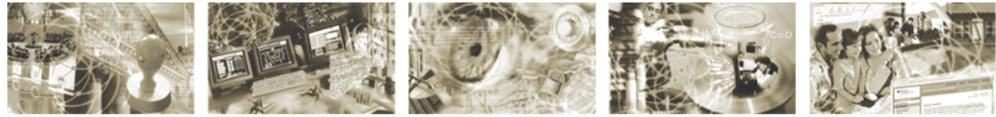




Bundesamt
für Sicherheit in der
Informationstechnik



Technical Guideline TR-03112-7

eCard-API-Framework – Protocols

Version 1.1.3 draft

5. June 2013

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn

E-Mail: ecard.api@bsi.bund.de
Internet: <https://www.bsi.bund.de>
© Bundesamt für Sicherheit in der Informationstechnik 2013

Contents

1	Overview of the eCard-API-Framework	6
2	Connection establishment in distributed systems.....	8
2.1	General security requirements.....	8
2.2	Connection establishment for SOAP binding.....	8
2.3	Connection Establishment for PAOS binding	9
2.3.1	Setting up a Trusted Channel.....	10
2.3.2	PAOS Communication.....	10
2.3.3	Session Termination	11
2.4	TC_API_Open.....	11
2.4.1	Security mechanisms for the channel established with TC_API_Open.....	12
2.5	TC_API_Close.....	13
2.6	StartPAOS.....	14
3	ISO/IEC 24727 protocols.....	15
3.1	PIN Compare.....	15
3.1.1	Marker.....	15
3.1.2	DIDCreate.....	24
3.1.3	DIDUpdate.....	24
3.1.4	DIDGet.....	25
3.1.5	DIDAuthenticate.....	25
3.1.6	Non-supported functions.....	25
3.2	Mutual authentication.....	26
3.2.1	Marker.....	27
3.2.2	DIDCreate.....	29
3.2.3	DIDUpdate.....	29
3.2.4	DIDGet.....	29
3.2.5	CardApplicationStartSession.....	29
3.2.6	DIDAuthenticate.....	30
3.2.7	Non-supported functions.....	31
3.2.8	Minimum requirements in terms of algorithms.....	32
3.3	Password Authenticated Connection Establishment.....	32
3.3.1	Marker.....	33
3.3.2	DIDCreate.....	33
3.3.3	DIDUpdate.....	33
3.3.4	DIDGet.....	34
3.3.5	DIDAuthenticate.....	34
3.3.6	CardApplicationStartSession.....	36
3.3.7	Non-supported functions.....	36
3.4	Chip Authentication.....	37
3.4.1	Marker.....	37
3.4.2	DIDCreate.....	37
3.4.3	DIDUpdate.....	38
3.4.4	DIDGet.....	39
3.4.5	DIDAuthenticate.....	39
3.4.6	Non-supported functions.....	40
3.5	Terminal Authentication.....	41
3.5.1	Marker.....	41
3.5.2	DIDCreate.....	41
3.5.3	DIDUpdate.....	41

3.5.4	DIDGet.....	41
3.5.5	DIDAuthenticate.....	42
3.5.6	Non-supported functions.....	44
3.6	Extended Access Control.....	44
3.6.1	EAC protocol specification.....	45
3.6.2	Marker.....	45
3.6.3	Call and return of CardApplicationStartSession.....	46
3.6.4	Overview of EAC protocol sequence.....	51
3.6.5	Phase 1 - Extended PACE protocol.....	53
3.6.6	Phase 2 - Combination of Terminal and Chip Authentication.....	56
3.6.7	Phase 3 - Optional additional message with signature forwarded separately.....	57
3.6.8	Phase 4 - Secure messaging with APDU batches.....	58
3.6.9	DIDCreate, DIDUpdate and DIDGet.....	58
3.6.10	Non-supported functions.....	58
3.7	Restricted Identification.....	59
3.7.1	Marker.....	59
3.7.2	DIDCreate.....	59
3.7.3	DIDUpdate.....	59
3.7.4	DIDGet.....	60
3.7.5	DIDAuthenticate.....	60
3.7.6	Non-supported functions.....	60
3.8	RSA Authentication.....	61
3.8.1	Marker.....	62
3.8.2	DIDCreate.....	65
3.8.3	DIDUpdate.....	65
3.8.4	DIDGet.....	65
3.8.5	CardApplicationStartSession.....	65
3.8.6	DIDAuthenticate.....	66
3.8.7	Verification of the certificate path.....	66
3.8.8	Invocation of INTERNAL AUTHENTICATE.....	67
3.8.9	Invocation of EXTERNAL AUTHENTICATE.....	68
3.8.10	VerifyCertificate.....	68
3.8.11	Non-supported functions.....	68
3.8.12	Minimum requirements in terms of algorithms.....	68
3.9	Generic cryptography.....	69
3.9.1	Marker.....	70
3.9.2	DIDCreate.....	74
3.9.3	DIDUpdate.....	74
3.9.4	DIDGet.....	74
3.9.5	Encipher.....	74
3.9.6	Decipher.....	74
3.9.7	GetRandom.....	74
3.9.8	Hash.....	75
3.9.9	Sign.....	75
3.9.10	VerifySignature.....	75
3.9.11	VerifyCertificate.....	75
3.9.12	DIDAuthenticate.....	75
3.9.13	Non-supported functions.....	76
4	Protocols for GetCertificate.....	77
4.1	GetCertificate by means of Simple Enrollment Protocol.....	77
5	Basic Update Protocol.....	80

A	PAOS Example.....	86
---	-------------------	----

Table of Figures

Figure 1: The architecture of the eCard-API-Framework.....	6
Figure 2: Connection establishment for PAOS binding.....	9
Figure 3: Message Sequence after CardApplicationStartSession(EACSession).....	51
Figure 4: Basic Update Protocol	80

1 Overview of the eCard-API-Framework

The objective of the eCard-API-Framework is the provision of a simple and homogeneous interface to enable standardised use of the various smart cards (eCards) for different applications.

The structure of the eCard-API-Framework is shown in Figure 1. This shows that the eCard-API-Framework is sub-divided into the following layers:

