INTERNATIONAL STANDARD

ISO/IEC 24727-4

First edition
2008-11-01

Identification cards — Integrated circuit card programming interfaces —

Part 4: Application programming interface (API) administration

Cartes d'identification — Interfaces programmables de cartes à puce — Partie 4: Administration d'interface de programmation (API)

Reference number
ISO/IEC 24727-4:2008(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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

ISO/IEC 24727-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identification*.

ISO/IEC 24727 consists of the following parts, under the general title *Identification cards — Integrated circuit card programming interfaces*:

— *Part 1: Architecture*

— *Part 2: Generic card interface*

— *Part 3: Application interface*

— *Part 4: Application programming interface (API) administration*

— *Part 5: Testing*

— *Part 6: Registration authority procedures for the authentication protocols for interoperability*

Introduction

ISO/IEC 24727 is a set of programming interfaces for interactions between integrated circuit cards (ICCs) and external applications to include generic services for multi-sector use. The organization and the operation of the ICCs conform to ISO/IEC 7816-4.

ISO/IEC 24727 is relevant to ICC applications desiring interoperability among diverse application domains. ISO/IEC 7498-1:1994 is used as the layered architecture of the client-application to card-application connectivity. That is, the client-application, through the application interface, assumes that there is a protocol stack through which it will exchange information and transactions among card-applications using commands conveyed through the message structures defined in ISO/IEC 7816. The semantics of action requests through the interface defined in ISO/IEC 24727-3 refers to application protocol data units (APDUs) as characterized through the interface defined in ISO/IEC 24727-2, and in the following International Standards:

— ISO/IEC 7816-4:2005, *Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange*

— ISO/IEC 7816-8:2004, *Identification cards — Integrated circuit cards — Part 8: Commands for security operations*

— ISO/IEC 7816-9:2004, *Identification cards — Integrated circuit cards — Part 9: Commands for card management*

The goal of ISO/IEC 24727 is to maximize the applicability and solution space of software tools that provide application interface support to card-aware client-applications. This effort includes supporting the evolution of card systems as they become more powerful, peer-level partners with existing and future applications while minimizing the impact to existing solutions conforming to ISO/IEC 24727.

By conforming to this part of ISO/IEC 24727, interoperable implementations of ISO/IEC 24727-3 and ISO/IEC 24727-2 can be realized. Implementation details are not defined within this part of ISO/IEC 24727; it is assumed that an implementation conforms to an accepted security policy. The specific security policy is outside the scope of ISO/IEC 24727.

Identification cards — Integrated circuit card programming interfaces —

Part 4:
Application programming interface (API) administration

1 Scope

ISO/IEC 24727 defines a set of programming interfaces for interactions between integrated circuit cards and external applications to include generic services for multi-sector use.

This part of ISO/IEC 24727 standardizes the connectivity and security mechanisms between the client-application and the card-application.

It specifies API-Administration of service-independent and implementation-independent ISO/IEC 24727 compliant modules, including security, that enables action requests to a specific card-application of an ICC such that, when coupled to data model and content discovery operations, the card-application can be used by a variety of client-applications.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 7816-4:2005, *Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange*

ISO/IEC 9797-1:1999, *Information technology — Security techniques —Message Authentication Codes (MACs) —Part 1: Mechanisms using a block cipher*

ISO/IEC 24727-1, *Identification cards — Integrated circuit card programming interfaces — Part 1: Architecture*

ISO/IEC 24727-2, *Identification cards — Integrated circuit card programming interfaces — Part 2: Generic card interface*

ISO/IEC 24727-3, *Identification cards — Integrated circuit card programming interfaces — Part 3: Application interface*

IETF RFC 2246, *The TLS Protocol Version 1.0*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 24727-1, ISO/IEC 24727-2, ISO/IEC 24727-3 and the following apply.

3.1
channel
physical pathway allowing movement of information bits between a client-application and a card-application

3.2
component
executable code comprising a processing layer accessed with ISO/IEC 24727 defined application programming interfaces

3.3
confidentiality
access restricted to some defined level of differential-identity authentication

3.4
dubious-channel
channel that might allow information messages to be altered, dropped, replayed or overheard by eavesdroppers

3.5
instantiation
operational component implementation or communication channel implementation

3.6
integrity
state of immutability of information

3.7
ISO/IEC 24727 protocol stack
series of processing components connected by communication channels that connect a client-application to a card-application

3.8
loyal-channel
channel that intrinsically maintains the integrity of channel end-points along with the confidentiality, integrity and authenticity of information

3.9
loyal-platform
computing platform that is trusted to perform data transformations and communication while maintaining confidentiality, integrity and authenticity of information

3.10
loyal-stack
ISO/IEC 24727 stack in which the full stack from client-application to the integrated circuit card that contains the card-application is implemented on a single loyal-platform

3.11
path-protection-policy
specification of the security characteristics of all platforms and channels using ISO/IEC 24727 that connect the client-application to the card-application

3.12
proxy
application programming interface implementation that conveys action requests and parameters to a layer implementation located elsewhere

3.13
TC_API
application programming interface used by components of an ISO/IEC 24727 stack to effect the stack's instantiation in a network environment

3.14
trusted-channel
channel that explicitly assures the confidentiality, integrity and authenticity of information during the transfer process, independent of the characteristics of the transfer mechanism or media

3.15
trusted-path
connection between a client-application and a card-application in which all platforms and channels have security characteristics as defined by the client-application

4 Abbreviated terms

TLS	-	transport layer security
CC	-	cryptographic checksum
CG	-	cryptogram
D	-	decipher
DES	-	data encryption standard
DO	-	data object
E	-	encipher
K	-	key
Le	-	expected length
MAC	-	message authentication code
SSC	-	send sequence counter
API	-	application programming interface
APDU	-	application protocol data unit

5 Architecture specialization

ISO/IEC 24727-1, ISO/IEC 24727-2 and ISO/IEC 24727-3 define an architecture, application programming interface and a command-set communication message structure and protocol through which a client-application can access information and computation services from a card-application. The objective of ISO/IEC 24727 is to achieve interoperability among diverse implementations of card-applications and client-applications. ISO/IEC 24727-1 specifies the overarching architecture of ISO/IEC 24727. ISO/IEC 24727-2 specifies a generic request set through which card-application services may be accessed. ISO/IEC 24727-3 specifies the application interface through which client-applications shall access services provided by card-applications. The architectural overview of ISO/IEC 24727 as illustrated in Figure 1 envisions a variety of implementations that can satisfy the standard in sufficient detail to successfully conform to the testing procedures which will be specified in ISO/IEC 24727-5.

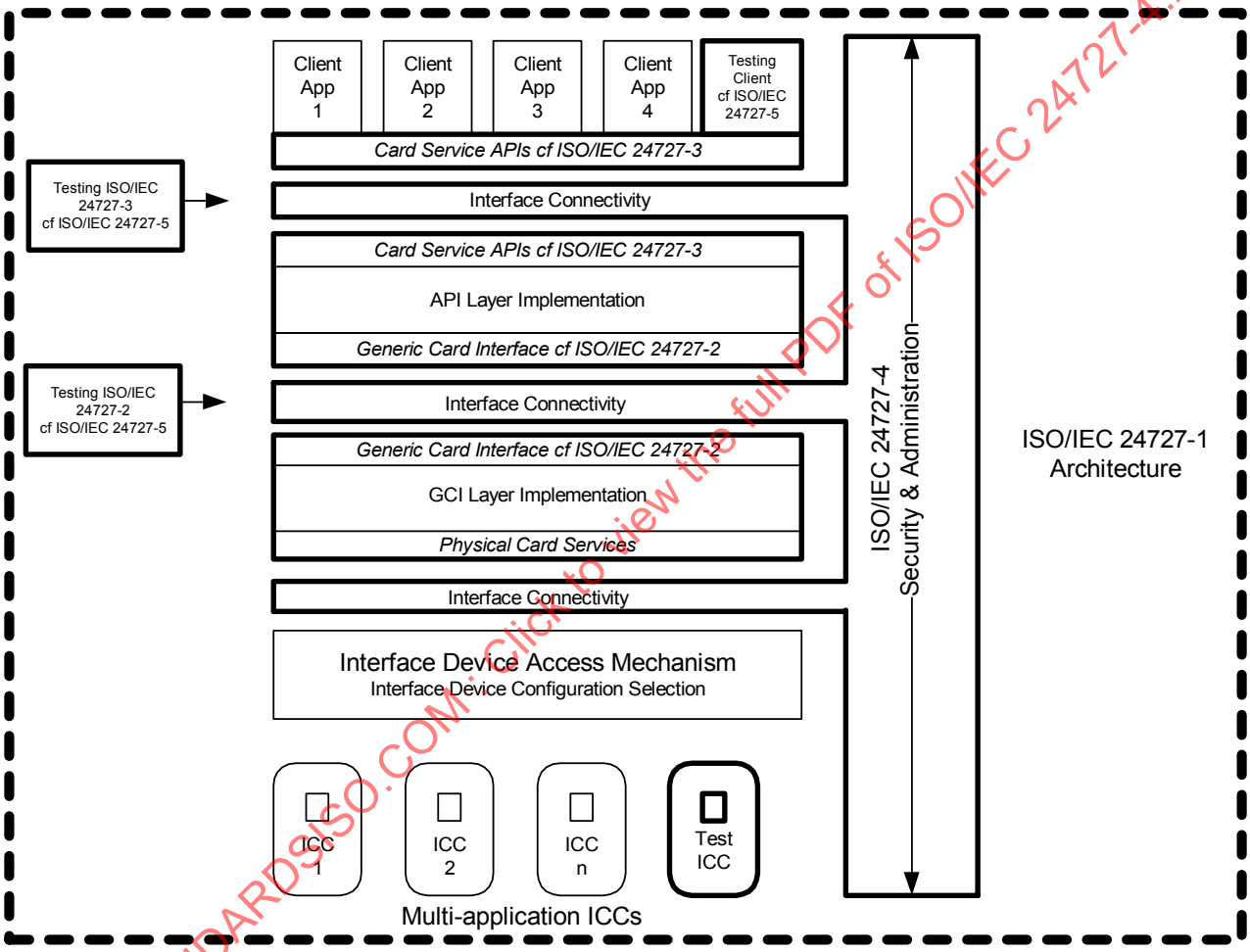


Figure 1 — Specialization of the ISO/IEC 24727 Architecture

This part of ISO/IEC 24727 details the stack instantiation and operational procedures that provide for the preparation and use of a card-application to provide information storage, retrieval and associated processing for client-applications.

This part of ISO/IEC 24727 does not mandate a specific implementation methodology, but it does provide a detailed definition of information organization and content to be supported by any conformant implementation. An ISO/IEC 24727 conformant stack shall instantiate at least one of the configurations defined in Clause 5.

This clause specifies a variety of instantiation configurations of the ISO/IEC 24727 protocol stack. The spectrum of configurations range from the full-network-stack to the loyal-stack. The opaque-ICC-stack is a

configuration in which the ISO/IEC 24727-2 instantiation shall be tightly coupled to the card-applications that it can support; tightly coupled meaning that the connection to the card-application containing ICC through operating system specific code for accessing an interface device is encompassed by the ISO/IEC 24727-2 instantiation. The remote-loyal-stack considers a configuration in which a loyal-stack shall be utilized on a platform that is remote from the client-application. The ICC-resident-stack considers a card-application that shall support the ISO/IEC 24727-3 layer implementation. Finally, the remote-ICC-stack discusses a stack in which the physical connection of the ICC shall be made to a different platform from the rest of the stack.

It is noted that all parts of ISO/IEC 24727 are neutral with respect to the physical interconnection mechanism used to complete a communication channel(s) from the client-application to the card-application. Consequently, references to ICC should be interpreted as being equally applicable to PICC or to non-card resident card-applications and that references to IFD are equally applicable to PCD or other card-application containing platform interfaces.

Figure 2 illustrates the generic elements of an ISO/IEC 24727 protocol stack.

