            missingCrlIdentifier ABSENT,
            encryptionKey ABSENT,
            pduFunctionalType (PduFuncType) PRESENT
          })
        })
      })
    })
  })
```

`IeeeCrl`, `EtsiCrl`, and `EtsiCtl` are renamings of structures exported from IEEE and ETSI ASN.1 modules, where the new names allow a consistent naming convention within the context of this document.

```
IeeeCrl ::= SecuredCrl
EtsiCrl ::= CertificateRevocationListMessage
EtsiCtl ::= TlmCertificateTrustListMessage
```

7.6.4 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 } |
   { SecurityMgmtInfoPdu  IDENTIFIED BY securityMgmtInfoId } |
   { SessionExtensionPdu  IDENTIFIED BY sessionExtensionId }
   ...
}

Iso21177AccessCtrlPduId ::= INTEGER (0..255)
   iso21177AccessCtrlPduId-reserved Iso21177AccessCtrlPduId ::= 0 --'00'H
   accessControlResultId    Iso21177AccessCtrlPduId ::= 1 --'01'H
   extendedAuthId           Iso21177AccessCtrlPduId ::= 2 --'02'H
   enhancedAuthId           Iso21177AccessCtrlPduId ::= 3 --'03'H
    securityMgmtInfoId       Iso21177AccessCtrlPduId ::= 4 --'04'H
    sessionExtensionId       Iso21177AccessCtrlPduId ::= 5 --'05'H
```

7.6.5 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)
```

7.6.6 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
```

7.6.7 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 authorization;

— 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
}
```

7.6.8 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 authorizations indicated within the field;

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

— the component `contents` contains a single authorization 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
}
```

7.6.9 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 specified in IEEE 1609.2;

— the component `idRegExp` indicates that the responder is being requested to provide an IEEE 1609.2 certificate whose name matches the indicated regular expression. The regular expression is conformant to IEEE 1003.1-2017, Clause 9;

— the component `issuer` indicates that the responder is being requested to provide an IEEE 1609.2 certificate such that one of the issuing certificate authority certificates in the certificate chain, see IEEE 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)),
    ...
}
```

7.6.10 ExtendedAuthResponse

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

```
ExtendedAuthResponse ::=
  Ieee1609Dot2Data-Signed-PduFunctionalType {
    ExtendedAuthResponsePayload,
    PduFunctionalType(iso21177ExtendedAuth)
  }
```

7.6.11 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 `Iso21177AccessControlPdu` that contained the corresponding request. This includes the nonce, providing assurance that the response is to the specific request and has not been replayed.

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

7.6.12 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(%7bEnhancedAuthPduTypes%7d%7b@.messageId%7d),
         ...
}

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'
```

7.6.13 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))
}
```

7.6.14 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
}
```

7.6.15 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))
```

7.6.16 SecurityMgmtInfoPdu

The ASN.1 type of the access control PDUs associated with security management information is `SecurityMgmtInfoPdu`.

```
SecurityMgmtInfoPdu ::= CHOICE {
    request SecurityMgmtInfoRequest,
    response SecurityMgmtInfoResponse,
    ...
}
```

7.6.17 SecurityMgmtInfoRequest

In this ASN.1 type, the field id is set by the requester and should be unique to all requests in the session.

The requests for ETSI TS 102 941 CRLs and CTLs shall use the format and shall have the semantics specified in ETSI TS 103 097.

```
SecurityMgmtInfoRequest ::= SEQUENCE {
    id        OCTET STRING (SIZE(2)),
    messageId      SEC-MGMT-INFO-REQUEST.
                  &id({SecurityMgmtInfoRequestTypes}),
    value          SEC-MGMT-INFO-REQUEST.
```

```
                    &Type({SecurityMgmtInfoRequestTypes}{@.messageId}),
    ...
}

SEC-MGMT-INFO-REQUEST ::= CLASS {
&id SecurityMgmtInfoRequestId UNIQUE,
&Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}

SecurityMgmtInfoRequestTypes SEC-MGMT-INFO-REQUEST ::= {
{ EtsiTs102941CrlRequest IDENTIFIED BY etsiCrlRequestId } |
{ EtsiTs102941CtlRequest IDENTIFIED BY etsiCtlRequestId } |
{ IeeeCrlRequest         IDENTIFIED BY ieeeCrlRequestId } |
{ CertChainRequest       IDENTIFIED BY certChainRequestId } ,
...
}