- Application-Layer
- Identity-Layer
- Service-Access-Layer
- Terminal-Layer

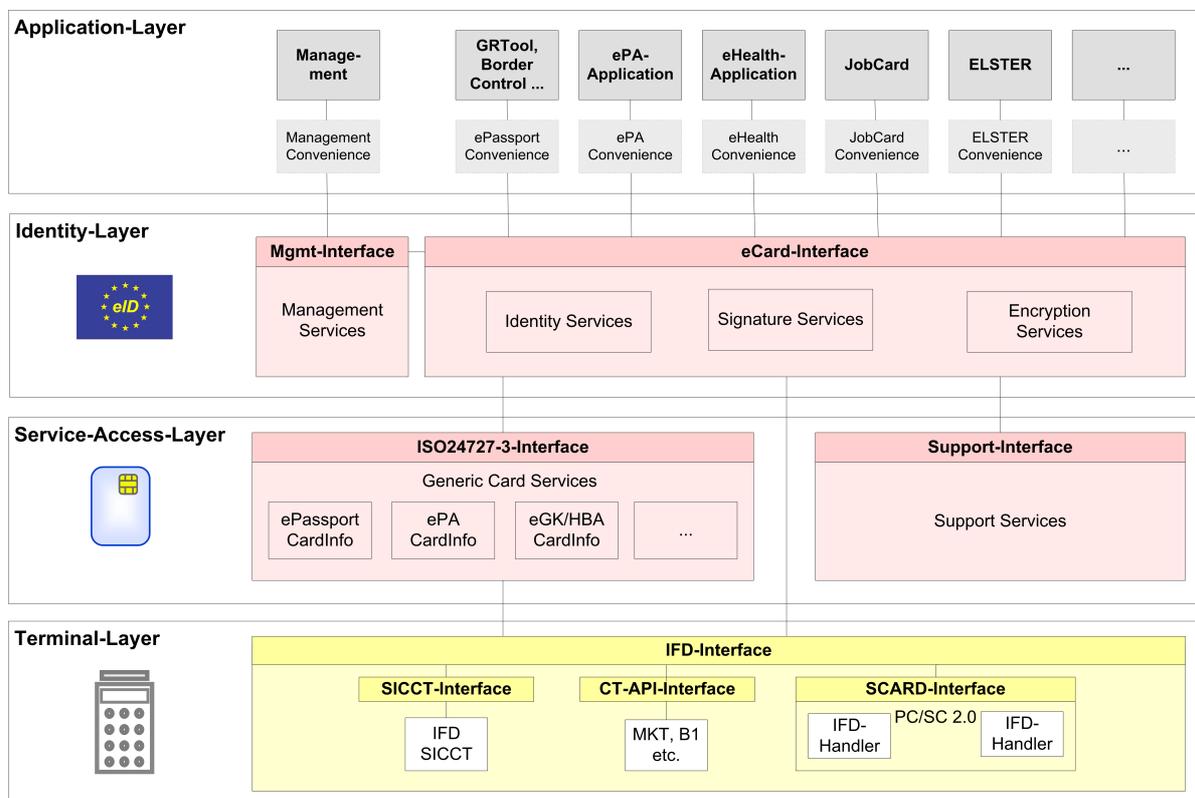


Figure 1: The architecture of the eCard-API-Framework

The **Application-Layer** contains the various applications which use the eCard-API-Framework to access the eCards and their associated functions. Application-specific "convenience interfaces", in which the recurring invocation sequences may be encapsulated in application-specific calls, may also exist in this layer. However, these interfaces are currently *not* within the scope of the e-Card-API-framework.

The **Identity-Layer** comprises the eCard-Interface and the Management interface, and therefore functions for the use and management of electronic identities as well as for management of the eCard-API-Framework.

The *eCard-Interface* (refer to [TR-03112-2]) allows to request certificates as well as the encryption, signature and time-stamping of documents.

In the *Management-Interface* (refer to [TR-03112-3]), functions for updating the framework and the management of trusted identities, smart cards, card terminals, and default behaviour are available.

The **Service-Access-Layer** provides, in particular, functions for cryptographic primitives and biometric mechanisms in connection with cryptographic tokens, and comprises the ISO24727-3-Interface and the Support-Interface.

The *ISO24727-3-Interface* defined in the present document is a webservice-based implementation of the standard of the same name [ISO24727-3]. This interface contains functions to establish (cryptographically protected) connections to smart cards, to manage card applications, to read or write data, to perform cryptographic operations and to manage the respective key material (in the form of so-called "differential identities"). In the process, all functions which use or manage "differential identities" are parameterised by means of protocol-specific object identifiers so that the different protocols which are defined in the present document MAY be used with a standardised interface (refer to [TR-03112-7]).

The *Support-Interface* (refer to [TR-03112-5]) contains a range of supporting functions.

The **Terminal-Layer** primarily contains the *IFD-Interface* (refer to [TR-03112-6]). This layer takes over the generalisation of specific card terminal types and various interfaces as well as communication with the smart card. For the user it is unimportant whether the card is addressed by PC/SC, a SICCT terminal or a proprietary interface, or whether it has contacts or is contact-less.

2 Connection establishment in distributed systems

Some of the protocols specified here involve two instances of the eCard-API-Framework, which run on different systems (Server and Client, which in general both contain a SAL (Server-SAL/Client-SAL, resp.) and additional program logic) and communicate with each other via potentially insecure networks. Therefore, the relevant aspects of security have to be taken into consideration when setting up the transport channel.

2.1 General security requirements

~~To The security of the communications between the different *modules* and *instances* of the eCard-API-Framework dictates that the TLS protocol in accordance with [RFC4346] MUST be used, and that the cryptographic algorithms and security parameters MUST meet the requirements set out in [TR-02102] and [TR-03116], part 4.~~

~~Moreover, public server services, such as eService for EAC-based authentication on the Internet, MUST use X.509 certificates.~~

For communication between different modules of the eCard-API-Framework X.509 certificates MAY be used, whereby the associated private keys are to be adequately protected. Alternatively, anonymous TLS cipher suites, such as TLS_DH_anon from [RFC4346] or TLS_ECDH_anon from [RFC4492], MAY be used, although in this case appropriate security measures are necessary in the operational environment in order to avert man-in-the-middle attacks while the connection is being established.

In both cases there MUST be an exclusive binding of the communication context at application level to the TLS channel which has been established in this process. This communication context is established on connection to the IFD layer via the function `EstablishContext` and represented by the `ContextHandle` (cf. [TR-03112-6], Section 3.1.1). When connecting to the SAL, this communication context corresponds to a connection to the card application established by means of `CardApplicationConnect`, which is represented by a `ConnectionHandle` (cf. [TR-03112-4], Section 3.2.1).

As such, one single TLS channel is typically sufficient to establish communication between a SAL and the IFD layer — irrespective of the number of card terminals and cards connected — whereas a separate TLS channel is required for every connection to a card application for communication to take place between the Identity-Layer or the Application-Layer and the SAL.

2.2 Connection establishment for SOAP binding

When using the SOAP binding [SOAPv1.1], the connection is established simply by setting up a TLS-protected channel between the user of the web service (service consumer) and the provider of the web service (service provider) via which web service messages MAY henceforth be exchanged. In this case the service consumer and service provider take the roles of TLS/http client and TLS/http server, respectively.

~~For the protection of the communication channel with SOAP-binding at least TLS v1.1 according to [RFC4346] MUST be supported.~~ Activation of this protocol is indicated by the URI `urn:ietf:rfc:5246 for version TLS 1.2` or `urn:ietf:rfc:4346 for TLS 1.1` in the `Protocol`-parameter of the `PathSecurity` element (see also [TR-03112-4] and Section 2.4.1).

2.3 Connection Establishment for PAOS binding

When using the PAOS binding [PAOSv2.0], however, a more complex process is required to establish the connection as, in this case, the TLS/http server acts as the user of the web service (service consumer), the TLS/http client acts as the provider of the web service (service provider) and the TLS/http client MUST initiate the connection.

The general connection sequence is shown in Figure 2. The procedure which enables the Client to establish a connection with the Server is described in the following sections.