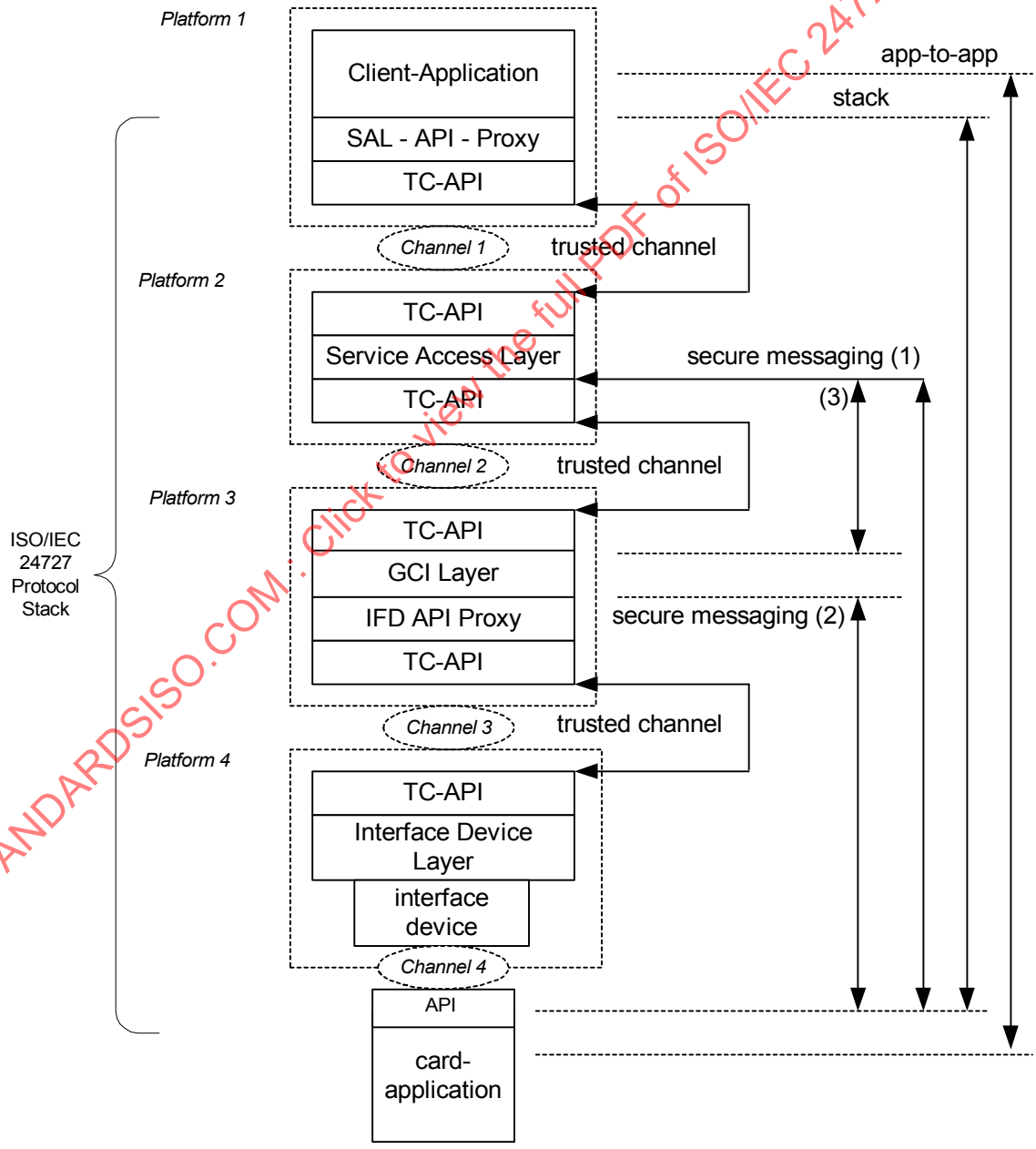


Figure 2 — Generic Elements of the ISO/IEC 24727 Stack

These elements define a general stack configuration. This general stack configuration allows a full ISO/IEC 24727 compliant stack to be segmented across a range of one to four distinct computer platforms. In Figure 2, these platforms are designated as Platform 1, 2, 3 & 4 respectively.

In a fully segmented stack, four distinct communication channels are required to connected the components that comprised the stack. These channels are designated as Channel 1, 2, 3 & 4 respectively. In Clause 5.1 through Clause 5.6, six distinct stack configurations are specified. These comprise the set of allowed ISO/IEC 24727 conformant stacks. A conformant ISO/IEC 24727 stack instantiation shall encompass at least one of these six stack specifications. A conformant ISO/IEC 24727 stack instantiation may encompass more than one of these six stack configurations.

Figure 3 provides a descriptive legend to be used in interpreting the details of the remaining figures in Clause 5.

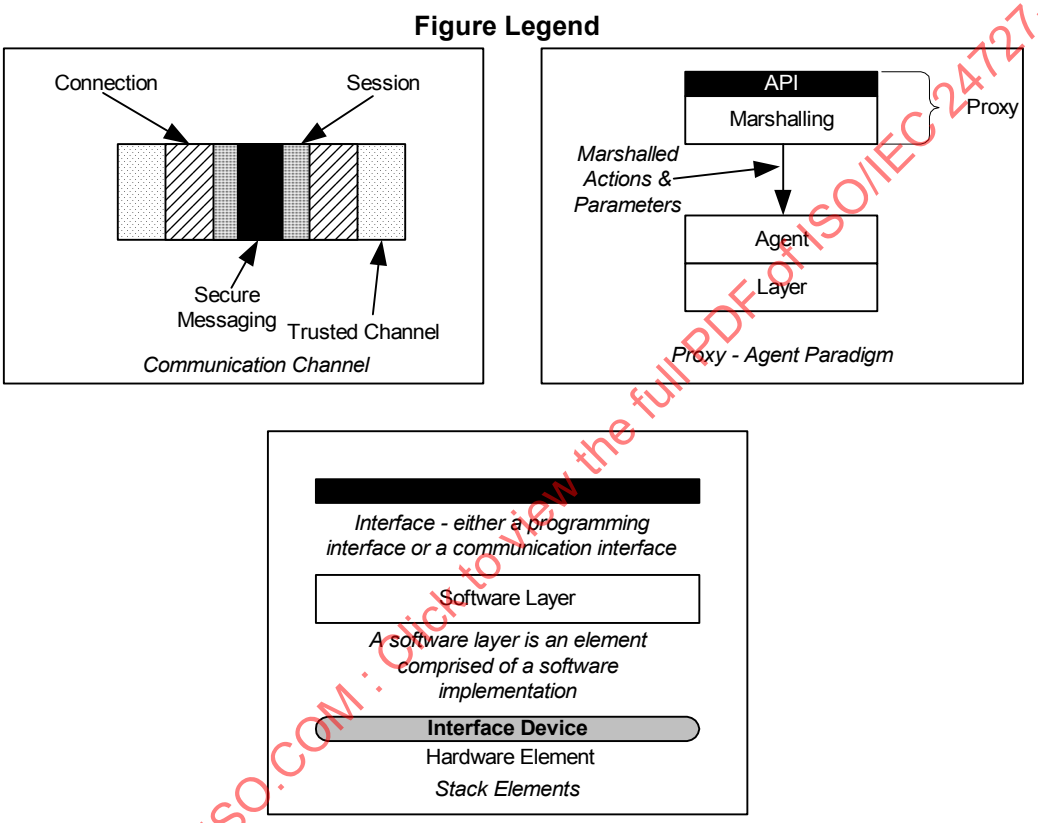


Figure 3 — Legend for Following Figures

A proxy and agent pair (of processes) allow the extension of an API from one point in a stack to a different point by conveying the action requests and associated parameters through a standard description; a procedure known as "marshalling".

5.1 Full-network-stack

A general interconnection between a client-application and a card-application allows each component of the stack to be connected to its adjacent components by way of a network connection, as illustrated in Figure 4.

The full-network-stack configuration of ISO/IEC 24727 entails a segmentation of the ISO/IEC 24727 stack into its interoperable constituent components. While it may be unlikely that such a stack configuration shall be used in a routine operational setting, by testing conformance of this configuration, a fine level of granularity of component interoperability can be confirmed.

Realization of a full-network-stack shall be effected through static network configurations of the various components. Dynamic process invocation of the components is not required by ISO/IEC 24727. The establishment of end-to-end security characteristics shall be effected through parameters passed through the ISO/IEC 24727-3 API. Stack instantiation and operation may subsequently entail out-of-band parameter passing with respect to the ISO/IEC 24727-2 interface, particular in establishing the appropriate keys to enable subsequent secure messaging between the ISO/IEC 24727-2 layer implementation and the card-application.

Corresponding stack establishment and connection (and session) security characteristics establishment shall be met by instantiations of all other stack configurations specified in the various sub-clauses of Clause 5.

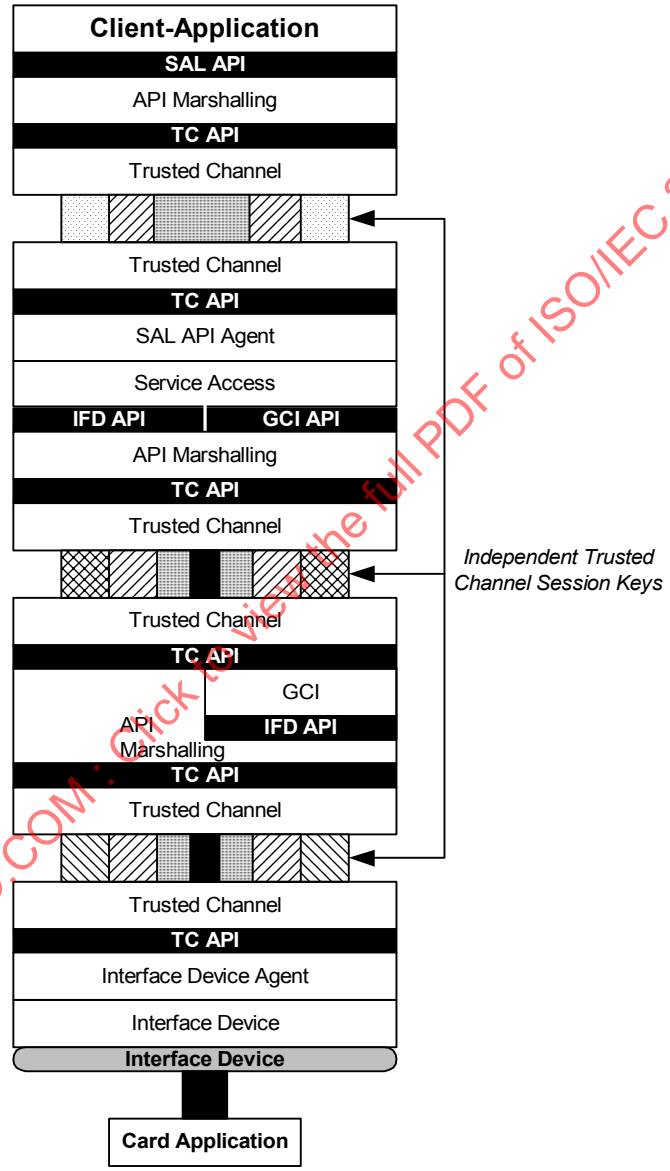


Figure 4 — Networked Connections Between Client-Application and Card-Application

In Figure 4, the client-application to SAL-API proxy connection is effected by a loyal-channel. All other communication channels indicated in the figure are dubious-channels. If the path-protection-policy invoked by the client-application specifies an increased level of security beyond a dubious-channel for any communication channel, then this level shall be achieved through the use of trusted-channels or, if applicable, through the use of secure messaging.

The ISO/IEC 24727-3 implementation, ISO/IEC 24727-2 implementation and card broker implementations all comprise components that shall be effected on trusted platforms if the entire stack is to be viewed as a secure category beyond a level of intrinsic security and discussed in Clause 6.

5.2 Loyal-stack

As illustrated in Figure 5, a loyal-stack is an implementation of the complete ISO/IEC 24727 stack on a loyal-platform, hence using loyal-channels for all connections except for any connection to an ICC via an interface device. For the operating system specific interface device API layer, an enhanced security category shall be achieved for communication via the interface device to the card-application through the use of secure messaging.

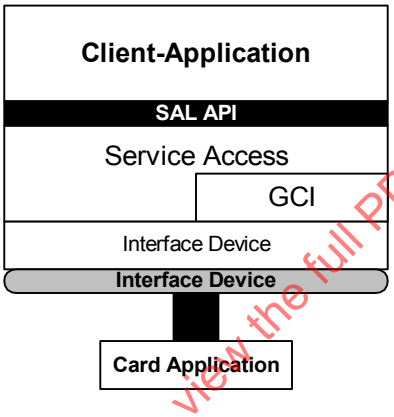


Figure 5 — Proprietary Implementations of ISO/IEC 24727-2 & ISO/IEC 24727-3 Layers

A loyal-stack shall effect either an intrinsic path-protection-policy category, as defined in Clause 6, or be completely instantiated in a loyal environment with the only possible dubious-channel being the connection of the stack to the card-application via an interface device using secure messaging as noted above. The use of the TC_API is not mandatory when running a secure channel within a Loyal Stack.

5.3 Opaque-ICC-stack

An opaque-ICC-stack incorporates the ISO/IEC 24727-2 layer instantiation and the interface device layer instantiation into a single component. This component encompasses the operating system specific connection via an interface device with the card-application. This component shall exist as a static network accessible process in which its trusted-channel layer shall present itself as the server component (see IETF RFC 2246) in the negotiation of a trusted-channel.

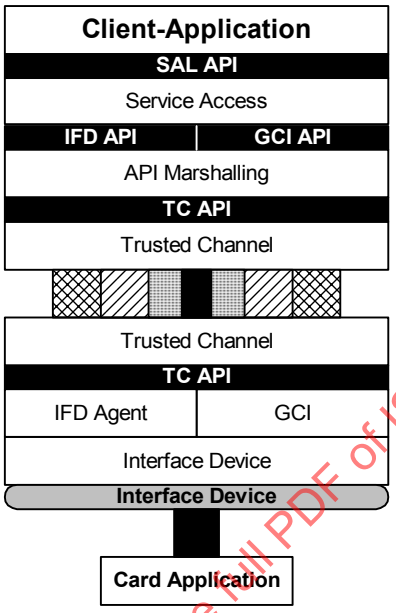


Figure 6 — Opaque ICC Stack

The keys through which to effect secure messaging between the ISO/IEC 24727-2 layer instantiation and the card-application shall be obtained through the procedural element invoked during the ISO/IEC 24727-2 layer's bootstrap operation as specified in ISO/IEC 24727-2.

5.4 Remote-loyal-stack

A remote-loyal-stack segments a loyal-stack into two distinct components, allowing the client-application to be displaced on a network from its supporting ISO/IEC 24727 stack and the card-application with which it communicates. An SAL-API proxy component is directly linked to the client-application and this proxy communicates through a trusted-channel with the remainder of the loyal-stack as illustrated in the Figure 7.