SecurityMgmtInfoRequestId::= INTEGER (0..255)
securityMgmtInfoRequestId-reserved
                          SecurityMgmtInfoRequestId ::= 0 --'00'H
etsiCrlRequestId          SecurityMgmtInfoRequestId ::= 1 --'01'H
etsiCtlRequestId          SecurityMgmtInfoRequestId ::= 2 --'02'H
ieeeCrlRequestId          SecurityMgmtInfoRequestId ::= 3 --'03'H
certChainRequestId        SecurityMgmtInfoRequestId ::= 4 --'04'H
```

7.6.18 EtsiCrlRequest

This ASN.1 type is used to request an ETSI certificate revocation list (CRL) as specified in ETSI TS 102 941. In this ASN.1 type:

— rootCa is the HashedId8 of the Root CA certificate that has issued the CRL being requested;

— issuedAfter is a Time32 such that requester is requesting that the CRL returned in the response has a thisUpdate value greater than this issuedAfter value.

```
EtsiCrlRequest ::= SEQUENCE {
    rootCa       HashedId8,
    issuedAfter  Time32,
    ...
}
```

7.6.19 CertChainRequest

This ASN.1 type is used to request a chain of IEEE certificates where the bottom certificate in the chain is the issuer of a certificate *C*.

— `issuerId` is the value that appears in the field *C*.`issuer`.

— `length` is the requested length of the chain. This follows the same syntax as the parameter "Signer Identifier Cert Chain Length" to the primitive Sec-SignedData.request in IEEE 1609.2, except that the value 0 indicates that the requester is requesting the entire certificate chain back to the root CA certificate.

```
CertChainRequest ::= SEQUENCE {
    issuerId HashedId8,
    length INTEGER,
    ...
}
```

7.6.20 SecurityMgmtInfoResponse

In this ASN.1 type, the field ID shall be identical to its value in the request to which the response is reponding.

```
SecurityMgmtInfoResponse ::= SEQUENCE {
    id        OCTET STRING (SIZE(2)),
```

```
messageId SEC-MGMT-INFO-RESPONSE.&id({SecurityMgmtInfoResponseTypes}),
   value          SEC-MGMT-INFO-RESPONSE.
             &Type({SecurityMgmtInfoResponseTypes}{@.messageId}),
   ...
}

SEC-MGMT-INFO-RESPONSE ::= CLASS {
&id SecurityMgmtInfoResponseId UNIQUE,
&Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}

SecurityMgmtInfoResponseTypes SEC-MGMT-INFO-RESPONSE ::= {
{SecurityMgmtInfoErrorResponse IDENTIFIED BY securityMgmtInfoErrorId
      } |
{EtsiCrlResponse               IDENTIFIED BY etsiCrlResponseId } |
{EtsiCtlResponse               IDENTIFIED BY etsiCtlResponseId } |
{IeeeCrlResponse               IDENTIFIED BY ieeeCrlResponseId } |
{CertChainResponse             IDENTIFIED BY certChainResponseId }
...
}

SecurityMgmtInfoResponseId::= INTEGER (0..255)
  securityMgmtInfoErrorId SecurityMgmtInfoResponseId ::= 0 --'00'H
  etsiCrlResponseId       SecurityMgmtInfoResponseId ::= 1 --'01'H
  etsiCtlResponseId       SecurityMgmtInfoResponseId ::= 2 --'02'H
  ieeeCrlResponseId       SecurityMgmtInfoResponseId ::= 3 --'03'H
  certChainResponseId        SecurityMgmtInfoResponseId ::= 4 --'04'H
```

7.6.21 SecurityMgmtInfoErrorResponse

This ASN.1 type is used to indicate that requested security management information is not available and to give a reason why it is not available.

— `unavailable (0)` is a generic response used if no other error value is appropriate.

— `inappropriate_type (1)` is used if the request was for the wrong type of security management material:

  — an etsiCrlRequest when the indicated root CA is in the IEEE CRL system.

```
SecurityMgmtInfoErrorResponse ::= ENUMERATED {
  unavailable(0), inappropriate_type(1), ...
}
```

7.6.22 EtsiCrlResponse

This ASN.1 type is used to respond to an `EtsiCrlRequest`. This ASN.1 type consists of a single field, containing the requested CRL, which has a thisUpdate value greater than the issuedAfter value in the corresponding request. This type shall be as specified in ETSI TS 102 941.

```
EtsiCrlResponse ::= EtsiCrl
```

7.6.23 EtsiCtlResponse

This ASN.1 type is used to respond to an `EtsiCtlRequest`. In this ASN.1 type:

— `ctl` is the most recent CTL issued by the TLM indicated in the corresponding request or by its successor TLM.