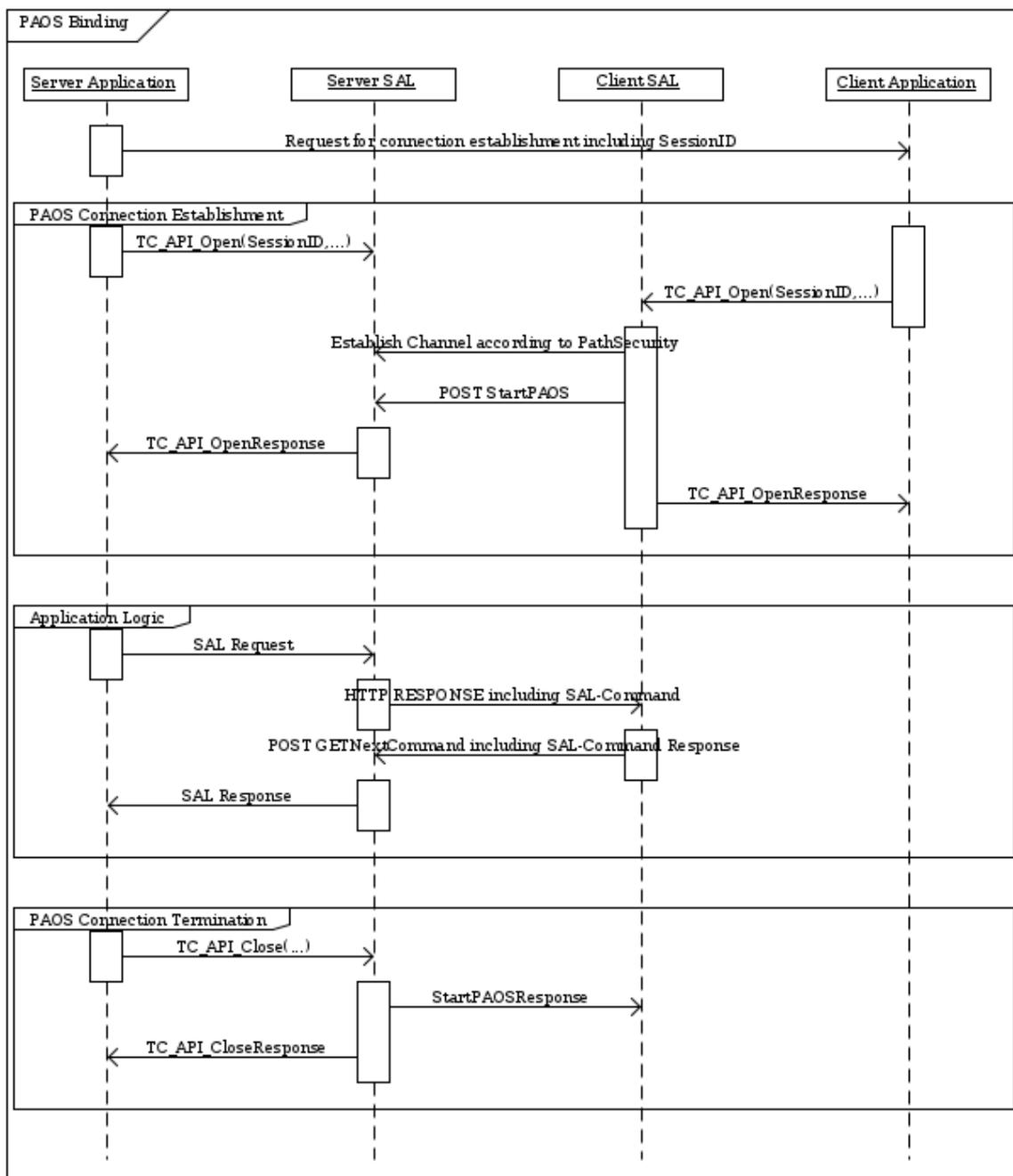


Figure 2: Connection establishment for PAOS binding

2.3.1 Setting up a Trusted Channel

Server and Client share the following parameters necessary to establish a trusted channel:

- `SessionIdentifier` (REQUIRED)
A unique identifier of the authentication session.
- `PSK` (CONDITIONAL)
A cryptographically strong pre-shared key if required by the used TLS-Cipher Suite.

The Trusted Channel is established by two calls to `TC_API_Open` to the Server-SAL and the Client-SAL, respectively:

- The Server sends a `TC_API_Open` call to the Server-SAL with `ChannelHandle` parameter set as follows:
 - `ProtocolTerminationPoint` is not present or is set to <http://127.0.0.1> for `localhost`.
 - `SessionIdentifier` SHALL contain a unique identifier of the authentication session generated by the Server.
 - `Binding` is set to ~~`urn:liberty:paos:2003-08`~~ or `urn:liberty:paos:2006-08` and indicates the need to establish a PAOS connection according to ~~`[PAOSv1.1]`~~ or `[PAOSv2.0]` respectively.
 - `PathSecurity` specifies security measures required while establishing the PAOS connection (cf. Section 2.4.1). If the `Protocol`-element is set to `urn:ietf:rfc:4279`, the `Parameters`-element SHALL contain the pre-shared key PSK.
- The Client sends a `TC_API_Open` call to the Client-SAL with the following parameters:
 - `ServerAddress` – is REQUIRED and specifies the address of the `eServer`-SAL.
 - `SessionIdentifier` – is REQUIRED and specifies the unique identifier of the session, which has been generated by the Server.
 - `Binding` – is OPTIONAL and indicates the web service binding, which is to be used for the established communication channel. If this parameter is omitted the default value `urn:liberty:paos:2006-08` is assumed and the binding specified in `[PAOSv2.0]` is used.
 - `PathSecurity` specifies security measures required while establishing the PAOS connection (cf. Section 2.4.1). If the `Protocol`-element is set to `urn:ietf:rfc:4279`, the `Parameters`-element SHALL contain the pre-shared key PSK.
 - ~~`PathSecurity Protocol` MUST be equal to `urn:ietf:rfc:4279` in order to indicate that the TLS-PSK-mechanism according to `[RFC4279]` is used.~~
 - ~~`PathSecurity Parameters` MUST be present and contain the Pre-Shared-Key in hexadecimal notation wrapped in a `<PSK>`-tag.~~

Note: Both calls may be internal calls if Server/Server-SAL and/or Client/Client-SAL are integrated components.

2.3.2 PAOS Communication

The PAOS connection from the Client-SAL to the Server-SAL is finally established in this step.

The Request-Response Message Exchange Pattern (cf. ~~Section 8 of `[PAOSv1.1]` and~~ Section 4 of `[PAOSv2.0]`) is used for ~~transmitting the `SessionIdentifier` to~~ communication between Client-SAL and the Server-SAL. Hence the Client-SAL sends an HTTP POST or GET request with a PAOS-specific

HTTP-header (cf. [Section 6 of \[PAOSv1.1\]](#) and Section 9.3.1 of [PAOSv2.0]) to the Server-SAL. The information contained in the PAOS-specific HTTP-header indicates which PAOS version is supported (e.g. [urn:liberty:paos:2003-08](#) or [urn:liberty:paos:2006-08](#)) and which service is to be invoked at the eService indicated by the name space defined in the corresponding WSDL (e.g. [urn:iso:std:iso-iec:24727:tech:schema](#)).

The `SessionIdentifier` is transmitted as part of a SOAP-enveloped `StartPAOS`-call (refer to Section 2.6), which SHALL contain information about the card terminals and connected cards available at the client.

[An example is given in Appendix A.](#)

2.3.3 Session Termination

In order to terminate the PAOS-connection the Server sends `TC_API_Close` to the Server -SAL, which results in an HTTP POST Response to the Client, which SHALL contain a SOAP-enveloped `StartPAOSResponse`-element (refer to Section 2.6).

2.4 TC_API_Open

The function `TC_API_Open` can be used to initiate the establishment of a connection with a specific binding (e.g. [\[PAOSv1.1\]](#) or [PAOSv2.0]) and specific security parameters between two systems. If successful, there will be a communication channel to the specified system which may be used to transmit calls.

Input:

- `ChannelHandle` [`ChannelHandleType`] (REQUIRED)
Specifies the system (cf. `ProtocolTerminationPoint`) with which a connection is to be established or the system (cf. `SessionIdentifier`) from which a connection is to be accepted. If both elements are omitted, the called SAL will generate a `SessionIdentifier` and return it in `TC_API_OpenResponse`. Furthermore the `Binding` and `PathSecurity` MAY be specified within the provided `ChannelHandle`.
The specific characteristics of the `ChannelHandle` when establishing a secure PAOS-based connection, ~~which is linked to the TLS channel set up beforehand,~~ are detailed in Section 2.4.1. ~~If the `Protocol` element within `PathSecurity` is set to `urn:ietf:rfe:4279` and the `Parameters` element is omitted, the eService-SAL MUST generate the `PSK-Element` and return it within `TC_API_OpenResponse`.~~

Output:

- `ChannelHandle` [`ChannelHandleType`] (CONDITIONAL)
The `ChannelHandle` is returned if its content has changed.