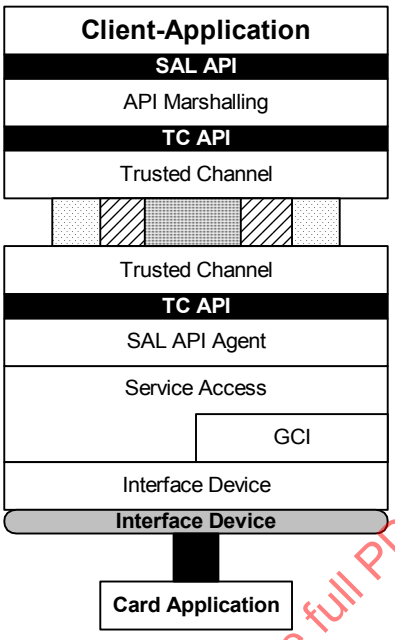
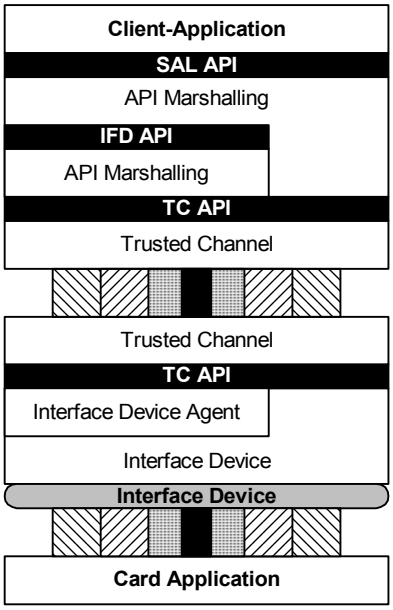


Figure 7 — Remote-loyal-stack

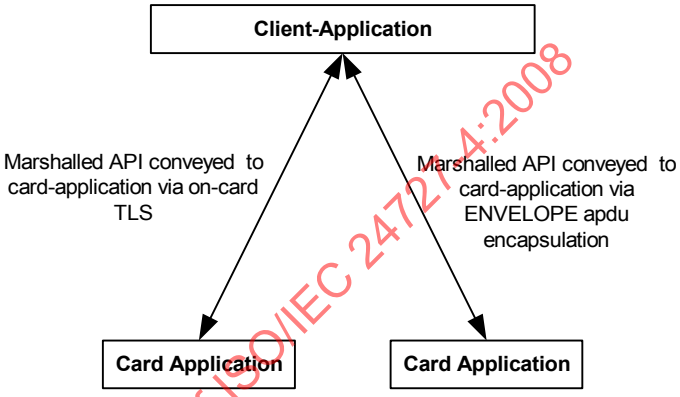
The ISO/IEC 24727-3 layer and ISO/IEC 24727-2 layer component shall exist as a static network accessible component at the time the remote-loyal-stack is instantiated by action from the client-application. This component shall exist as a static network accessible process in which its trusted-channel layer shall present itself as the server component (see IETF RFC 2246) in the negotiation of a trusted-channel with the client-application component.

5.5 ICC-resident-stack

An ICC-resident-stack provides the complete ISO/IEC 24727 stack instantiation within a card-application. The only off-card components are the SAL-API-Proxy and the interface device layer which provide syntactic, semantic and physical connectivity between the client-application and the card-application as illustrated in Figure 8. As illustrated in Figure 8, either of two distinct paths can be used in conveying the marshalled API to the client application. The appropriate path is determined through the addressing returned to the client-application by the CardApplicationPath request defined in ISO/IEC 24727-3.



Stack Configuration



Two path mechanisms between client-application and card-application

Figure 8 — ICC Resident Stack

The card-application's trusted-channel layer instantiation shall exist as a static network accessible component at the time the ISO/IEC 24727 stack is instantiated through action by the client-application. This component shall exist as a static network accessible process in which its trusted-channel layer shall present itself as the server entity (see IETF RFC 2246) in the negotiation of a trusted-channel with the client-application component. Support of TLS by the ICC is not defined by ISO/IEC 7816. The marshalled elements of the SAL API may also be conveyed to the card-application encapsulated within an ENVELOPE apdu as defined in ISO/IEC 7816-4 and as indicated in Figure 8.

5.6 Remote-ICC-stack

Figure 9 illustrates a remote-ICC-stack. In this configuration, the full ISO/IEC 24727 stack is implemented on the same platform as the client-application. The physical connection point of the card-application may be present on a different platform from the client-application with a trusted-channel providing connectivity between the two components of the complete stack.

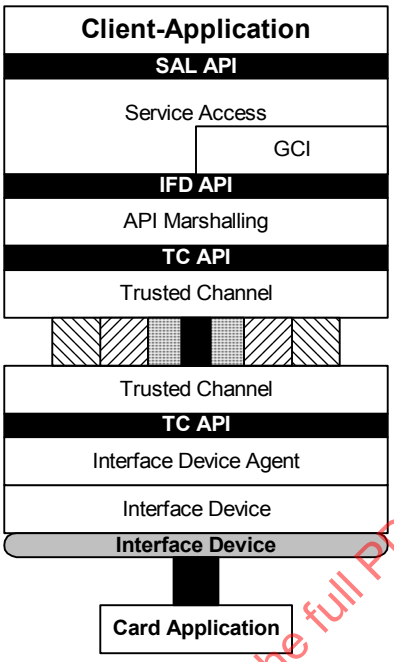


Figure 9 — Remote-ICC-Stack

In this configuration, the interface device API layer instantiation shall exist as a static network accessible component at the time of stack instantiation by action of the client-application. This component shall exist as a static network accessible process in which its trusted-channel layer shall present itself as the server entity (see IETF RFC 2246) in the negotiation of a trusted-channel with the client-application component.

6 Security architecture

Instantiation of a card-application, including the access control rules used within the card-application to secure information storage and computational process access, shall be effected by a client-application through the ISO/IEC 24727-3 API or by a completely functionally equivalent procedure. Connection to an ISO/IEC 24727 stack shall be effected by a client-application through the ISO/IEC 24727-3 API. Stack configurations to be accessed by a client-application shall be instantiated prior to access by a client-application. Security characteristics of any stacks made available shall be conveyed from the ISO/IEC 24727-3 layer to the client-application through the ISO/IEC 24727-3 API CardApplicationPath request response. It shall be the prerogative of the client-application to select the acceptable stack configuration that provides access to the desired card-application. Clause 6.1 defines the semantics to be used to describe the stack security characteristics.

6.1 Path-protection-policy

A conformant ISO/IEC 24727 stack shall effect a communication channel between a client-application and a card-application via intermediate stack components. As illustrated in Figure 2, these components and adjacent channels comprise several discrete segments. Each of these segments shall be based on a variety of

communication mechanisms, each with its own specific set of protocols specified by the stack configuration. Accordingly, the security characteristics accorded the information flowing between the client-application and the card-application shall be constrained to the (security) mechanisms specified through the path-protection-policy established for each stack configuration. The only points of channel access, regardless of stack configuration, shall be the client-application and the card application. The specification of the path-protection-policy shall be conveyed to the client-application through the ISO/IEC 24727-3 API as a parameter of the CardApplicationPath request of the Connection Service.

The coherence of the security characteristics of the communication channel between the client-application and the card-application shall be characterized by a path-protection-policy-class that shall determine the granularity of security mechanisms used to effect the channel according to the following definitions:

Path-protection-policy-classes

- end-to-end - a single key or key set shall be used to secure the channel between the client-application and the card-application.
- segmented - different keys or key sets shall be used to secure the various segments of the channel between the client-application and the card-application.
- agnostic - no specification is given for the security characteristics of the channel between the client-application and the card-application; the strength or weakness of the security characteristics of the channel is immaterial.

Within a specific path-protection-policy-class, various categories of security shall be established by a stack configuration according to the path-protection-policy-category. The path-protection-policy-category determines the specific security-facets that shall be achieved by the stack configuration and, implicitly, the mechanisms that shall be used to effect these facets, according to the following definitions

Path-protection-policy-categories

- Intrinsic - result of platform + channel default facets
- Protected - confidentiality + data integrity
- Source-authenticated - card-application authentication + Protected
- Mutually-authenticated - card-application authentication + Source-authenticated + Protected

The card-application authentication and the Source-authenticated path-protection-policy-categories shall be effected through the ISO/IEC 24727-3 CardApplicationStartSession request.

The security features to be established according to path-protection-policy-categories, and the implicit mechanisms that may be used to effect these security-facets, are defined as follows:

Characteristics & Mechanisms

- confidential – eavesdropping prevented across the specific channel segment (mechanisms)
  - confidential-trusted-channel
  - loyal-platform
  - loyal-channel
- data integrity – data integrity maintained across the specific channel segment (mechanisms)
  - MAC-trusted-channel
  - loyal-platform
  - loyal-channel
- source integrity – authentication of differential-identity used to access information (mechanisms)
  - client-application authentication (internal-auth)
  - client-application authentication (external-auth)

Due to the composition of the various stack configurations, not all path-protection-policy-classes are available on all configurations. Table 1 illustrates the most stringent classes that can be achieved on the various stack configurations defined in Clause 5.

Stack Configuration	Protected	Client-Source	Mutual-Auth
Loyal	end-to-end	end-to-end	end-to-end
Full-Network	segmented	segmented	segmented
Opaque-ICC	segmented	segmented	segmented
Remote-Loyal	segmented	segmented	segmented
ICC-Resident	end-to-end	end-to-end	segmented
Remote-ICC	end-to-end	end-to-end	segmented

Table 1 — Path-protection-policy-classes per category per stack configuration

In Table 1, "protected" refers to the establishment of data privacy (meaning protection from eavesdroppers) and data integrity (meaning protection from any change in data values) within the various stack configurations. "Client-source" refers to the establishment of an authentication state between the client-application and the card-application or between the card-application and the client-application. "Mutual-auth" refers to the simultaneous establishment of authentication states in both the client-application and the card-application. Each element of the table indicates the highest level of path-protection-policy-class that can be achieved for a specific path-protection-policy-category through a specific type of stack configuration.

Establishing a path between a client-application and a card-application is achieved by the client-application selecting an acceptable stack configuration that is included as one element the available path(s) returned by the CardApplicationPath request and then performing a CardApplicationConnect request to establish a connection to the card-application. More stringent security characteristics can then be establish by performing a CardApplicationStartSession with the desired authentication protocol specified.

6.2 ACL – ACR mapping

All access rules defined by access control lists shall be enforced by any conformant ISO/IEC 24727-3 layer implementation. It is expected that such enforcement will be accomplished by establishing access rules within the card-application using mechanisms presented by an ISO/IEC 24727-2 layer implementation. This does not preclude these AR enforcement mechanisms from being augmented by on-ICC or on-PICC mechanisms where available.

6.3 Secure messaging

Any path-protection-policy that references secure messaging shall use Annex A mechanisms. Secure messaging shall use common keys at each terminus of the secure messaging pathway in order to compute the necessary cryptograms required by the Annex A mechanisms. Secure messaging can only be implemented at the point within the stack at which all communication with the card-application has been reduced to ISO/IEC 7816-4 compliant APDUs. This point is realized within either the ISO/IEC 24727-3 layer implementation or the ISO/IEC 24727-2 layer implementation. In order to effect secure messaging, common keys shall be available at this point as well as within the card-application. The key(s) used to effect secure messaging within an ISO/IEC 24727-2 layer instantiation may either be intrinsic to the layer or derived by the procedural elements during the bootstrap operation defined in ISO/IEC 24727-2. The key(s) used to effect secure messaging within an ISO/IEC 24727-3 layer instantiation shall be provided by the client-application directly, or through the successful execution of an appropriate authentication protocol.

Any conformant ISO/IEC 24727-2 layer instantiation that presents a capability to perform secure messaging shall reside on a loyal-platform.

When an APDU is protected by secure messaging, it may be transmitted as such at the GCI according to ISO/IEC 24727-2, or as an argument of a REQUEST or CONFIRMATION at the IFD-API.

6.4 Trusted-channel key administration

Security derived from the use of a trusted-channel shall be independent of any secure messaging, or of any application-to-application security mechanisms used via the ISO/IEC 24727-3 session between a client-application and a card-application. The trusted-channel encryption mechanisms, including all key administration, shall be independent of any differential-identity mechanisms used within a card-application.

7 Connection components

The general architecture defined in ISO/IEC 24727-1 identifies a number of discrete components that can be accessed through interfaces specified by the ISO/IEC 24727 standard.

All ISO/IEC 24727 conformant interfaces shall be uniquely identified through the version numbers established by the ASN.1 specifications of the interfaces.

7.1 Action request and response semantics

All stack requests from the ISO/IEC 24727-3 API to the card-application shall occur synchronously.

7.2 Proxy – Agent Architecture

ISO/IEC 24727-3 defines an extensible API, the Service Access Layer API, through which a client-application shall derive services from a remotely located card-application. This API may be instantiated as a component forming a proxy that shall be directly linked to the client-application through a language specific binding based on the implementation language of the client-application. In a similar fashion, the IFD API defined in Clause 7 can also implemented through a standard proxy – agent facility as illustrated in Figure 10.