```
EtsiCtlResponse ::= EtsiCtl
```

7.6.24 IeeeCrlResponse

This ASN.1 type is used to respond to an `IeeeCrlRequest`. This ASN.1 type consists of a single field, containing the requested CRL, which has an issueDate value greater than the `issuedAfter` value in the corresponding request.

```
IeeeCrlResponse ::= IeeeCrl
```

7.6.25 CertChainResponse

This ASN.1 type is used to respond to a CertChainRequest. It consists of an ordered sequence of certificates such that:

— the `HashedId8` of the first certificate in the chain is equal to the `issuerId` value in the corresponding request;

— each certificate in the chain issues the certificate previous to it;

— the number of certificates in the chain is the number indicated by the `length` field in the request, except that:

  — if `length` was positive with an absolute value longer than the length of the chain, the chain contains all of the certificates back to the root CA certificate;

  — if `length` was negative with an absolute value longer than the length of the chain, the chain contains a single certificate.

```
CertChainResponse ::= SEQUENCE OF Certificate
```

7.6.26 SessionExtensionPdu

The ASN.1 type of the access control PDUs used to request an extension of a session with a different certificate is `SessionExtensionPdu`.

To create a `SessionExtensionPdu`, the sender:

a) selects the new certificate;

b) sets the new certificate as the payload of a `SessionExtensionPdu` with type identified by `sessionExtensionContentsId`, i.e. a `SessionExtensionContentsPdu` as specified in 7.6.26;

c) creates a `SessionExtensionInner` as specified below by creating a signed SPDU where:

  1) the contents are the `SessionExtensionContentsPdu`,

  2) the `pduFunctionalType` field is present in the signed SPDU and is set to `iso21177SessionExtension`,

  3) the signature is generated using the old certificate, i.e. the certificate used in the handshake or in the previous session extension exchange;

d) creates a `SessionExtensionInnerPdu` by encapsulating the `SessionExtensionInner` in a `SessionExtensionPdu` with type identified by `sessionExtensionInnerId`;

e) creates a `SessionExtensionOuter` as specified below by creating a signed SPDU where:

  1) the content is the `SessionExtensionInnerPdu`,

  2) the `pduFunctionalType` field is present in the signed SPDU and is set to `iso21177SessionExtension`,

  3) the signature is generated using the new certificate, i.e. the certificate contained in the `SessionExtensionContents`;

f) encapsulating the `SessionExtensionOuter` in a `SessionExtensionPdu` with type identified by `sessionExtensionOuterId`, and then creates an `Iso21177AccessControlPdu` containing that `SessionExtensionPdu`;

g) sends the `Iso21177AccessControlPdu` to the other party.

On receipt, the other party generates and send an Iso21177AccessControlPdu containing a `SessionExtensionPdu` with type identified by `sessionExtensionAckId`, with the `SessionExtensionAck` set equal to the HashedId8 of the "new" certificate from the received `IsoAccessControlPdu`. The `HashedId8` is calculated as specified in IEEE 1609.2.

The new certificate shall contain in the appPermissions field a PsidSsp containing the ITS-AID that appeared in the SignedData HeaderInfo in the initial handshake from the sender (the "handshake ITS-AID"). The SSP associated with the ITS-AID shall grant at least the privileges that were granted by the SSP associated with the handshake ITS-AID in the handshake certificate. The definition of "at least the privileges" is ITS-AID dependent and can also depend on the access control policy.

```
SessionExtensionPdu ::= SEQUENCE {
    messageId   SESS-EXT-PDU.
                  &id({SessionExtensionPduTypes}),
    value       SESS-EXT-PDU.
                  &Type({SessionExtensionPduTypes}{@.messageId}),
    ...
}

SESS-EXT-PDU ::= CLASS {
     &id SessionExtensionPduId UNIQUE,
     &Type
} WITH SYNTAX {&Type IDENTIFIED BY &id}

SessionExtensionPduTypes SESS-EXT-PDU ::= {
{ SessionExtensionOuter IDENTIFIED BY sessionExtensionOuterId } |
{ SessionExtensionInner IDENTIFIED BY sessionExtensionInnerId } |
{ SessionExtensionContents IDENTIFIED BY sessionExtensionContentsId } |
{ SessionExtensionAck       IDENTIFIED BY sessionExtensionAckId } ,
     ...
}