Error codes:

- [/resultminor/al/common#noPermission](#)
- [/resultminor/al/common#internalError](#)
- [/resultminor/al/common#parameterError](#)
- [/resultminor/dp#nodeNotReachable](#)
- [/resultminor/dp#timeout](#)
- [/resultminor/dp#unknownProtocol](#)

-
- [/resultminor/dp#unknownWebserviceBinding](#)

2.4.1 Security mechanisms for the channel established with TC_API_Open

Depending on the specific purpose for which a communication channel is intended, certain fundamental security requirements MUST be met when setting one up with TC_API_Open and appropriate security mechanisms, which are addressed by duly configured PathSecurity elements, MUST be deployed.

Furthermore appropriate security measures MUST be taken on a client system to ensure that a TC_API_Open invocation to the Client-SAL MUST NOT be accepted directly from a remote system but only from localhost.

2.4.1.1 TLS

The usage of TLS to set up a secure channel is indicated by the URI urn:ietf:rfc:5246 for version TLS 1.2 or urn:ietf:rfc:4346 for TLS 1.1. The supported Cipher Suites, certificate verification and handshake MUST follow the requirements from [TR-03116], part 4. Various PathSecurity alternatives SHOULD be supported as a basic principle, but the TLS procedure set out in Section 2.4.1.2 MUST be supported with previously exchanged "pre-shared keys" in accordance with [RFC4279].

2.4.1.2 TLS with pre-shared keys

This section details a specific form of the PathSecurity element, which is used to establish a PAOS connection protected with "pre-shared keys" via TLS using TC_API_Open.

In this procedure a TLS channel is established between the two SAL instances in accordance with [RFC4279], whereby a previously exchanged secret – the “pre-shared key” – is incorporated in the session key used.

TLS with pre-shared key is indicated by stating the URI urn:ietf:rfc:4279 in the protocol element within PathSecurity. The SessionIdentifier is used as psk_identity. In this case the (otherwise optional) Parameters element in PathSecurity MUST be of the type TLS_PSK_ParameterType which contains the following elements:

- PSK [hexBinary] (REQUIRED)
Contains the value of the pre-shared key which is incorporated, in the manner specified in [RFC4279], into the session key of the TLS channel being established.
- CipherSuite [string, 0..*] (OPTIONAL)
If present, MUST contain a series of *_PSK_* cipher suites according to [TR-03116], part 4, shown as a string. The TLS_PSK_... and TLS_RSA_PSK_... cipher suites from [RFC4279] MUST be supported as a minimum. If no CipherSuite element is given, the default cipher suite TLS_RSA_PSK_WITH_AES_256_CBC_SHA is used.
- DIDName [DIDNameType] (OPTIONAL)
If present, MUST contain the name of the DID which is used on the server-side to establish the TLS channel. The DID given here MUST be a private RSA key suitable for decryption for which a corresponding X.509-based server certificate exists.

2.4.1.3 Error codes

The following error messages MAY also occur in addition to the `ResultMinor` values listed in section 2.4:

- `.../dp#trustedChannelEstablishmentFailed`
- `.../dp#unknownProtocol`
- `.../dp#unknownCipherSuite`
- `.../dp#unknownWebserviceBinding`
- `.../sal#unknownDIDName`
- `.../sal#securityConditionsNotSatisfied`
- `.../sal#functionNotSupported`
- `.../sal#certificateChainInterrupted`
- `.../sal/FunctionalityByCurrentProtocolVersionNotSupported`
- `.../il/signature#certificateNotFound`
- `.../il/signature#certificateFormatNotCorrectWarning`
- `.../il/signature#invalidCertificateReference`
- `.../il/signature#certificatePathNotValidatedWarning`
- `.../il/signature#certificateStatusNotCheckedWarning`
- `.../il/signature#improperRevocationInformationWarning`
- `.../il/signature#invalidCertificatePath`
- `.../il/signature#certificateRevoked`
- `.../il/signature#invalidCertificateExtension`

2.5 TC_API_Close

The function `TC_API_Close` is used to actively close a previously established connection between two systems. The communication channel may no longer be used; the `ChannelHandle` loses its validity.

Input:

- `ChannelHandle` [`ChannelHandleType`] (REQUIRED)
Specifies the connection which is to be closed down.

This function has no output and uses the following error codes:

- `/resultminor/al/common#noPermission`
- `/resultminor/al/common#internalError`
- `/resultminor/al/common#parameterError`
- `/resultminor/dp#communicationError`
- `/resultminor/dp#unknownChannelHandle`

2.6 StartPAOS

The function `StartPAOS` is used for the establishment of a PAOS connection according to [\[PAOSv1.1\]](#) and [\[PAOSv2.0\]](#). In order to avoid the additional transmission of the otherwise necessary SAL-calls `CardApplicationPath` and `CardApplicationConnect` the Client-SAL SHOULD incorporate information about connected card terminals and card applications in form of `ConnectionHandle`-elements into the `StartPAOS`-structure.

Input:

- `SessionIdentifier` [string] (REQUIRED)
Allows to identify the session between the eService-SAL and the Client-SAL.
- `ConnectionHandle` [`ConnectionHandleType`, 0..*] (CONDITIONAL)
SHOULD occur for each connected card application and each empty card terminal slot available at the client.
- `UserAgent` [0..1] (OPTIONAL)
If present, SHALL contain information identifying the used Client. MUST NOT contain any information about the user, the computer of the user or any software installed on the computer of the user apart from the Client.
 - `Name` [string] (REQUIRED)
SHALL contain the name of the Client. SHALL NOT contain any version information.
 - `VersionMajor` [integer] (REQUIRED)
SHALL contain the major version of the Client.
 - `VersionMinor` [integer] (REQUIRED)
SHALL contain the minor version of the Client.
 - `VersionSubminor` [integer, 0..1] (OPTIONAL)
If present, SHALL contain the subminor version of the Client.
- `SupportedAPIVersions` [0..*] (OPTIONAL)
If present, SHALL contain version numbers of supported versions of the eCard-API-specification.
 - `Major` [integer] (REQUIRED)
 - `Minor` [integer, 0..1] (OPTIONAL)
If not present, all minor version corresponding to the given major version are supported.
 - `Subminor` [integer, 0..1] (OPTIONAL)
If not present, all subminor version corresponding to the given major/minor version are supported.
- `SupportedDIDProtocols` [anyURI, 0..*] (OPTIONAL)
If present, SHALL contain a list of supported DIDProtocols.

This function has no output and uses the following error codes:

- [/resultminor/al/common#noPermission](#)
- [/resultminor/al/common#internalError](#)
- [/resultminor/al/common#parameterError](#)
- [/resultminor/dp#nodeNotReachable](#)
- [/resultminor/dp#timeout](#)

3 ISO/IEC 24727 protocols

This section contains the protocol-specific definitions of the Crypto and Differential Identity Services (cf. Sections 3.5 and 3.6 of [TR-03112-4]) for some authentication protocols in accordance with [ISO24727-3], as required for the use of typical signature cards, electronic health insurance cards, healthcare professional ID cards and the planned electronic identity cards.

Note that the protocol identifies the used cryptographic protocol including the used commands as well as the secure messaging to be used after successful completion of the cryptographic protocol.