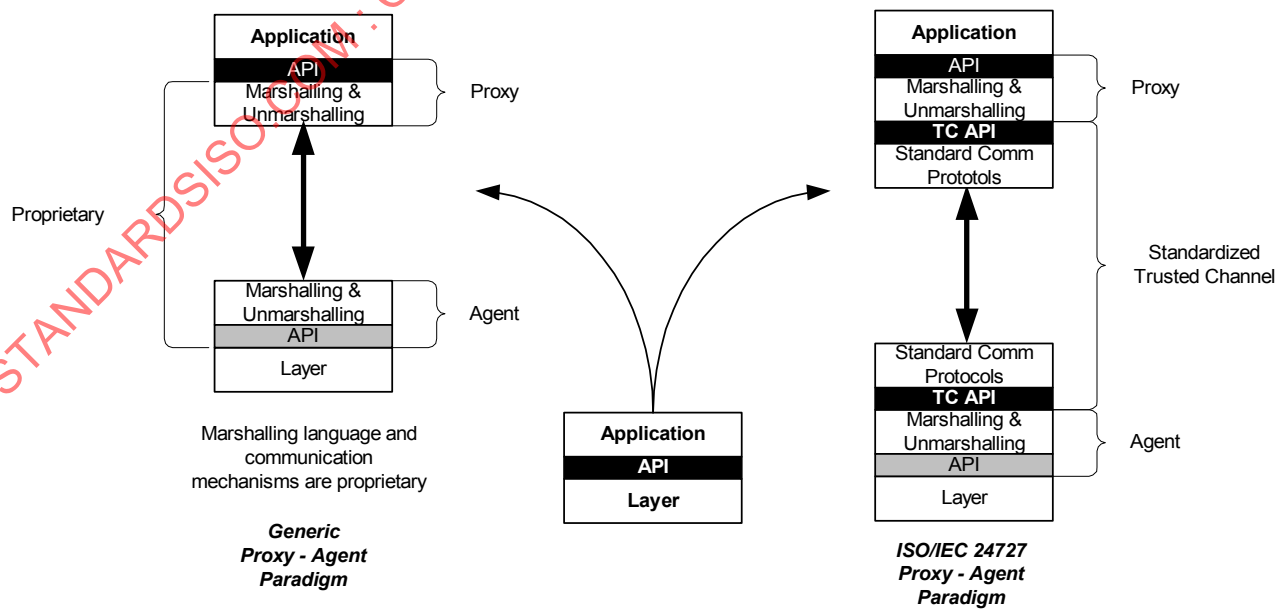


Figure 10 — Proxy – Agent Facilities

As indicated in the figure, the functions of the SAL-API proxy shall be to (a) provide a language specific binding of the ISO/IEC 24727-3 API for direct linkage to the client-application, (b) marshal all aspects of the ISO/IEC 24727-3 API into an ASN.1 data stream, and (c) project this data stream into the ISO/IEC 24727 stack through a trusted- channel. This trusted-channel allows the data stream to be delivered to any other stack component at any location on the network. Since this delivery is done through a trusted-channel, the security of the overall ISO/IEC 24727 stack is maintained by this component. In appropriate stack configurations, similar facilities can be used for the IFD API.

7.3 Trusted-channel Interface

ISO/IEC 24727-1 defines an architecture that allows for various components of the ISO/IEC 24727 protocol stack to be distributed across a wide-area network connection between a client-application and a card-application. This clause defines an API that shall be used by various components to effect a connection across a general network. This API shall provide access to communication facilities through the various trusted channel protocols identified in Annex A.2. A conformant ISO/IEC 24727 stack configuration that uses a trusted channel established through this API shall support at least the TLS protocol as specified in IETF RFC 2246.

API Function	Functional Description
TC_API_Open	This function initiates the handshake through which the client and server orientation is established between the two ends of the channel and through which the security characteristics of the channel are established.
TC_API_Close	This function terminates the trusted-channel.
TC_API_Write	This function transfers a message through the trusted-channel to the terminus point at the other end of the trusted-channel.
TC_API_Read	This function accepts a message from the trusted-channel.
TC_API_Reset	This function flushes any pending messages in the trusted-channel and re-initialises the trusted-channel.
TC_API_GetStatus	This function retrieves the current status of the trusted-channel, including the status of any pending messages in the trusted-channel.

Table 2 — Trusted Channel API

7.3.1 TC_API_Open request

7.3.1.1 Purpose

The TC_API_Open request initiates the trusted-channel handshake phase.

7.3.1.2 Action

```
OUT     status_code         TC_API_Open(
IN      octet string        remoteAddress,
IN      octet string        channelParams,
OUT     struct_handle       channelHandle
                            );
```

7.3.1.3 Parameters

remoteAddress	address of remote channel node
channelParams	security and connection parameters including an OID to designate the specific protocol as defined in Annex A
channelHandle	opaque handle for the opened channel

7.3.1.4 Prerequisites

NONE

7.3.1.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_NODE_NOT_REACHABLE
```

7.3.1.6 Impact on current state

On successful completion, a bi-directional channel will be established to the specified remote network address.

7.3.2 TC_API_Close request

7.3.2.1 Purpose

The TC_API_Close request terminates the trusted-channel.

7.3.2.2 Action

```
OUT      status_code          TC_API_Close(
IN       struct_handle        channelHandle
                              );
```

7.3.2.3 Parameters

```
channelHandle        handle of the currently open channel
```

7.3.2.4 Prerequisites

NONE

7.3.2.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_UNKNOWN_HANDLE
```

7.3.2.6 Impact on current state

On successful completion of this request, the currently open channel will be closed.

7.3.3 TC_API_Read request

7.3.3.1 Purpose

The TC_API_Read request retrieves a message from the trusted-channel.

7.3.3.2 Action

```
OUT      status_code          TC_API_Read(
IN       struct_handle        channelHandle,
OUT      octet string         message
                              );
```

7.3.3.3 Parameters

channelHandle opaque handle of the open channel

message buffer containing the message read from channel

7.3.3.4 Prerequisites

NONE

7.3.3.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_WARNING_BUFFER_LENGTH_EXCEEDED
API_UNKNOWN_HANDLE
```

7.3.3.6 Impact on current state

On the successful completion of this request, the buffer "message" contains a series of bytes read from the open channel.

7.3.4 TC_API_Write request

7.3.4.1 Purpose

The TC_API_Write request places a message structure into the trusted-channel.

7.3.4.2 Action

```
OUT       status_code          TC_API_Write(
IN        struct_handle        channelHandle,
IN        octet string         message
                               );
```

7.3.4.3 Parameters

channelHandle	opaque handle to the open channel
message	buffer containing string of bytes

7.3.4.4 Prerequisites

NONE

7.3.4.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_UNKNOWN_HANDLE
```

7.3.4.6 Impact on current state

On successful completion of this request, the message buffer has been written to the channel.

7.3.5 TC_API_Reset request

7.3.5.1 Purpose

The TC_API_Reset request purges any pending messages in the trusted-channel and reinitialises the channel to a "post-handshake" state.

7.3.5.2 Action

```
OUT       status_code          TC_API_Reset(
IN        struct_handle        channelHandle
                               );
```

7.3.5.3 Parameters

```
channelHandle          opaque handle for the open channel
```

7.3.5.4 Prerequisites

NONE

7.3.5.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_UNKNOWN_HANDLE
```

7.3.5.6 Impact on current state

On successful completion of this request, the channel is in the immediate post-handshake state with no pending messages.

7.3.6 TC_API_GetStatus request

7.3.6.1 Purpose

The TC_API_GetStatus request returns the current state of the trusted-channel.

7.3.6.2 Action

```
OUT     status_code         TC_API_GetStatus(
IN      struct_handle       channelHandle,
OUT     octet string        statusString
                            );
```

7.3.6.3 Parameters

channelHandle	opaque handle for the open channel
statusString	current state of the channel

7.3.6.4 Prerequisites

NONE

7.3.6.5 Return codes

```
API_OK
API_TIMEOUT_ERROR
API_UNKNOWN_ERROR
API_UNKNOWN_HANDLE
```

7.3.6.6 Impact on current state

On successful completion of this request, the current state of the open channel is returned in the statusString parameter.

7.4 Interface Device API

The Interface Device API comprises the following group of requests, which are specified in a generic manner:

— Slot terminal related requests
— Slot related requests
— User related requests

Slot terminal related requests

The Interface Device API contains the following slot terminal related requests:

— EstablishContext
— ReleaseContext
— ListIFDs
— GetIFDCapabilities
— GetStatus
— Wait
— Cancel
— ControlIFD

Slot related requests

The Interface Device API contains the following slot related functions:

— Connect
— Disconnect
— BeginTransaction
— EndTransaction
— Transmit

User related requests

The Interface Device API contains the following user related requests:

— VerifyUser
— ModifyVerificationData
— Output

The Interface Device API defined in this clause shall only be accessed from within a SAL layer or a GCI layer. If the return code from any action request is different from IFD_OK, then output parameters may be absent or not correctly set.

7.4.1 Establish Context

A slot terminal related request.

7.4.1.1 Purpose

The `EstablishContext` request creates a context through which further commands can be sent to the IFD-Layer.

7.4.1.2 Action

```
OUT     Status                  EstablishContext (
IN      ChannelHandleType       ChannelHandle OPTIONAL,
OUT     ContextHandleType       ContextHandle
                        );
```

7.4.1.3 Parameters

`ChannelHandle` is an optional parameter, which may be used to address an established channel to a remote system. The `ChannelHandle` is obtained using the function `TC_API_Open` defined in ISO/IEC 24727-4. If the call is addressed to the local system the parameter `ChannelHandle` may be omitted.

`ContextHandle` is the returned handle which may be used in other calls to address the established context with the IFD-Layer.

7.4.1.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CHANNEL_HANDLE`	The provided ChannelHandle is invalid.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.2 ReleaseContext

A slot terminal related request.

7.4.2.1 Purpose

The `ReleaseContext` request releases an established context with the IFD-Layer.

7.4.2.2 Action

```
OUT     Status                  ReleaseContext (
IN      ContextHandleType       ContextHandle
                                );
```

7.4.2.3 Parameters

`ContextHandle` is the handle to the context which shall be released.

7.4.2.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.3 ListIFDs

A slot terminal related request.

7.4.3.1 Purpose

The `ListIFDs` request returns a list of currently available Interface Devices (IFD) on the platform.

7.4.3.2 Action

```
OUT     Status                  ListIFDs(
IN      ContextHandleType       ContextHandle,
OUT     string                  IFDName[ ]
                                );
```

7.4.3.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`IFDName` is the unique name of the IFD, which is used to address the specific IFD.

7.4.3.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.4 GetIFDCapabilities

A slot terminal related request.

7.4.4.1 Purpose

The `GetIFDCapabilities` request returns information about the capabilities of a specific IFD and its associated functional units.

7.4.4.2 Action

```
OUT     Status               GetIFDCapabilities(
IN      ContextHandleType    ContextHandle,
IN      string               IFDName,
OUT     IFDCapabilitiesType  IFDCapabilities
                             );
```

7.4.4.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`IFDName` is the unique name of the IFD, which is used to address the specific IFD.

`IFDCapabilities` contains information about the capabilities of the given IFD and its functional units. It is of type `IFDCapabilitiesType`, which is structured as follows:

```
structure
{
    SlotCapabilityType              SlotCapability[ ],
    DisplayCapabilityType               DisplayCapability[ ],
    KeyPadCapapbilityType               KeyPadCapability[ ],
    BioSensorCapabilityType             BioSensorCapability[ ],
    BooleanType                         OpticalSignalUnit,
    BooleanType                         AcousticSignalUnit
} IFDCapabilitiesType
```

`SlotCapability` is present for every existing slot of the IFD and contains information about this slot. It is of type `SlotCapabilityType`, which is structured as follows:

```
structure
{
     nonNegativeInteger       Index,
     BooleanType              ContactBased
} SlotCapabilityType
```

`Index` – is the index of the slot ranging from 0 to number of slots minus 1.

`ContactBased` – indicates whether the slot is for contact-based or contactless cards.

`DisplayCapability` is present for every existing display of the IFD and contains information its capabilities. It is of type `DisplayCapabilityType`, which is structured as follows:

```
structure
{
      nonNegativeInteger  Index,
      nonNegativeInteger  Lines,
      nonNegativeInteger  Columns,
      nonNegativeInteger  VirtualLines
                             OPTIONAL,
      nonNegativeInteger  VirtualColumns
                             OPTIONAL
}DisplayCapabilitiesType
```

`Index` – is the index of the display ranging from 0 to number of displays minus 1.

`Lines` – contains the number of visible lines supported by the display of the card terminal.

`Columns` – contains the number of visible columns supported by the display of the card terminal.

`VirtualLines` – optionally contains the number of virtual lines supported by the display of the card terminal via scrolling.

`VirtualColumns` – optionally contains the number of virtual columns supported by the display of the card terminal via panning.

`KeyPadCapability` is present for every existing key pad of the IFD and contains information its capabilities. It is of type `KeyPadCapabilityType`, which is structured as follows:

```
structure
{
   nonNegativeInteger  Index,
   positiveInteger     Keys
}KeyPadCapabilitiesType
```

`Index` – is the index of the key pad ranging from 0 to number of key pads minus 1.

`Keys` – contains the number keys of the key pad.

`BioSensorCapability` is present for every existing biometric sensor of the the IFD and contains information about the capabilities of this biometric sensor. It is of type `BioSensorCapabilityType`, which is structured as follows:

```
structure
{
   nonNegativeInteger  Index,
   nonNegativeInteger  BiometricType
} BioSensorCapabilityType
```

`Index` – is the index of the biometric sensor ranging from 0 to number of biometric sensors minus 1.

`BiometricType` – indicates the type of the biometric sensor as defined in ISO/IEC 19784-1:2006, 7.8.

`OpticalSignalUnit` indicates whether an optical signal unit (e.g. LED) is available in the card terminal.

`AcousticSignalUnit` indicates whether an acoustic signal unit (e.g. beep) is available in the card terminal.

7.4.4.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided `ContextHandle` is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.
`API_UNKNOWN_ERROR`	There was some unknown error.

7.4.5 GetStatus

A slot terminal related request.

7.4.5.1 Purpose

The `GetStatus` request returns the current status of an Interface Device and its associated functional units.

7.4.5.2 Action

```
OUT     Status                GetStatus(
IN      ContextHandleType     ContextHandle,
IN      string                IFDName OPTIONAL,
OUT     IFDStatusType         IFDStatus [ ]
                              );
```

7.4.5.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`IFDName` optionally indicates the name of a specific IFD for which the status shall be returned. If this parameter is omitted the status of all existing IFDs is returned.

`IFDStatus` may be present for the particular IFD or all IFDs and contains the current status of an IFD. It is of type `IFDStatusType`, which is structured as follows:

```
structure
{
   String                  IFDName,
   BooleanType             Connected OPTIONAL,
   SlotStatusType          SlotStatus [ ],
   BooleanType             ActiveAntenna OPTIONAL,
   SimpleFUStatusType      DisplayStatus [ ],
   SimpleFUStatusType      KeyPadStatus [ ],
   SimpleFUStatusType      BioSensorStatus [ ]
} IFDStatusType
```

`Connected` contains the information whether there is currently a connection to the IFD. If the terminal is directly connected to the host (via RS232, USB etc.) the parameter may be omitted.

`SlotStatus` is present for every existing slot of the IFD and contains information about its current status. It is of type `SlotStatusType`, which is structured as follows:

```
structure
{
    nonNegativeInteger  Index,
    BooleanType         CardAvailable,
    OCTET STRING        ATRorATS OPTIONAL
} SlotStatusType
```

`Index` – is the index of the slot ranging from 0 to number of slots minus 1.

`CardAvailable` – is TRUE if a card is captured by the indicated slot.

ATRorATS – may contain the ATR or ATS of a captured card.

ActiveAntenna — contains the status of the RF-antenna used by the proximity coupling device(s). If there are no slots for contactless cards this parameter may be omitted.

DisplayStatus — is present for every existing display of the IFD and contains information about its current status. It is of type SimpleFUStatusType, which is structured as follows:

```
structure
{
    NonNegativeInteger  Index,
    BooleanType         Available
} SimpleFUStatusType
```

Index – is the index of the functional unit ranging from 0 to number of functional units minus 1.

Available – indicates whether the functional unit is currently busy or whether it is available for requests.

KeyPadStatus — is present for every existing key pad of the IFD and contains information about its current status. It is of type SimpleFUStatusType, which is defined above.

BioSensorStatus — is present for every existing biometric sensor of the IFD and contains information about its current status. It is of type SimpleFUStatusType, which is defined above.

7.4.5.4 Return Codes

IFD_OK	The request was successful.
IFD_TIMEOUT_ERROR	The request timed out before completion.
IFD_INVALID_CONTEXT_HANDLE	The provided ContextHandle is invalid.
IFD_UNKNOWN_IFD	The provided IFDName is unknown.
IFD_UNKNOWN_ERROR	There was some unknown error.

7.4.6 Wait

A slot terminal related request.

7.4.6.1 Purpose

The `Wait` request allows the client-application to be informed that some event at an indicated list of IFDs has occurred.

7.4.6.2 Action

```
OUT     Status                      Wait(
IN      ContextHandleType           ContextHandle,
IN      positiveInteger             TimeOut OPTIONAL,
IN      CallbackChannelHandleType   CallbackChannel OPTIONAL,
IN      IFDStatusType               IFDStatus [ ],
OUT     string                      SessionIdentifier OPTIONAL,
OUT     IFDStatusType               IFDEvent [ ]
                                );
```

7.4.6.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`TimeOut` is an optional parameter, which specifies the time in milliseconds which shall be waited for an event. If this parameter is not present, the call will wait forever or until `Cancel` is called.

`CallbackChannel` a callback function allowing an asynchronous callback from the Wait function when triggered by a preset event.

`IFDStatus` is present for every IFD, which shall be monitored and contains the currently assumed status of the IFD. It is of type `IFDStatusType`, which is defined above.

`SessionIdentifier` a session identifier that identifies the specific Wait session when the callback function is called.

`IFDEvent` returns information about the occurred event(s). While it is also of type `IFDStatusType` it should only contain the information about the occurred events. I.e. the subset of the status information which has changed.

7.4.6.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.7 Cancel

A slot terminal related request.

7.4.7.1 Purpose

The `Cancel` request tries to terminate the currently processed command at a given IFD.

7.4.7.2 Action

```
OUT     Status              Cancel (
IN      ContextHandleType   ContextHandle,
IN      string              SessionIdentifier  OPTIONAL,
IN      string              IFDName
                            );
```

7.4.7.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`SessionIdentifier` a session identifier that identifies the specific Wait session being cancelled and its associated callback function.

`IFDName` is the unique name of the IFD, which is used to address the IFD at which the currently processed command shall be cancelled.

7.4.7.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_CANCEL_NOT_POSSIBLE`	The current operation can not be cancelled.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.8 ControlIFD

A slot terminal related request.

7.4.8.1 Purpose

The `ControlIFD` request allows to send commands directly to the IFD.

7.4.8.2 Action

```
OUT     Status                  ControlIFD(
IN      ContextHandleType       ContextHandle,
IN      string                  IFDName,
IN      OCTET STRING            Command,
OUT     OCTET STRING            Response
                            );
```

7.4.8.3 Parameters

`ContextHandle`	addresses the established context with the IFD-Layer.
`IFDName`	is the unique name of the IFD to which the command shall be sent.
`Command`	is the command which shall be sent to the IFD.
`Response`	is the response which shall be returned by the IFD.

7.4.8.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CHANNEL_HANDLE`	The provided ChannelHandle is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.9 Connect

A slot related request.

7.4.9.1 Purpose

The `Connect` request connects to an ICC in a particular slot of an IFD.

7.4.9.2 Action

```
OUT     Status                Connect(
IN      ContextHandleType     ContextHandle,
IN      string                IFDName,
IN      NonNegativeInteger    Slot,
IN      BooleanType           Exclusive OPTIONAL,
OUT     SlotHandleType        SlotHandle
                              );
```

7.4.9.3 Parameters

`ContextHandle`	addresses the established context with the IFD-Layer.
`IFDName`	is the unique name of the IFD, which is used to connect to a card.
`Slot`	is the index of the slot ranging from 0 to number of slots minus 1.
`Exclusive`	indicates whether the connection to the card shall be exclusive or shared. If the parameter evaluates to `TRUE` the connection is exclusive. If the parameter evaluates to `FALSE` or if the parameter is missing, the connection to the card may be shared with other requestors.
`SlotHandle`	is a handle identifying the connection to the smart card.

7.4.9.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.
`IFD_UNKNOWN_SLOT`	The addressed slot is unknown.
`IFD_SHARING_VIOLATION`	The request was not successful, because the card is already used by another process.
`IFD_NO_CARD`	The request was not successful, because there is no card captured by the indicated slot.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.10 Disconnect

A slot related request.

7.4.10.1 Purpose

The `Disconnect` request terminates the connection to a slot and may optionally perform an additional action, like ejecting the card for example.

7.4.10.2 Action

```
OUT     Status              Disconnect(
IN      SlotHandleType      SlotHandle,
IN      string              Action OPTIONAL
                            );
```

7.4.10.3 Parameters

`SlotHandle`	is a handle identifying the connection to the smart card.
`Action`	is an optional parameter, which may specify an additional action, where the following values are possible:
`Reset`	resets the card
`Unpower`	unpowers and terminates access to the card
`Eject`	ejects the card from the reader
`Confiscate`	is used to indicate that a sophisticated commercial reader should move the card to the confiscation bin and not return it to the user

7.4.10.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_SLOT_HANDLE`	The provided SlotHandle is invalid.
`IFD_UNKNOWN_ACTION`	The requested action to be performed is unknown.
`API_UNKNOWN_ERROR`	There was some unknown error.

7.4.11 BeginTransaction

A slot related request.

7.4.11.1 Purpose

The `BeginTransaction` request starts a transaction through which a series of linked requests can be sent to the indicated slot, with rollback facilities if all of the requests are not successfully completed.

7.4.11.2 Action

```
OUT     Status              BeginTransaction(
IN      SlotHandleType      SlotHandle
                            );
```

7.4.11.3 Parameters

`SlotHandle` is a handle identifying the connection to the smart card.

7.4.11.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_SLOT_HANDLE`	The provided SlotHandle is invalid.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.12 EndTransaction

A slot related request.

7.4.12.1 Purpose

The `EndTransaction` request terminates a currently open transaction with the indicated card.

7.4.12.2 Action

```
OUT     Status            EndTransaction(
IN      SlotHandleType    SlotHandle
                          );
```

7.4.12.3 Parameters

`SlotHandle` is a handle identifying the connection to the smart card.

7.4.12.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_SLOT_HANDLE`	The provided SlotHandle is invalid.
`IFD_NO_TRANSACTION_STARTED`	Indicates that no transaction was started.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.13 Transmit

A slot related request.

7.4.13.1 Purpose

The `Transmit` request sends an APDU to the indicated card.

7.4.13.2 Action

```
OUT     Status              Transmit(
IN      SlotHandleType      SlotHandle,
IN      OCTET STRING        InputAPDU,
OUT     OCTET STRING        OutputAPDU,
                            );
```

7.4.13.3 Parameters

`SlotHandle`	is a handle identifying the connection to the smart card.
`InputAPDU`	is a byte string containing an APDU, which shall be sent to the card.
`OutputAPDU`	is a byte string containing a response APDU from the card.

7.4.13.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_SLOT_HANDLE`	The provided SlotHandle is invalid.
`IFD_UNKNOWN_ERROR`	There was some unknown error.

7.4.14 VerifyUser

A user related request.

7.4.14.1 Purpose

The `VerifyUser` request initiates the verification of a user with PIN or biometrics.

7.4.14.2 Action

```
OUT     Status                  VerifyUser(
IN      SlotHandleType          SlotHandle,
IN      InputUnitType           InputUnit,
IN      nonNegativeInteger      DisplayIndex OPTIONAL,
IN      AltVUMessagesType       AltVUMessages OPTIONAL,
IN      positiveInteger         TimeoutUntilFirstKey OPTIONAL,
IN      positiveInteger         TimeoutAfterFirstKey OPTIONAL,
IN      OCTET STRING            Template
OUT     OCTET STRING            Response
                                );
```

7.4.14.3 Parameters

`SlotHandle` is a handle identifying the connection to the smart card.

`InputUnit` indicates which input unit shall be used to capture the verification data. It is of type InputUnitType, which is defined as follows:

```
choice
{
    PinInputType            PinInput,
    BiometricInputType      BiometricInput
} InputUnitType
```

`PinInput` is used in case the user verification shall be performed with a pin (encompassing a password). It is of type `PinInputType`, which is defined as follows:

```
structure
{
 nonNegativeInteger     Index,
 PasswordAttributesType
   PasswordAttributes
} PinInputTypeType
```

`Index` is the index of the key pad which shall be used to capture the PIN.

`PasswordAttributes` specifies the format of the PIN through a structure.