SessionExtensionPduId ::= INTEGER(0..255)
  sessionExtensionPduId-reserved SessionExtensionPduId ::= 0 -- '00'H
  sessionExtensionOuterId        SessionExtensionPduId ::= 1 -- '01'H
  sessionExtensionInnerId        SessionExtensionPduId ::= 2 -- '02'H
  sessionExtensionContentsId     SessionExtensionPduId ::= 3 -- '03'H
  sessionExtensionAckId          SessionExtensionPduId ::= 4 -- '04'H

SessionExtensionOuter ::=
  Ieee1609Dot2Data-Signed-PduFunctionalType {
    SessionExtensionInnerPdu,
    PduFunctionalType(iso21177SessionExtension)
  }

SessionExtensionInnerPdu ::= SessionExtensionPdu (WITH COMPONENTS{
  messageId (sessionExtensionInnerId)
})

SessionExtensionInner ::=
  Ieee1609Dot2Data-Signed-PduFunctionalType {
    SessionExtensionContentsPdu,
    PduFunctionalType(iso21177SessionExtension)
  }

SessionExtensionContentsPdu ::=
  SessionExtensionPdu (WITH COMPONENTS{
  messageId (sessionExtensionContentsId)
})

SessionExtensionContents ::= Certificate

SessionExtensionAck ::= HashedId8
```

7.7 App-Sec Interface

7.7.1 App-Sec-Configure.request

This service primitive `App-Sec-Configure.request` instructs the security subsystem to activate an instance of the secure session services.

```
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
)
```

The parameters of the service primitive are as shown in Table 2.

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

Name	Type	Valid range	Description
'Application ID'	Structure See 5.5 (`ITSsapiid`)	See 5.5	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 6.10.
– 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 specified in IEEE 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 –			
'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 specified in this document; it is simply used as an identifier.

Table 2 *(continued)*

Name	Type	Valid range	Description
– 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).

7.7.2 App-Sec-Configure.confirm

This service primitive `App-Sec-Configure.confirm` returns the results of the corresponding request primitive.

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

The parameters of the service primitive are as shown in Table 3.

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

7.7.3 App-Sec-StartSession.indication

This service primitive `App-Sec-StartSession.indication` 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.

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

The parameters of the service primitive are as shown in Table 4.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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

7.7.4 App-Sec-Data.request

This service primitive `App-Sec-Data.request` 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.

```
App-Sec-Data.request (
   Application ID,
```

```
    Session ID,
    Cryptomaterial Handle,
    Data,
    Signing Parameters
)
```

The parameters of the service primitive are as shown in Table 5.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (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 as per IEEE 1609.2 associated with a certificate. See IEEE 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 IEEE 1609.2 `SecSignedData.request` primitive. See IEEE 1609.2 for further details.	The set of parameters to be provided to the IEEE 1609.2 `SecSignedData.request` primitive

On receipt, the security subsystem attempts to sign the indicated Data with the indicated cryptomaterial handle and Signing Parameters.

7.7.5 App-Sec-Data.confirm

This service primitive `App-Sec-Data.confirm` returns the result of the corresponding request primitive.

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

The parameters of the service primitive are as shown in Table 6.

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 `SecSignedData.confirm`. See IEEE 1609.2 for further details.	The result of the operation
'Signed Data'	Octet String	An Ieee1609Dot2Data of type signedData. See IEEE 1609.2for 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.

7.7.6 App-Sec-Incoming.request

This service primitive `App-Sec-Incoming.request` is generated by the application to request that the security subsystem applies the access control policy to an incoming APDU.

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

The parameters of the service primitive are as shown in Table 7.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (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 Ieee-1609Dot2Data' is false, any Octet String. See IEEE 1609.2 for further details.	The received APDU
'Is Ieee1609Dot-2Data'	Boolean	True, False	Whether the received APDU is an Ieee1609Dot2Data
'Signed Data Ver-ification Param-eters'	Structured	The set of parameters to be pro-vided to the IEEE 1609.2 Sec-SignedDataVerification.request primitive. See IEEE 1609.2 for further details.	Provided only if 'Is Ieee1609Dot2Da-ta' 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.

7.7.7 App-Sec-Incoming.confirm

This service primitive App-Sec-Incoming.confirm returns the result of the corresponding request primitive.

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

The parameter of the service primitive is as shown in Table 8.

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

Name	Type	Valid range	Description
'Result'	Enumerated	Success, Invalid Ieee1609Dot2Data Type, Invalid Signed Ieee1609Dot2Data, Invalid APDU as per access control policy / request sent, Invalid APDU as 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 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.

7.7.8 App-Sec-EndSession.request

This service primitive `App-Sec-EndSession.request` is used to request that a particular session is ended.

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

The parameters of the service primitive are as shown in Table 9.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 6.11, step g.1.ii).

7.7.9 App-Sec-EndSession.indication

This service primitive `App-Sec-EndSession.indication` 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 as 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`.

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

The parameters of the service primitive are as shown in Table 10.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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.

Table 10 *(continued)*

Name	Type	Valid range	Description
'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.

7.7.10 App-Sec-Deactivate.request

This service primitive `App-Sec-Deactivate.request` is used to request that a particular instance of the secure session service is deactivated as specified in 6.13.