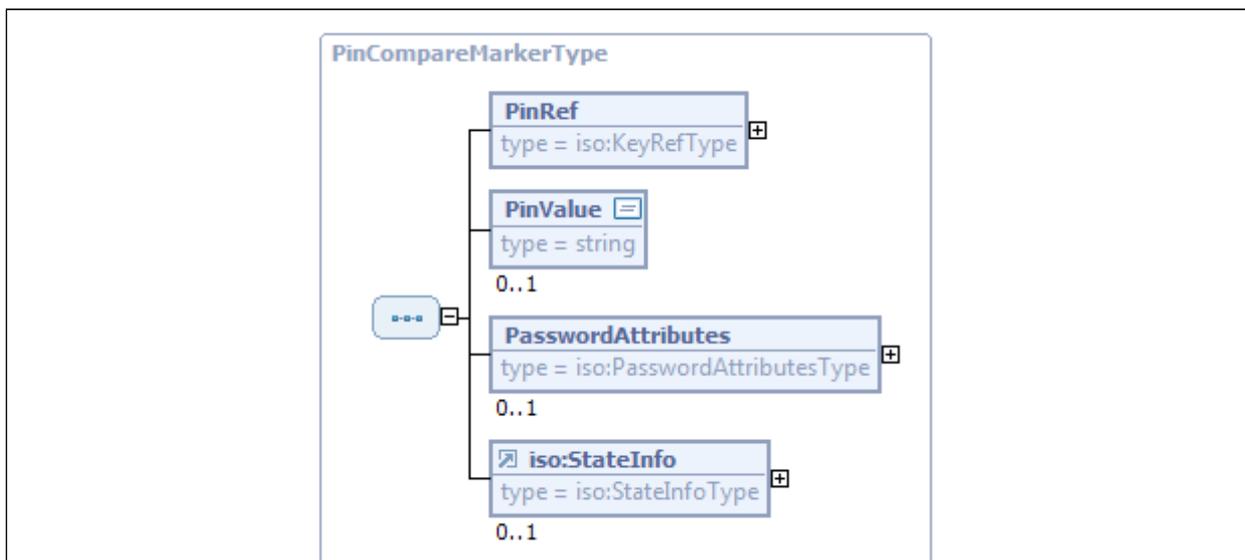
3.1 PIN Compare

Authentication of a user is performed by means of a PIN in this protocol, which is also specified in abridged form in Annex A.9 of [ISO24727-3].

The identifier for this protocol is [urn:oid:1.3.162.15480.3.0.9](https://www.iana.org/assignments/oid-registry/oid-registry.xhtml#iso-1-3-162-15480-3-0-9) for iso(1) identified-organization (3) CEN (162) CEN 15480 (15480) part3(3) annex-a(0) pin-compare(9).

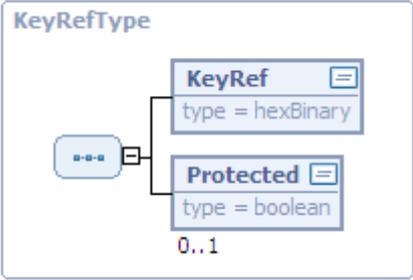
The following types are used to specify the generic structures from [TR-03112-4] for this protocol in more detail.

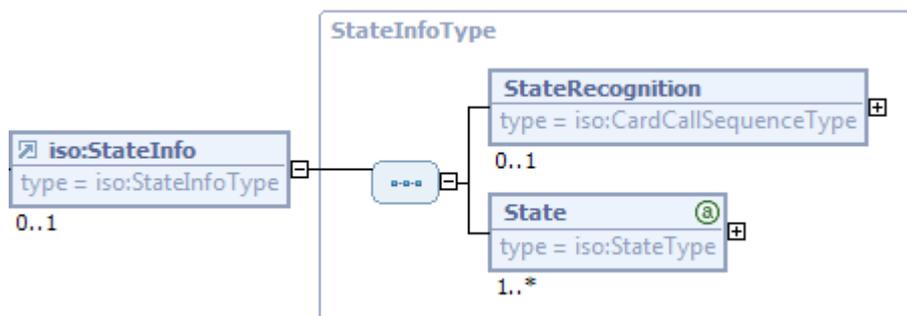
3.1.1 Marker



This type specifies the structure of the DID marker for this authentication protocol.

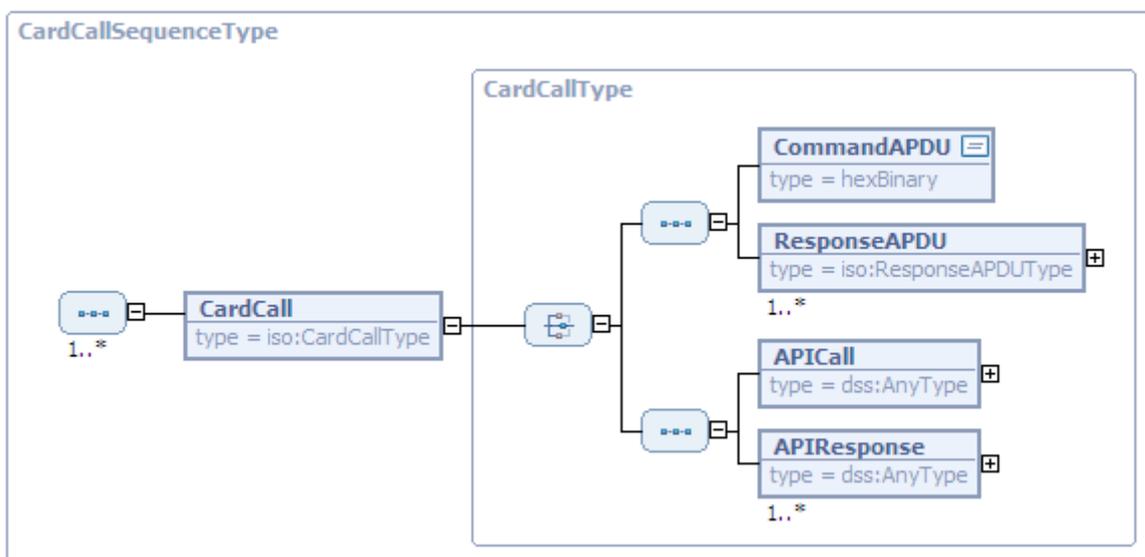
Name	Description
PinRef	Contains the key reference of the PIN. Details on the <code>KeyRefType</code> are provided below.
PinValue	MAY contain the value of the PIN. If this value is missing despite being required, it is input at the terminal. The PIN needs to be stated in the following cases: <ul style="list-style-type: none"> When creating the PIN object with <code>DIDCreate</code> (see Section

	<p>3.1.2).</p> <ul style="list-style-type: none"> • When changing the PIN with DIDUpdate (see Section 3.1.3) and <ul style="list-style-type: none"> ◦ CHANGE REFERENCE DATA (i.e. unblockingPassword bit in pwdFlags (cf. [TR-03112-6]) is not set) ◦ RESET RETRY COUNTER (i.e. the unblockingPassword bit in pwdFlags is set) in the following cases: <ul style="list-style-type: none"> ▪ P1='00' (indicated by the absence of resetRetryCounter1 and resetRetryCounter2 in the PasswordFlags bit string) or ▪ P1='02' (indicated by the setting of resetRetryCounter1 and the absence of resetRetryCounter2 in the PasswordFlags bit string)
Password Attributes	Is an optional element which MAY contain the PasswordAttributes defined in [ISO7816-15]. For details please refer to [TR-03112-6].
iso: StateInfo	MAY contain information about the designated states if more than one possible state is defined for the key object. Details on the iso: StateInfo element are provided below.
 <p>The diagram shows a container labeled 'KeyRefType' with three dots inside. A line connects this container to two sub-elements: 'KeyRef' with 'type = hexBinary' and 'Protected' with 'type = boolean'. Below the sub-elements, the cardinality '0..1' is indicated.</p>	
The KeyRefType is used for the specification of markers (e.g. in the PinCompareMarkerType, MutualAuthMarkerType, RSAAuthMarkerType and the CryptoMarkerType).	
Name	Description
KeyRef	Contains the reference to the key object.
Protected	Is an optional element with which key objects MAY be marked as particularly sensitive (e.g. private signature key or PIN for qualified electronic signature). If such key objects exist in a CardInfo file, they MUST be covered by a signature issued by a trustworthy organisation, and any attempt to access a key object of this type from an unsigned part of a CardInfo file will result in an error message during import of the CardInfo file (cf. [TR-03112-3]).



The `iso:StateInfo` -element is of type `StateInfoType` and is used to specify states of key objects and a procedure which allows to recognize the current state of the key object. A status is uniquely identified by the necessary attribute `StateName`.