```
 structure
{
 PasswordFlagsType      pwdFlagsType,
 PasswordTypeType       pwdType,
 nonNegativeInteger     minLength,
 nonNegativeInteger     storedLength,
```

```
 nonNegativeInteger     maxLength
OPTIONAL,
 PadCharType            padChar OPTIONAL,
 DateTimeType
   lastPasswordChange
                                OPTIONAL,
} PasswordAttributesType
```

The parameters specified through the PasswordAttributesType structure are defined in ISO/IEC 7816-15.

`BiometricInput` is used in case the user verification shall be performed using biometrics. It is of type `BiometricInputType`, which is defined as follows:

```
structure
{
   nonNegativeInteger  Index,
   nonNegativeInteger  BiometricSubType
} BiometricInputType
```

`Index` is the index of the biometric sensor, which shall be used to capture biometric data.

`BiometricSubType` specifies the biometric subtype as defined in ISO/IEC FDIS 19784-1:2006, 7.14.

`DisplayIndex` is the index of the display which shall be used to display the messages to guide the user. If no messages shall be displayed or the IFD is not equipped with a display this parameter may be omitted.

`AltVUMessages` is an optional parameter, which is used to explicitly specify the different messages (coded as UTF-8 according to RFC3629), which shall be displayed within the verification process. The parameter is of type `AltVMessagesType`, which is specified as follows:

```
structure
{
   string    AuthenticationRequestMessage OPTIONAL,
   string    SuccessMessage OPTIONAL,
   string    AuthenticationFailedMessage OPTIONAL,
   string    RequestConfirmationMessage OPTIONAL,
   string    CancelMessage OPTIONAL
} AltVUMessagesType
```

The different parameters may be used to explicitly specify what is to be displayed within the verification process. If some parameter is omitted an appropriate (set of) default message(s) will be used.

`TimeoutUntilFirstKey` is an optional parameter, which specifies the timeout in milliseconds until the first key is pressed.

`TimeoutAfterFirstKey` is an optional parameter, which specifies the timeout in milliseconds after the first key is pressed.

`Template`	is the template in which the verification data shall be inserted by the IFD before they are sent to the card.
`Response`	is the response from the card.

7.4.14.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_SLOT_HANDLE`	The provided SlotHandle is invalid.
`IFD_UNKNOWN_INPUT_UNIT`	There indicated input unit is unknown.
`IFD_CANCELLATION_BY_USER`	The user cancelled the operation.
`IFD_UNKNOWN_ERROR`	There was some unknown error.
`IFD_UNKNOWN_PIN_FORMAT`	The indicated PIN-format is unknown.
`IFD_UNKNOWN_BIOMETRIC_SUBTYPE`	The indicated PIN-format is unknown.

7.4.15 ModifyVerificationData

A user related request.

7.4.15.1 Purpose

The `ModifyVerificationData` request initiates the modification of the verification data (PIN or biometric template). This function may also be used for unblocking of a PIN by providing a personal unblocking key (PUK).

7.4.15.2 Action

```
OUT     Status                  ModifyVerificationData (
IN      SlotHandleType          SlotHandle,
IN      InputUnitType           InputUnit,
IN      nonNegativeInteger      DisplayIndex OPTIONAL,
IN      AltMVDMessagesType      AltMVDMessages OPTIONAL,
IN      OCTET STRING            OldReferenceData OPTIONAL,
IN      positiveInteger         TimeoutUntilFirstKey OPTIONAL,
IN      positiveInteger         TimeoutAfterFirstKey OPTIONAL,
IN      BooleanType             RepeatInput OPTIONAL,
IN      OCTET STRING            Template,
                                );
```

7.4.15.3 Parameters

`SlotHandle` is a handle identifying the connection to the smart card.

`InputUnit` indicates which input unit shall be used to capture the verification data. It is of type `InputUnitType`, which is defined above.

`DisplayIndex` is the index of the display which shall be used to display the messages to guide the user. If no messages shall be displayed or the IFD is not equipped with a display this parameter may be omitted.

`AltMVDMessages` is an optional parameter, which is used to explicitly specify the different messages (coded as UTF-8 according to RFC3629), which shall be displayed within the process of modifying the verification data. The parameter is of type `AltMVDMessagesType`, which is specified as follows:

```
structure
{
    string    AuthenticationRequestMessage OPTIONAL,
    string    SuccessMessage OPTIONAL,
    string    AuthenticationFailedMessage OPTIONAL,
    string    EnterNewAuthenticationDataMessage OPTIONAL,
    string    RepeatInputMessage OPTIONAL,
    string    ComparisonOfRepeatedDataFailed OPTIONAL,
    string    RequestConfirmationMessage OPTIONAL,
    string    CancelMessage OPTIONAL
} AltMVDMessagesType
```

The different parameters may be used to explicitly specify what is to be displayed within the process of modifying the verification data. If some parameter is omitted an appropriate (set of) default message(s) will be used.

OldReferenceData	is an optional parameter, which may be provided by the client-application to allow the (remote) initiation of the modification or unblocking of the reference data. It is assumed that in this case any formatting of the reference data is performed by the client-application. If the old reference data shall be captured by the IFD, this parameter may be omitted.
TimeoutUntilFirstKey	is an optional parameter, which specifies the time out in milliseconds until the first key is pressed.
TimeoutAfterFirstKey	is an optional parameter, which specifies the time out in milliseconds after the first key is pressed.
RepeatInput	indicates whether the user shall be forced to repeat the input of the verification data.
Template	is the template in which the verification data shall be inserted by the IFD before they are sent to the card within a CHANGE REFERENCE DATA command according to ISO/IEC 7816-4.
Response	is the response from the card.

7.4.15.4 Return Codes

IFD_OK	The request was successful.
IFD_TIMEOUT_ERROR	The request timed out before completion.
IFD_INVALID_SLOT_HANDLE	The provided SlotHandle is invalid.
IFD_UNKNOWN_INPUT_UNIT	The specified input unit is unknown.
IFD_CANCELLATION_BY_USER	The user cancelled the operation.
IFD_REPEATED_DATA_MISMATCH	The repeated identification data do not match.
IFD_UNKNOWN_PIN_FORMAT	The indicated PIN-format is unknown.
IFD_UNKNOWN_BIOMETRIC_SUBTYPE	The indicated PIN-format is unknown.
IFD_UNKNOWN_ERROR	There was some unknown error.

7.4.16 Output

A user related request.

7.4.16.1 Purpose

The `Output` request is used to display a message at the IFD.

7.4.16.2 Action

```
OUT     Status                  Output (
IN      ContextHandleType       ContextHandle,
IN      string                  IFDName,
IN      OutputInfoType          OutputInfo
                                );
```

7.4.16.3 Parameters

`ContextHandle` addresses the established context with the IFD-Layer.

`IFDName` specifies the IFD at which there should be some output.

`OutputInfo` a structure through which variable output information is conveyed.

```
structure
{
        positiveInteger     Timeout          OPTIONAL,
        nonNegativeInteger  DisplayIndex     OPTIONAL,
        string              Message          OPTIONAL,
        BooleanType         AcousticalSignal OPTIONAL,
        BooleanType         OpticalSignal    OPTIONAL
} OutputInfoType
```

`TimeOut` is an optional parameter, which specifies the time in milliseconds until the displayed message or signal will disappear. If this parameter is not present, the output will stay there forever or until Cancel is called.

`DisplayIndex` is the index of the display which shall be used to display the message. If only optical or acoustical signals shall be sent this parameter may be omitted.

`Message` contains the text which shall be displayed at the indicated display (coded as UTF-8 according to RFC3629). If only an optical or acoustical signal shall be sent the Message parameter may be omitted.

`AcousticalSignal` optionally indicates that an acoustic signal shall be set. If the IFD does not have an appropriate signalling unit this parameter may be ignored.

`OpticalSignal` optionally indicates that an optical signal shall be set. If the IFD does not have an appropriate signalling unit this parameter may be ignored.

7.4.16.4 Return Codes

IFD_OK	The request was successful.
IFD_TIMEOUT_ERROR	The request timed out before completion.
IFD_INVALID_CONTEXT_HANDLE	The provided ContextHandle is invalid.
IFD_UNKNOWN_IFD	The provided IFDName is unknown.
IFD_UNKNOWN_DISPLAY_INDEX	The specified display index is unknown.
API_UNKNOWN_ERROR	There was some unknown error.

7.4.17 SignalEvent

A function offered at the IFD_API calling level.

7.4.17.1 Purpose

A function called by the Interface Device Layer to signal the occrence of a specific IFDEvent.

7.4.17.2 Action