```
App-Sec-Deactivate.request (
   Application ID,
   secure session Instance ID
)
```

The parameters of the service primitive are as shown in Table 11.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 6.13, step g.1.ii).

7.7.11 App-Sec-Deactivate.confirm

This service primitive `App-Sec-Deactivate.confirm` confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

No effect of receipt is specified.

7.7.12 App-Sec-Deactivate.indication

This service primitive `App-Sec-Deactivate.indication` is used by the security subsystem to notify the application that, as 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`.

```
App-Sec-Deactivate.indication (
   Application ID,
   secure session Instance ID
)
```

The parameters of the service primitive are as shown in Table 12.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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.

7.8 Security subsystem internal interface

7.8.1 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.

7.8.2 Sec-AuthState.request

This service primitive `Sec-AuthState.request` allows the access control subsystem to request the current authorization state from the authorization state subsystem.

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

The parameters of the service primitive are as shown in Table 13.

Table 13 — Parameters for Sec-AuthState.request

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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.

7.8.3 Sec-AuthState.confirm

The service primitive `Sec-AuthState.confirm` 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.

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

The parameters of the service primitive are as shown in Table 14.

Table 14 — Parameters for Sec-AuthState.confirm

Name		Type	Valid range	Description
'Application ID'		Structure. See 5.5 (`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 specified in IEEE 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 specified 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 specified in Table 15.

Table 15 — Parameters for Sec-AuthState.confirm

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 `AccessControl`, 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 vi `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 specified in ASN.1 and encoded with the canonical octet encoding rules (C-OER). The ASN .1 basic notation is specified in ISO/IEC 8824-1. The C-OER are specified in ITU-T X.696.

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 AccessControl

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 `BrokerInfo` 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 `BrokerInfo` 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 `App-AL-Data.request` is used by the application to submit an APDU for transmission across the secure session.

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

The parameters of the service primitive are as shown in Table 16.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (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

The service primitive App-AL-Data.confirm confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

8.3.3 App-AL-Data.indication

The service primitive App-AL-Data.indication 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.

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

The parameters of the service primitive are as shown in Table 17.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (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

Table 17 *(continued)*

Name	Type	Valid range	Description
'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

The service primitive `App-AL-EnableProxy.request` is used by the application to configure the adaptor layer to proxy TLS handshake messages between two secure session instances.

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

The parameters of the service primitive are as shown in Table 18.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 specified 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 shall 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 recived 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 `Sec-AL-AccessControl.request` 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 7.6 but is opaque to the adaptor layer.

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

The parameters of the service primitive are as shown in Table 19:

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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

The service primitive `Sec-AL-AccessControl.confirm` confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

No effect on receipt isspecified.

8.4.3 Sec-AL-AccessControl.indication

The service primitive `Sec-AL-AccessControl.indication` 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`.

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

The parameters of the service primitive are as shown in Table 20.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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

The service primitive `Sec-AL-EndSession.request` is used to request that a particular session is ended.

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

The parameters of the service primitive are as shown in Table 21.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 6.11, step b).

8.4.5 Sec-AL-EndSession.confirm

The service primitive Sec-AL-EndSession.confirm 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

The service primitive `App-Sess-EnableProxy.request` 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.

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

The parameters of the service primitive are as shown in Table 22.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (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 specified 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 6.10.4 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 service primitive `Sec-Sess-Configure.request` is specified 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)
)
```

The parameters of the service primitive `Sec-Sess-Configure.request` are as shown in Table 23.