Name	Description
StateRecognition	Indicates the sequence of calls to the card through which the current state of the key object can be determined, if the card supports this. The <code>CardCallSequenceType</code> is explained in greater detail below. The commands in the respective <code>StateRecognition</code> elements SHOULD be selected in such a way as to permit the clear identification of the state of the key object.
State	This element is present for each state of the key object. More details concerning this element are provided below (see page 21).



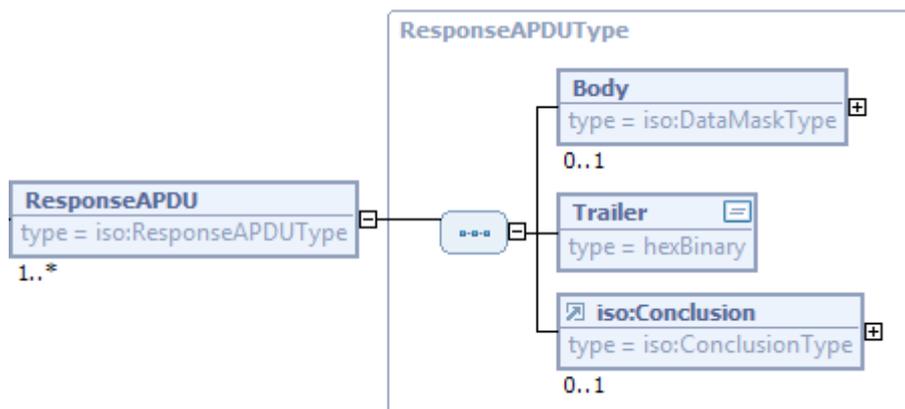
The `CardCallSequenceType` contains a sequence of `CardCall` -elements, which allow to specify a command and a set of possible response pairs for either APDUs or API-calls as defined in [TR-03112-4] or [TR-03112-6] for example.

While the `CommandAPDU / ResponseAPDU` - alternative may be used to identify the card type (cf. [TR-03112-4]) or send statically defined commands to a smart card, the `APICall / APIResponse` - alternative is more powerful and allows to invoke arbitrary API-calls, which may also involve the user (see `ModifyVerificationData` defined in [TR-03112-6] for example).

As a `CardCall`- element contains a request and a sequence of possible responses it MAY be used to

specify a tree structure, which is traversed in order to recognize the state of a key object or a card type (cf. [TR-03112-4], Annex A.3-A.4).

Name	Description
CardCall	<p>Defines a call to the smart card which is given by a CommandAPDU/APICall- element and a sequence of possible ResponseAPDU/APIResponse elements.</p> <p>The sequence of the CardCall elements is to be understood as an AND operation when identifying a status or card type.</p>
CommandAPDU	<p>Contains the APDU command which is to be sent to the card (cf. [ISO7816-4], [ISO7816-8] and [ISO7816-9]).</p> <p>For security reasons no APDUs with CLA values '0x' or '1x' SHOULD be used, where x is any half byte, nor should the INS values '20', '21', '24', '2C' and '22' be used, as this would correspond to the smart card commands VERIFY, CHANGE REFERENCE DATA, RESET RETRY COUNTER and MANAGE SECURITY ENVIRONMENT (see [ISO7816-4], Sections 7.5.6 - 7.5.11), which an attacker could use to decrement the retry counter in the event of a malicious "card or status detection" and thereby provoke a denial-of-service attack under some circumstances.</p> <p>In each case such APDUs MUST be denied if the CardCall element is not signed by a trustworthy body (cf. Signature element in [TR-03112-4], Annex A.7).</p>
ResponseAPDU	<p>Defines a sequence of valid replies from the smart card for the identification of a certain status or smart card type.</p> <p>The sequence of the ResponseAPDU elements allows to traverse a tree structure.</p> <p>The structure of the ResponseAPDUType is explained in detail below.</p>
APICall	Contains an arbitrary API-call, as defined in [TR-03112-4] or [TR-03112-6] for example.
APIResponse	MAY appear multiple times and contain a corresponding response. Note that the APIResponse -element MAY contain a decision element similar to the iso:Conclusion -element explained below such that it is possible to construct a decision tree using API-calls and responses.

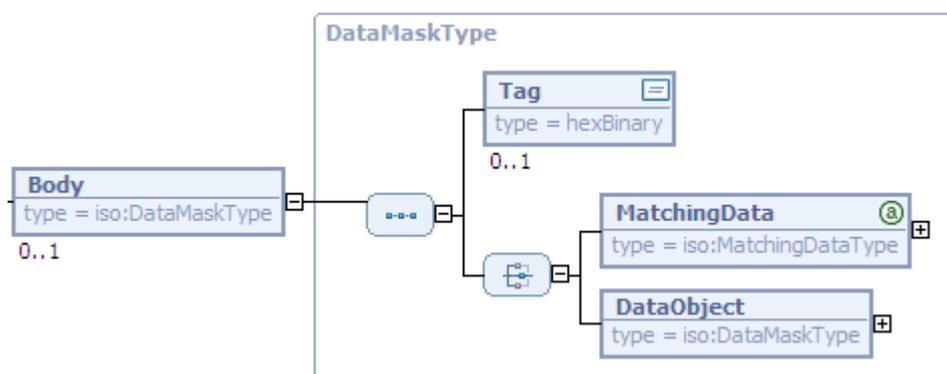


The ResponseAPDU element is part of a CardCall element and specifies a possible reply from an

eCard when invoking a CommandAPDU. The ResponseAPDU comprises an (optional) Body, a Trailer (if successful equal to '9000') and a iso:Conclusion -element, which is present if and only if the CardCall -element is used to specify a decision tree structure.

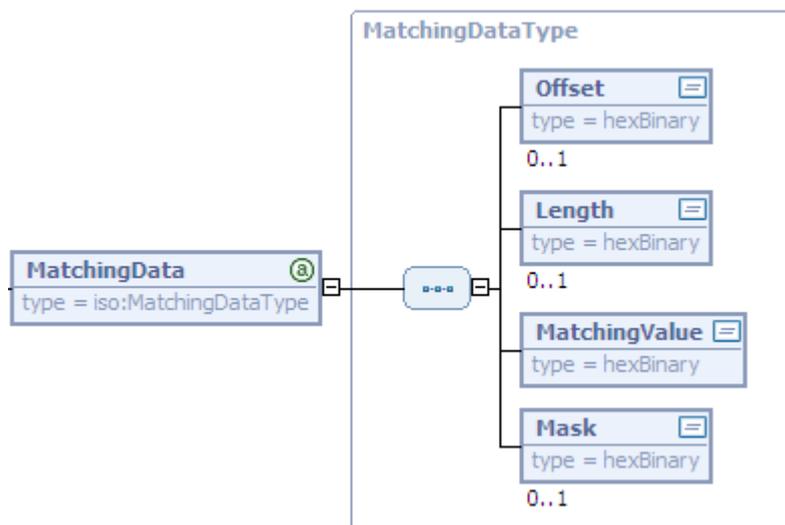
The structure of the DataMaskType defined below allows any parts to be filtered out of the body element and use this information to check for matching data.

Name	Description
Body	MAY contain information indicating which part of the data returned by the smart card is relevant to check for matching data. Further details on the DataMaskType are provided below.
Trailer	Contains the expected status of the invocation (if successful '9000').
iso:Conclusion	The iso:Conclusion- element represents the leaf of the decision tree and allows to determine a specific state or card type or conclude that another sequence of iso:CardCall -elements is necessary. Further details on this element are provided below.