```
OUT     Status                  SignalEvent (
IN      ContextHandleType       ContextHandle,
IN      string                  SessionIdentifier OPTIONAL,
IN      IFDStatusType           IFDEvent[ ],
                                );
```

7.4.17.3 Parameters

`ContextHandle`	addresses the established context with the IFD-Layer.
`SessionIdentifier`	a session identifier that identifies the specific Wait session when the callback function is called.
`IFDEvent`	returns information about the occurred event(s). While it is also of type `IFDStatusType` it should only contain the information about the occurred events; i.e. the subset of the status information which has changed.

7.4.17.4 Return Codes

`IFD_OK`	The request was successful.
`IFD_TIMEOUT_ERROR`	The request timed out before completion.
`IFD_INVALID_CONTEXT_HANDLE`	The provided ContextHandle is invalid.
`IFD_UNKNOWN_IFD`	The provided IFDName is unknown.

Annex A
(normative)

Path-protection Mechanisms

A.1 Secure messaging

This clause assumes that two components A and B exchange command response pairs according to ISO/IEC 7816-3:2006, 12.1.1 via a channel C. This is shown on the left side of Figure A.1. Channel C shall become a trusted channel with the features (confidentiality, integrity, no message drop, …) according to 6.1. The trusted channel shall be implemented by two additional components A_{SM} and B_{SM} on either side of channel C as shown on the right side of Figure A.1.

Component A_{SM} shall convert command APDUs received from component A from clear mode into secured mode. Afterwards the secured command APDU shall be sent via channel C to component B_{SM}. Furthermore A_{SM} shall receive response APDUs from channel C, transforms them into clear mode and sends the clear response APDUs to component A.

With respect to A_{SM} the component B_{SM} shall perform the opposite transformations, i.e. unsecured command APDUs and secures response APDUs.

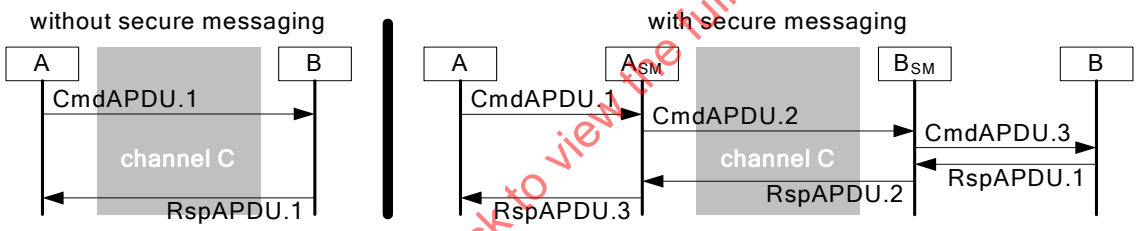


Figure A.1 — Communication with and without secure messaging

This clause contains transformation rules for

- securing a command APDU in component A_{SM}. A receiver has to perform the inverse transformation in order to get the original command APDU.
- securing a response APDU in component B_{SM}. A receiver has to perform the inverse transformation in order to get the original response APDU.

The transformation rules outlined here are a subset of the rules defined in ISO/IEC 7816-4. They apply only to interfaces where APDUs according to ISO/IEC 7816-3:2006, 12.1.1 are used for message exchange. Presently within the scope of ISO/IEC 24727, the channel between ISO/IEC 24727-3 and ISO/EC 24727-2 implementation is the only channel where such messages occur.

The transformation rules are such that with respect to the right side of Figure A.1

1. CmdAPDU.1 and CmdAPDU.3 are identical in error free operation mode.
2. RspAPDU.1 and RspAPDU.3 are identical in error free operation mode.
3. Confidentiality is ensured for CmdAPDU.1 and RspAPDU.1, this is achieved by enciphering the transferred data.
4. Integrity is ensured for CmdAPDU.1 and RspAPDU.1, this is achieved using a MAC.
5. Authenticity is ensured for CmdAPDU.1 and RspAPDU.1, this is achieved by the same mechanism as integrity.

A.1.1 Basic transformation rules

A.1.1.1 Padding

Padding shall be performed according to ISO/IEC 9797-1:1999, 6.1.2 (padding method 2).

A.1.1.2 Modification of command header

A command header CH shall consist of the bytes CLA, INS, P1 and P2. During the transformation the bits b4 and b3 in CLA shall be set. Other bits of the header shall not be changed. The result shall be indicated by CH'.

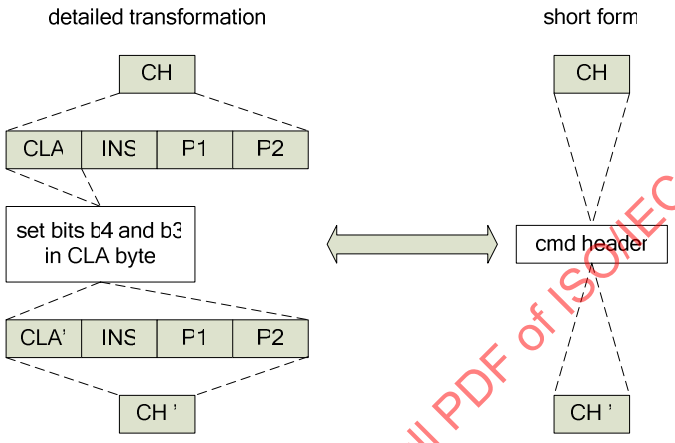


Figure A.2 — Command header transformation

This clause assumes that the CLA byte shall be in accordance with ISO/IEC 7816-4:2005, 5.1.1, Table 2 – First inter-industry values of CLA. Therefore the logical channel number, being indicated in CLA, shall be in the range zero to three. This restriction shall be in accordance with ISO/IEC 24727-2. In case future versions of this series of standards make use of logical channels with numbers greater than three then secure messaging shall be indicated in bit b6 of CLA and the original command header shall be encapsulated in a DO with tag '89' (see ISO/IEC 7816-4:2005, Table 27).

NOTE Only basic logical channel is supported at GCI.

A.1.1.3 Encipher data field

Here it is shown how a data field of a command APDU shall be enciphered to ensure confidentiality. How the block "enciphering" works shall be specified in clause A.1.1.7.

Note: padding is part of block "enciphering".

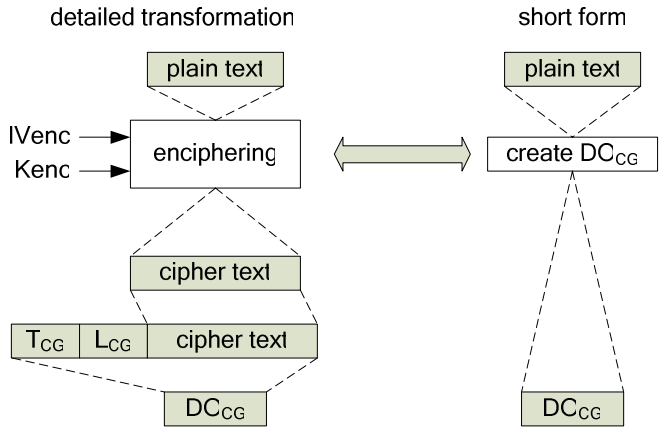


Figure A.3 — Creation of a DO containing enciphered data in case of odd INS code

A.1.1.4 Calculate DO for MAC

For MAC calculation refer to clause A.1.1.6.

Note: padding is part of block “MAC calculation”.

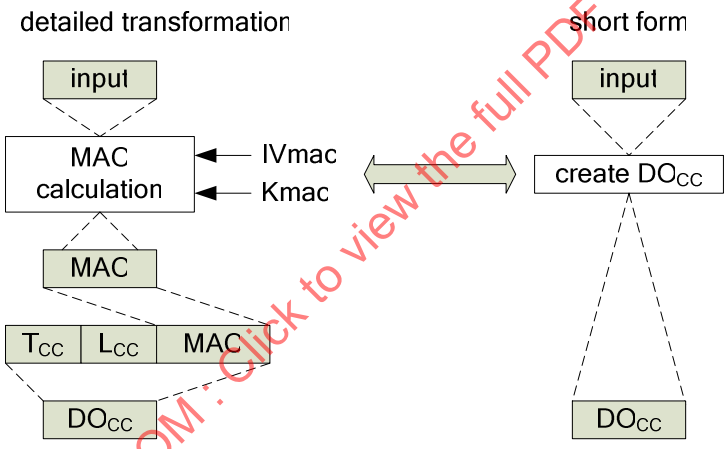


Figure A.4 — Creation of a DO containing a cryptographic checksum

A.1.1.5 Le field

This clause specifies how an Le field shall be encapsulated in a DO.

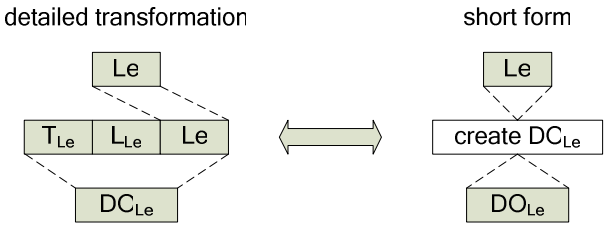


Figure A.5 — Creation of a DO containing an Le field

A.1.1.6 MAC calculation

This clause describes how an authentication code shall be calculated from the following input:

1. IVmac is an octet string containing SendSequenceCounter.
2. InputData is an arbitrary octet string of arbitrary length.
3. Kmac is a key used in the cipher algorithm.

With respect to ISO/IEC 9797-1;1999, Clause 5 the following shall be used:

- Use AES as block cipher algorithm.
- Use padding method 2 from ISO/IEC 9797-1;1999, 6.1.2.
- Use MAC algorithm 1 from ISO/IEC 9797-1;1999, 7.1 where the concatenation of IVmac and InputData is used as data string D: D = IVmac || InputData.
- The length m of MAC shall be 128 bit.

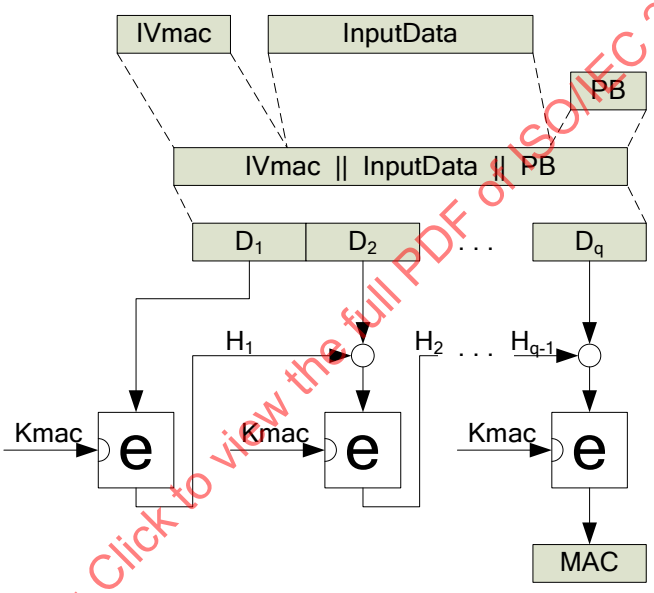


Figure A.6 — MAC calculation

A.1.1.7 Enciphering

This clause describes how a cipher text shall be calculated from the following input:

1. IVenc is an octet string containing SendSequenceCounter.
2. InputData is an arbitrary octet string of arbitrary length.
3. Kenc is a key used in the cipher algorithm.

For enciphering the following shall be used:

- Use AES as block cipher algorithm.
- Use padding method 2 from ISO/IEC 9797-1:1999, 6.1.2 in order to calculate P = InputData || PaddingString.
- Use CBC mode (Cipher Block Chaining) with initial value IVenc to calculate the cipher text from plain text P.

A.1.1.8 Status bytes

This clause specifies how status bytes shall be encapsulated in a DO.

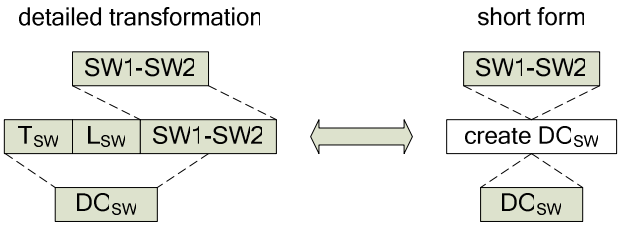


Figure A.7 — Creation of a DO containing status bytes

A.1.2 Securing a command APDU

First increment the SendSequenceCounter by one. The new value of SendSequenceCounter shall be used in steps two and four. Then the following steps shall be performed:

(1) The original command APDU CmdAPDU.1 is encapsulated in a DO with tag T_{cmd}. This DO is encapsulated in an ENVELOPE command.
(2) The command header is transformed as shown in A.1.1.2 and the command data field is encapsulated in a DO with tag T_{CG} as shown in A.1.1.3 and the Le field is encapsulated in a DO with tag T_{Le} as shown in A.1.1.5.
(3) The transformed command header is padded as shown in A.1.1.1. The result is concatenated with DO_{CG} and DO_{Le}.
(4) The result from step (3) is the input for MAC calculation.
(5) The secured command APDU has the following elements:
    a. Transformed command header CH'.
    b. The data field of the secured command APDU contains DO_{CG}, DO_{Le} and DO_{CC}.
    c. An Le field named "New Le".

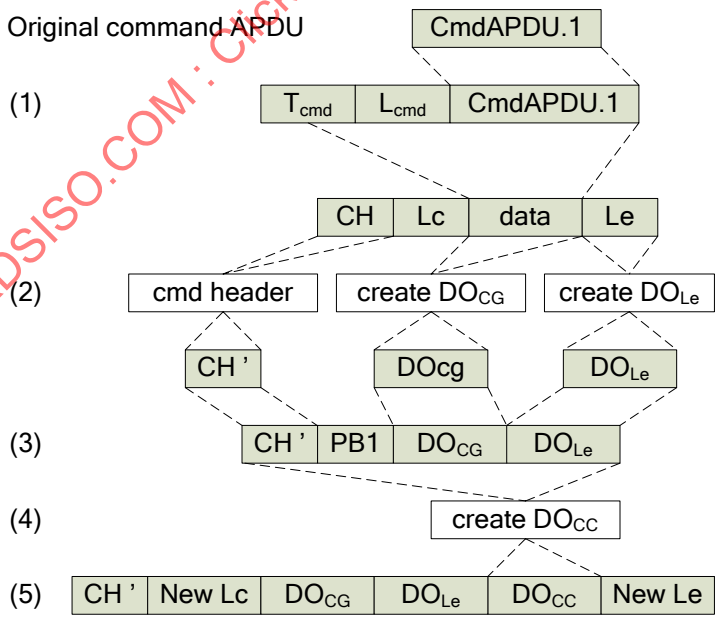


Figure A.8 — Securing a case 4 command APDU

A.1.3 Securing a response APDU

First increment the SendSequenceCounter by one. The new value of SendSequenceCounter shall be used in steps two and four. Then the following steps shall be performed.

(1) The original response APDU RspAPDU.1 is encapsulated in a DO with tag T_{rsp}. This DO is taken as the response data field of a response APDU.
(2) The data of this response APDU is enciphered and encapsulated in an DO with tag T_{CG} as shown in A.1.1.3 and the status bytes are encapsulated in a DO with tag T_{SW} as shown in A.1.1.6.
(3) DO_{CG} and DO_{SW} are concatenated and then padded as shown in A.1.1.1.
(4) The result from step (3) is the input for MAC calculation.
(5) The secured response APDU contains the following elements
    a. The data field is the concatenation of DO_{CG}, DO_{SW} and DO_{CC}.
    b. The trailer contains the status bytes “SW1–SW2”.

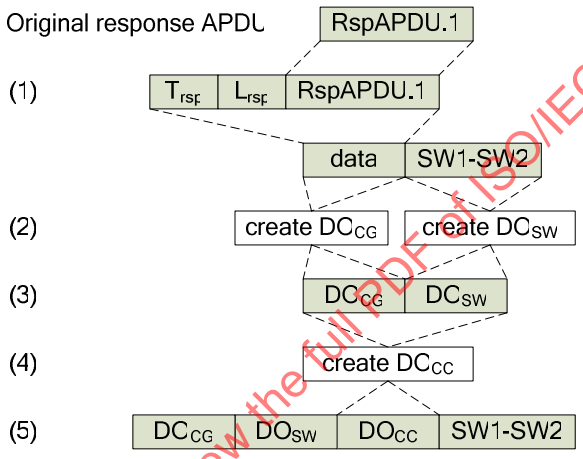


Figure A.9 — Securing a response APDU with a data field

A.1.4 Definitions

Table A.1 — Values used in clause A.1

Term	Value
CH	CLA INS P1 P2 = ’00 C3 0000’
Le	‘0000’
New Le	‘0000’
SW1–SW2	’90 00’
T_{cmd}	‘52’ = command–to–perform
T_{rsp}	‘53’ = discretionary data
T_{CG}	‘85’
T_{Le}	‘97’
T_{CC}	‘8E’
T_{SW}	‘99’
SendSequenceCounter	octet string, result from establishing a trusted channel
Kenc	key, result from establishing a trusted channel
Kmac	key, result from establishing a trusted channel

A.2 Trusted path protocols

An ISO/IEC 24727 compliant middleware stack shall provide a TLS protocol in Trusted Channel implementation used to connect various stack components, except the connection to ICC. Additional protocols may be provided as described in the following clauses of this annex. Secure Messaging defined in ISO/IEC 7816-4 may used at the connection to ICC.

A.2.1 Transport layer security (TLS)

Protocol as specified in: IETF RFC 2246, *The TLS Protocol Version 1.0.*

Annex B
(normative)

IFD - API: Web Service Binding

The Web Service Binding for the IFD-API is defined in the following files:

— `ISOCommon.XSD` – defines basic types such as the `ResponseType`, which forms the basis for all response messages.

— `ISOIFD.XSD` – in which the parameters for each IFD-API request and response are specified as XML-elements.

— `ISOIFD.WSDL` – which includes `ISOIFD.XSD` and specifies the Web Service binding for the IFD-API.

B.1 Specification of `ISOCommon.XSD`

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:iso:std:iso-iec:24727:tech:schema"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema">
    <import namespace="urn:oasis:names:tc:dss:1.0:core:schema"
schemaLocation="oasis-dss-core-schema-v1.0-os.xsd"></import>

    <!-- Definition of Basic Types -->

    <simpleType name="SlotHandleType">
        <restriction base="hexBinary">
        </restriction>
    </simpleType>

    <complexType name="ChannelHandleType">
        <sequence>
            <element name="ProtocolTerminationPoint" type="anyURI" maxOccurs="1"
minOccurs="0"></element>
            <element name="SessionIdentifier" type="string" maxOccurs="1"
minOccurs="0"></element>
            <element name="Binding" type="anyURI" maxOccurs="1"
                minOccurs="0" default="http://schemas.xmlsoap.org/soap/http">
            </element>
        </sequence>
    </complexType>

    <simpleType name="ContextHandleType">
        <restriction base="hexBinary">
        </restriction>
    </simpleType>
```

```
    <!-- Define Response Type -->

    <complexType name="RequestType">
        <complexContent>
            <restriction base="dss:RequestBaseType">
            </restriction>
        </complexContent>
    </complexType>

    <complexType name="ResponseType">
        <complexContent>
            <restriction base="dss:ResponseBaseType">
                <sequence>
                    <element ref="dss:Result"/>
                </sequence>
            </restriction>
        </complexContent>
    </complexType>

</schema>
```

B.2 Specification of `ISOIFD.XSD`

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:iso:std:iso-iec:24727:tech:schema"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema">

    <!-- Definition of Basic Types -->

    <include schemaLocation="ISOCommon.xsd"></include>

    <!-- Card terminal related functions -->

    <!-- EstablishContext  -->

    <element name="EstablishContext">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ChannelHandle"
                            type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"
/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>
```

```
    <element name="EstablishContextResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" maxOccurs="1" minOccurs="0">
                        </element>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <!-- ReleaseContext  -->

    <element name="ReleaseContext">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <element name="ReleaseContextResponse" type="iso:ResponseType">
    </element>