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

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 6.10.
– 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 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 needs to be matched by the peer's IEEE 1609.2 certificate presented in the handshake. A match of any entry in this array is acceptable. See IEEE 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
– 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

Table 23 *(continued)*

Name	Type	Valid range	Description
'Max Incoming Sessions'	Integer	Any positive integer	The maximum number of incoming client requests to accept during the lifetime of this session instance. Only client requests that pass all other checks count against this total
– The following parameters are provided only if 'Session Type' is External and are optional even in that case. They represent constraints on the peer certificate presented during the secure session handshake. If the parameters are not provided, no constraints are applied to the indicated field in the certificate.			
'Name Constraints'	A regular expression		If this parameter is provided, the ID field in the IEEE 1609.2 certificate needs to be of type name and needs to match the regular expression provided. See IEEE 1609.2 for further details
'Issuer Constraints'	A HashedId8 as specified in IEEE 1609.2.		If this parameter is provided, the issuer field in the IEEE 1609.2 certificate shall match the HashedId8 provided. See IEEE 1609.2 for further details
– The following parameters are provided only if 'Proxied' is True –			
'Proxying Session ID'	Integer	Any	Provided only if 'Role' is Client. The ID to be provided as the Proxying Session parameter to `AL-Sess-ClientHelloProxy.indication`
'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 specified in this document; it is simply used as an identifier.

9.3.1.3 Effect of receipt

On receipt, the secure session services create a secure session instance with the indicated parameters as specified in 6.4. If the parameter role is "Client", the instance attempts to carry out the handshake.

9.3.2 Sec-Sess-Configure.confirm

The service primitive `Sec-Sess-Configure.confirm` confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

9.3.3 Sec-Sess-Start.indication

The service primitive `Sec-Sess-Start.indication` is used by an instance of the secure session services to indicate that a handshake has taken place and to provide the certificate from the handshake to the security subsystem.

```
Sec-Sess-Start.indication (
   Application ID,
   Session ID,
   Certificate (optional)
)
```

The parameters of the service primitive are as shown in Table 24.

Table 24 — Parameters for Sec-Sess-Start.indication

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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
'Certificate'	An IEEE 1609.2 end-entity certificate	Any IEEE 1609.2 end-entity certificate	The certificate received from the peer trusted entity during the handshake. Need not be provided if 'Require Client Authentication' was set to False in the corresponding `Sec-Sess-Configure.request`. See IEEE 1609.2 for further details

On receipt, the security subsystem carries out the activities specified in 6.6, steps b) and c).

NOTE Although only the end-entity certificate from the handshake is passed to the security subsystem in this primitive, in order for the handshake to have completed correctly the entire certificate chain will be available to the Security Services specified in IEEE 1609.2. Any certificate in the chain can therefore be used by the security subsystem when applying the access control policy as in 6.6, step b). See 9.5 for further discussion of how the full chain is obtained during the handshake.

9.3.4 Sec-Sess-EndSession.indication

The service primitive Sec-Sess-EndSession.indication is used to indicate that the secure session services have detected that a particular secure session has ended, either because of a timeout or because the peer trusted entity has terminated the session.

```
Sec-Sess-EndSession.indication (
   Application ID,
   Session ID
)
```

The parameters of the service primitive are as shown in Table 25.

Table 25 — Parameters for Sec-Sess-EndSession.indication

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 takes the actions specified in 6.12, step b).

9.3.5 Sec-Sess-Deactivate.request

The service primitive `Sec-Sess-Deactivate.request` is used to request that a secure session instance is deactivated, i.e. that it stops initiating or responding to new session handshakes and that it deletes state such as the cryptomaterial handle reference used for authentication within handshakes.

```
Sec-Sess-Deactivate.request (
   Application ID,
   secure session Instance ID
)
```

The parameters of the service primitive are as shown in Table 26.

Table 26 — Parameters for Sec-Sess-Deactivate.request

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 secure session service carries out the activities specified in 6.13, step b).

9.3.6 Sec-Sess-Deactivate.confirm

The service primitive `Sec-Sess-Deactivate.confirm` confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

9.4 AL-Sess interface

9.4.1 AL-Sess-Data.request

The service primitive AL-Sess-Data.request is used by the adaptor layer to submit an ALPDU for transmission across the secure session.

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

The parameters of the service primitive are as shown in Table 27.

Table 27 — Parameters for AL-Sess-Data.request

Name	Type	Valid range	Description
'Application ID'	Structure. See 5.5 (`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 `Iso21177AdaptorLayerPdu`	The APDU to be sent

On receipt, the secure session services send the data over the indicated secure session instance. Mechanism-specific considerations are given in 9.5.

9.4.2 AL-Sess-Data.confirm

The service primitive `AL-Sess-Data.confirm` confirms receipt of the corresponding request primitive.

This service primitive has no parameters.

No effect on receipt is specified.