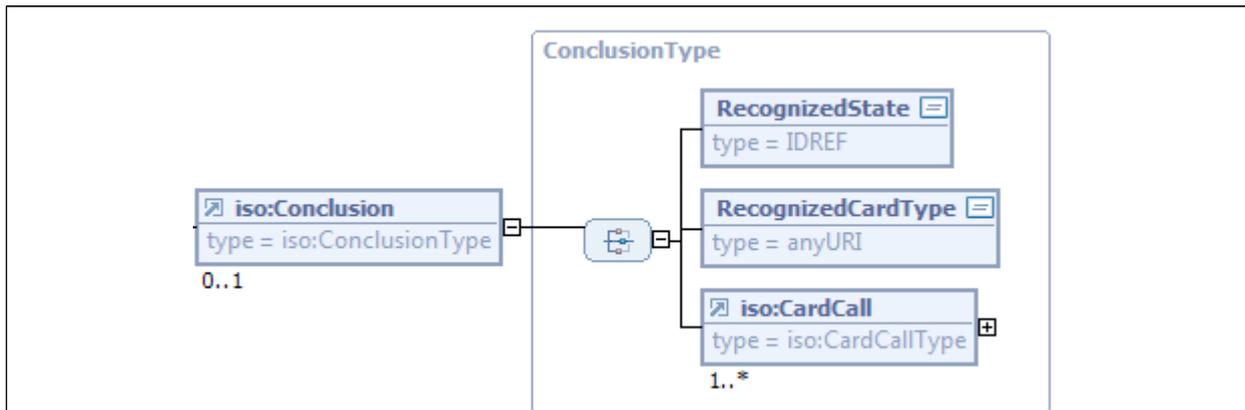
The body element is of the type DataMaskType and MAY be part of the ResponseAPDU element and MAY contain a Tag element and either MatchingData or a structured DataObject of the type DataMaskType.

Name	Description
Tag	MAY contain the tag under which the expected value (or another structured data object) is filed.
MatchingData	Specifies which parts of the returned data are relevant in terms of checking for matching data. The detailed structure of the MatchingData is explained below.
DataObject	Is of the type DataMaskType and therefore again contains information on the selective filtering of a complex data object, thus allowing analysis of TLV-encoded structures irrespective of how they are nested.



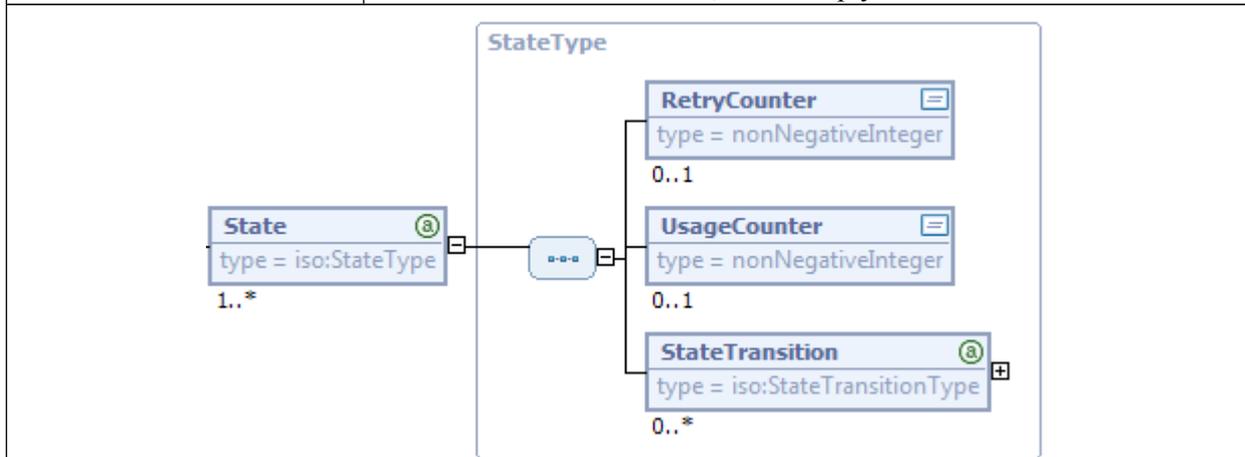
The MatchingData structure makes it possible to specify which parts of a data object returned in the body are relevant for the comparison.

Name	Description
Offset	MAY optionally contain an offset which is taken into account when determining the relevant value. For example, an offset of '03' and a mask <i>xy</i> would have the same effect as a mask '00 00 00 <i>xy</i> '.
Length	MAY contain the length of the value relevant for the comparison.
MatchingValue	<p>Contains the value which is either identical to the value returned by the eCard or contained therein.</p> <p>The optional attribute MatchingRule of the type MatchingRuleType in the MatchingDataType determines whether to check if the values are equal or contained. This type is defined as follows:</p> <pre> <simpleType name = "MatchingRuleType"> <restriction base = "string"> <enumeration value = "Equals" /> <enumeration value = "Contains" /> </restriction> </simpleType> </pre> <p>If this attribute is omitted then Equals is the default setting.</p>
Mask	The OPTIONAL Mask element, which is linked conjunctively with the MatchingValue element and the data returned by the card, makes it possible to filter out the significant parts of the data.



The `iso:Conclusion`-element is part of the `ResponseAPDU`-element.

Name	Description
RecognizedState	This element is present, if a state of a key object is recognized in an unambiguous manner and hence no further <code>iso:CardCall</code> -elements are necessary.
RecognizedCardType	In a similar manner this element is present, if a card type is recognized in an unambiguous manner (cf. [TR-03112-4], Annex A).
<code>iso:CardCall</code>	If the decision process is not finished there will be one or more <code>iso:CardCall</code> -elements, which imply a recursion.



The `State`-element is part of the `StateInfo`-element (see page 17) and it is present for each state of the key object.

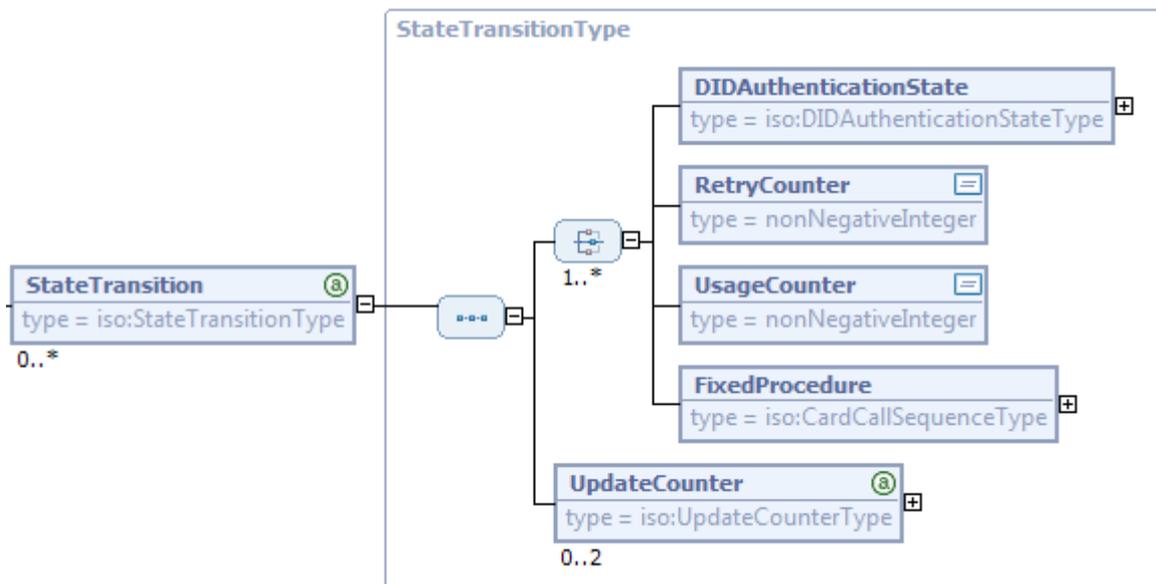
The `State`-element has a number of important attributes as explained in the following:

- `StateName` – is a required identifier of the state.
- `StateClass` – specifies the type of the state, where the following types are possible:
 - `Operational` – means that the key object is usable in this state.
 - `NotOperational` – means that the key object is not usable in this state. If the SAL recognizes that the key object is in such a state it **MUST** try to determine an appropriate sequence of `StateTransition`-elements, which lead to an operational state. If there is no path to an operational (or at least indetermined) state, the user **MUST** be informed

accordingly.