    <!-- ListIFDs  -->

    <element name="ListIFDs">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

<element name="ListIFDsResponse">
    <complexType>
        <complexContent>
            <extension base="iso:ResponseType">
                <sequence>
                    <element name="IFDName" maxOccurs="unbounded"
                        minOccurs="0" type="string" />
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
```

```
    <!-- GetIFDCapabilities  -->

    <element name="GetIFDCapabilities">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" />
                        <element name="IFDName" type="string" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <element name="GetIFDCapabilitiesResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence maxOccurs="1" minOccurs="0">
                        <element name="IFDCapabilities" maxOccurs="1"
                            minOccurs="1" type="iso:IFDCapabilitiesType" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <complexType name="IFDCapabilitiesType">
        <sequence>
            <element name="SlotCapability" type="iso:SlotCapabilityType"
maxOccurs="unbounded" minOccurs="1"/>
            <element name="DisplayCapability" type="iso:DisplayCapabilityType"
maxOccurs="unbounded" minOccurs="0"/>
            <element name="KeyPadCapability" type="iso:KeyPadCapabilityType"
maxOccurs="unbounded" minOccurs="0"></element>
            <element name="BioSensorCapability" type="iso:BioSensorCapabilityType"
maxOccurs="unbounded" minOccurs="0"></element>
            <element name="OpticalSignalUnit" type="boolean"></element>
            <element name="AcousticSignalUnit" type="boolean"></element>
        </sequence>
    </complexType>

    <complexType name="SlotCapabilityType">
        <sequence>
            <element name="Index" type="nonNegativeInteger" maxOccurs="1"
minOccurs="1"/>
            <element name="ContactBased" type="boolean"></element>
        </sequence>
    </complexType>

    <complexType name="DisplayCapabilityType">
        <sequence>
            <element name="Index" type="nonNegativeInteger"
                maxOccurs="1" minOccurs="1" />
            <element minOccurs="1" maxOccurs="1" name="Lines"
                type="nonNegativeInteger" />
            <element name="Columns" type="nonNegativeInteger" />
            <element name="VirtualLines" type="nonNegativeInteger"
                maxOccurs="1" minOccurs="0" />
```

```
            <element name="VirtualColumns" type="nonNegativeInteger"
                maxOccurs="1" minOccurs="0" />
        </sequence>
    </complexType>

    <complexType name="KeyPadCapabilityType">
        <sequence>
            <element name="Index" type="nonNegativeInteger"
                maxOccurs="1" minOccurs="1" />
            <element minOccurs="1" maxOccurs="1" name="Keys"
                type="positiveInteger" />
        </sequence>
    </complexType>

    <complexType name="BioSensorCapabilityType">
        <sequence>
            <element name="Index" type="nonNegativeInteger"
                maxOccurs="1" minOccurs="1" />
            <element minOccurs="1" maxOccurs="1" name="BiometricType"
                type="nonNegativeInteger" />
        </sequence>
    </complexType>

    <!-- GetStatus  -->

    <element name="GetStatus">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                        <element name="IFDName" type="string"
                            maxOccurs="1" minOccurs="0" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <element name="GetStatusResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence maxOccurs="1" minOccurs="1">
                        <element name="IFDStatus" maxOccurs="unbounded"
                            minOccurs="0" type="iso:IFDStatusType" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <complexType name="IFDStatusType">
        <sequence>
            <element name="IFDName" type="string" maxOccurs="1"
minOccurs="0"></element>
            <element name="Connected" type="boolean" maxOccurs="1"
                minOccurs="0" />
            <element minOccurs="1" maxOccurs="unbounded"
```

```
            name="SlotStatus" type="iso:SlotStatusType">
            <annotation>
               <documentation>Index of the slot.</documentation>
            </annotation>
         </element>
         <element name="ActiveAntenna" type="boolean" maxOccurs="1"
            minOccurs="0" />
         <element minOccurs="0" maxOccurs="unbounded" name="DisplayStatus"
            type="iso:SimpleFUStatusType">
            <annotation>
               <documentation>Index of the display.</documentation>
            </annotation>
         </element>
         <element minOccurs="0" maxOccurs="unbounded" name="KeyPadStatus"
            type="iso:SimpleFUStatusType">
            <annotation>
               <documentation>Index of the keypad.</documentation>
            </annotation>
         </element>
         <element minOccurs="0" maxOccurs="unbounded"
            name="BioSensorStatus" type="iso:SimpleFUStatusType" />
      </sequence>
   </complexType>

   <complexType name="SlotStatusType">
      <sequence>
         <element name="Index" type="nonNegativeInteger"
            maxOccurs="1" minOccurs="1" />
         <element minOccurs="1" maxOccurs="1" name="CardAvailable"
            type="boolean" />
         <element name="ATRorATS" type="hexBinary" maxOccurs="1"
minOccurs="0"></element>
      </sequence>
   </complexType>

   <complexType name="SimpleFUStatusType">
      <sequence>
         <element name="Index" type="nonNegativeInteger" />
         <element name="Available" type="boolean" />
      </sequence>
   </complexType>

   <!-- Wait -->

   <element name="Wait">
      <complexType>
         <complexContent>
            <extension base="iso:RequestType">
               <sequence>
                  <element name="ContextHandle"
                     type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                  <element name="TimeOut" type="positiveInteger"
                     maxOccurs="1" minOccurs="0" />
                  <element name="IFDStatus"
                     type="iso:IFDStatusType" maxOccurs="unbounded"
minOccurs="0" />
                  <element name="Callback"
                     type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0">
                  </element>
               </sequence>
```

```
            </extension>
        </complexContent>
    </complexType>
</element>

<element name="WaitResponse">
    <complexType>
        <complexContent>
            <extension base="iso:ResponseType">
                <sequence maxOccurs="1" minOccurs="1">
                    <element name="IFDEvent" type="iso:IFDStatusType"
maxOccurs="unbounded" minOccurs="0"></element>
                    <element name="SessionIdentifier" type="string" maxOccurs="1"
minOccurs="0"></element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>

<!-- Cancel  -->

<element name="Cancel">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="ContextHandle"
                        type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                    <element name="IFDName" type="string" maxOccurs="1"
minOccurs="0"/><element
                        name="SessionIdentifier" type="string" maxOccurs="1"
minOccurs="0">
</element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>

<element name="CancelResponse" type="iso:ResponseType" />

<!-- ControlIFD  -->

<element name="ControlIFD">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="ContextHandle"
                        type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                    <element name="IFDName" type="string" />
                    <element name="Command" type="hexBinary" />
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
```

```
    <element name="ControlIFDResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence>
                        <element name="Response" type="hexBinary" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

<!-- Card related functions -->

    <!-- Connect -->

    <element name="Connect">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="ContextHandle"
                            type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"
/>
                        <element name="IFDName" type="string" />
                        <element name="Slot" type="nonNegativeInteger" />
                        <element name="Exclusive" type="boolean"
                            maxOccurs="1" minOccurs="0" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <element name="ConnectResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence>
                        <element name="SlotHandle"
                            type="iso:SlotHandleType">
                        </element>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <!--Disconnect -->

    <element name="Disconnect">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="SlotHandle"
                            type="iso:SlotHandleType" />
                        <element name="Action" type="iso:ActionType"
                            maxOccurs="1" minOccurs="0" />
                    </sequence>
                </extension>
```

```
        </complexContent>
    </complexType>
</element>

<element name="DisconnectResponse" type="iso:ResponseType" />

<simpleType name="ActionType">
    <restriction base="string">
        <enumeration value="Reset" />
        <enumeration value="Unpower" />
        <enumeration value="Eject" />
        <enumeration value="Confiscate" />
    </restriction>
</simpleType>

<!-- BeginTransaction -->
<element name="BeginTransaction">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="SlotHandle"
                        type="iso:SlotHandleType" />
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>

<element name="BeginTransactionResponse" type="iso:ResponseType" />

<!-- EndTransaction -->
<element name="EndTransaction">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="SlotHandle"
                        type="iso:SlotHandleType" />
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>

<element name="EndTransactionResponse" type="iso:ResponseType" />

<!-- Transmit -->
<element name="Transmit">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="SlotHandle"
                        type="iso:SlotHandleType" />
                    <element name="InputAPDU" type="hexBinary" />
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
```

```
    <element name="TransmitResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence>
                        <element name="OutputAPDU" type="hexBinary"></element>
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

<!--User related functions  -->

    <!-- VerifyUser -->
    <element name="VerifyUser">
        <complexType>
            <complexContent>
                <extension base="iso:RequestType">
                    <sequence>
                        <element name="SlotHandle"
                            type="iso:SlotHandleType" />
                        <element name="InputUnit"
                            type="iso:InputUnitType" />
                        <element name="DisplayIndex"
                            type="nonNegativeInteger" maxOccurs="1" minOccurs="0">
                        </element>
                        <element name="AltVUMessages"
                            type="iso:AltVUMessagesType" maxOccurs="1" minOccurs="0"
/>
                        <element name="TimeoutUntilFirstKey"
                            type="positiveInteger" maxOccurs="1" minOccurs="0" />
                        <element name="TimeoutAfterFirstKey"
                            type="positiveInteger" maxOccurs="1" minOccurs="0" />
                        <element name="Template" type="hexBinary" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <element name="VerifyUserResponse">
        <complexType>
            <complexContent>
                <extension base="iso:ResponseType">
                    <sequence>
                        <element name="Response" type="hexBinary"
                            maxOccurs="1" minOccurs="1" />
                    </sequence>
                </extension>
            </complexContent>
        </complexType>
    </element>

    <complexType name="InputUnitType">
        <choice>
            <element name="PinInput" type="iso:PinInputType"></element>
            <element name="BiometricInput"
                type="iso:BiometricInputType">
            </element>
```

```
        </choice>
    </complexType>

    <complexType name="PinInputType">
        <sequence>
            <element name="Index" type="nonNegativeInteger" />
            <element name="PasswordAttributes" type="iso:PasswordAttributesType"
maxOccurs="1" minOccurs="0"/>
        </sequence>
    </complexType>

    <simpleType name="PadCharType">
        <restriction base="hexBinary">
            <length value="1" fixed="true"/>
        </restriction>
    </simpleType>

     <complexType name="PasswordAttributesType">
        <sequence>
            <element name="pwdFlags"
                type="iso:PasswordFlagsType">
            </element>
            <element name="pwdType"
                type="iso:PasswordTypeType">
            </element>
            <element name="minLength"
                type="nonNegativeInteger">
            </element>
            <element name="storedLength"
                type="nonNegativeInteger">
            </element>
            <element name="maxLength"
                type="nonNegativeInteger" maxOccurs="1" minOccurs="0">
            </element>
            <element name="padChar" type="iso:PadCharType"
                maxOccurs="1" minOccurs="0">
            </element>
            <element name="lastPasswordChange"
                type="dateTime" maxOccurs="1" minOccurs="0">
            </element>
        </sequence>
    </complexType>

    <simpleType name="PasswordFlagsType">
        <union memberTypes="iso:BitString">
            <simpleType>
                <list>
                    <simpleType>
                        <restriction base="token">
                            <enumeration value="case-sensitive" />
                            <enumeration value="local" />
                            <enumeration value="change-disabled" />
                            <enumeration value="unblock-disabled" />
                            <enumeration value="initialized" />
                            <enumeration value="needs-padding" />
                            <enumeration value="unblockingPassword" />
                            <enumeration value="soPassword" />
                            <enumeration value="disable-allowed" />
                            <enumeration value="integrity-protected" />
                            <enumeration value="confidentiality-protected" />
                            <enumeration value="exchangeRefData" />
```

```
                    <enumeration value="resetRetryCounter1" />
                    <enumeration value="resetRetryCounter2" />
                </restriction>
            </simpleType>
        </list>
    </simpleType>
</union>
</simpleType>

<simpleType name="PasswordTypeType">
    <restriction base="string">
        <enumeration value="bcd" />
        <enumeration value="ascii-numeric" />
        <enumeration value="utf8" />
        <enumeration value="half-nibble-bcd" />
        <enumeration value="iso9564-1" />
    </restriction>
</simpleType>

<simpleType name="BitString">
    <restriction base="string">
        <pattern value="[0-1]{0,}" />
    </restriction>
</simpleType>

<complexType name="BiometricInputType">
    <sequence>
        <element name="Index" type="nonNegativeInteger" />
        <element name="BiometricSubtype" type="nonNegativeInteger" />
    </sequence>
</complexType>

<complexType name="AltVUMessagesType">
    <sequence>
        <element name="AuthenticationRequestMessage" type="string"
            maxOccurs="1" minOccurs="0" />
        <element name="SuccessMessage" type="string" maxOccurs="1"
            minOccurs="0" />
        <element name="AuthenticationFailedMessage" type="string"
            maxOccurs="1" minOccurs="0" />
        <element name="RequestConfirmationMessage" type="string"
            maxOccurs="1" minOccurs="0" />
        <element name="CancelMessage" type="string" maxOccurs="1"
            minOccurs="0" />
    </sequence>
</complexType>


<!--ModifyVerificationData  -->

<element name="ModifyVerificationData">
    <complexType>
        <complexContent>
            <extension base="iso:RequestType">
                <sequence>
                    <element name="SlotHandle"
                        type="iso:SlotHandleType" maxOccurs="1" minOccurs="1" />
                    <element name="InputUnit"
                        type="iso:InputUnitType" maxOccurs="1" minOccurs="1" />
                    <element name="DisplayIndex"
                        type="nonNegativeInteger" maxOccurs="1" minOccurs="0">
```