- `RecognitionNecessary` – means that the class of the state is indetermined, which means that it is not clear whether the recognized state is operational or not, and the SAL MUST start the recognition procedure defined by the set of `iso:CardCall` -elements.

Name	Description
RetryCounter	Is an OPTIONAL element which contains the current value of the retry counter for the key object in accordance with [ISO7816-4], Table 34.
UsageCounter	Is an OPTIONAL element which contains the current value of the UsageCounter for the key object in accordance with [ISO7816-4], Table 34.
StateTransition	Contains information on the various state transitions provided for the key object. Further details on the StateTransition element are provided below.

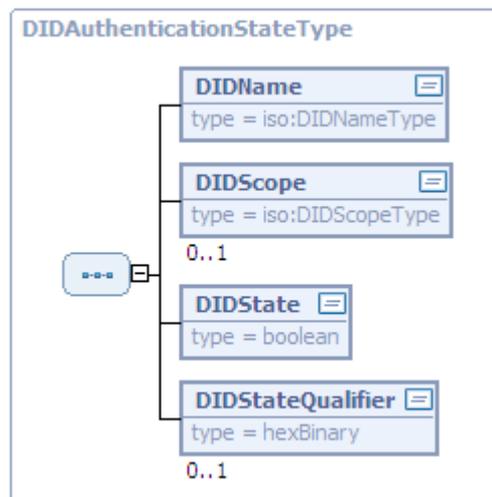


Each State -element defined above may contain an arbitrary number of StateTransition elements, which describe the possible transitions to other states. The StateTransition element has a mandatory TargetState attribute, used to refer to the StateName attribute of an existing state (cf. StateType above).

Transitions between states are triggered by events, which are described by a sequence of DIDAuthenticationState, RetryCounter, UsageCounter and FixedProcedure elements and MAY contain additional UpdateCounter -elements to update the values of the RetryCounter and UsageCounter variables, which MUST be maintained by the SAL to support cards, which do not allow to retrieve those values from the card.

Name	Description
DIDAuthenticationState	Indicates that the transition from one state to another is triggered by using the DID stated in this element. Details on the DIDAuthenticationStateType are provided below.
RetryCounter	Indicates that the transition from one state to another is triggered by reaching the value stated for the RetryCounter of the key

	<p>object.</p> <p>Note that some cards do not allow to retrieve the current value of the <code>RetryCounter</code> of a key object and hence the SAL MUST maintain a corresponding variable and update it according to the <code>UpdateCounter</code> element described below.</p>
<code>UsageCounter</code>	<p>Indicates that the transition from one state to another is triggered by reaching the value stated for the <code>UsageCounter</code> of the key object.</p> <p>Note that some cards do not allow to retrieve the current value of the <code>UsageCounter</code> of a key object and hence the SAL MUST maintain a corresponding variable and update it according to the <code>UpdateCounter</code> element described below.</p>
<code>FixedProcedure</code>	<p>Indicates that the transition from one state to another is triggered by executing a fixed sequence of smart card commands. Details on the <code>CardCallSequenceType</code> are provided on page 17.</p>
<code>UpdateCounter</code>	<p>The <code>UpdateCounter</code> -element is used to manage the <code>RetryCounter</code> and/or <code>UsageCounter</code> variables, which MUST be maintained by the SAL to support cards, which do not allow to retrieve those values from the card.</p> <p>The <code>UpdateCounterType</code> is an extension of the builtin integer type with two mandatory attributes:</p> <ul style="list-style-type: none"> • <code>operation</code> either has the value <code>set</code> or <code>add</code> to indicate, whether the provided value is to be set or added to the current value • <code>variable</code> either has the value <code>RetryCounter</code> or <code>UsageCounter</code> to indicate which variable is to be updated



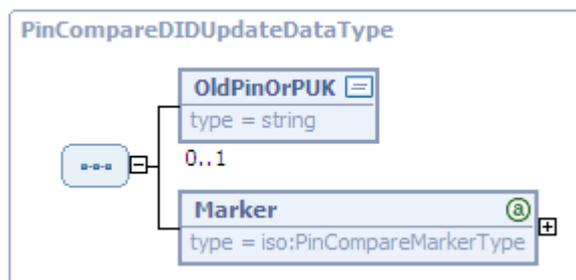
The `DIDAuthenticationStateType` is used in the specification of access rights (cf. `SecurityCondition` in [TR-03112-4]) and in the specification of state transitions (cf. `StateTransition` above).

Name	Description
DIDName	Specifies the name of the DID required for authentication on transition from one state to another.
DIDScope	MAY, if required, resolve ambiguities between local and global DIDs.
DIDState	Specifies the required authentication status and MUST be allocated with the value <code>True</code> when specifying state transitions.
DIDStateQualifier	MAY contain further information which is evaluated when using certificate-based authentication procedures.

3.1.2 DIDCreate

With `DIDCreate` use is made of `DIDCreationData` of the type `PinCompareMarkerType`.

3.1.3 DIDUpdate



This type specifies the structure of the `DIDUpdateDataType` for the PIN Compare protocol.

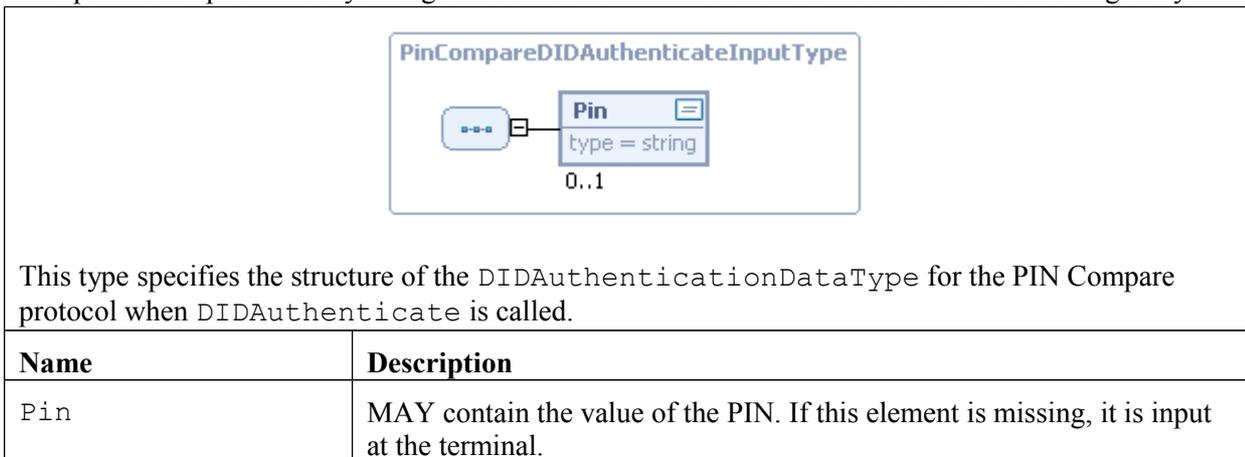
Name	Description
OldPinOrPUK	<p>MAY contain the old PIN or the Personal Unblocking Key (PUK).</p> <p>If this information is missing despite being required then it is input at the terminal.</p> <p>This information is needed if the settings in the <code>pwdFlags</code> element (cf. [TR-03112-6]) are as follows:</p> <ul style="list-style-type: none"> The <code>exchangeRefData</code> bit is set but not the <code>unblockingPassword</code> bit (i.e. it is a normal PIN which can only be changed by entering the old PIN (CHANGE REFERENCE DATA with <code>P1='00'</code>, cf. [ISO7816-4], Section 7.5.7)) or The <code>unblockingPassword</code> bit is set but not the <code>resetRetryCounter1</code> bit (i.e. it is a PUK which MUST be presented upon the RESET RETRY COUNTER command with <code>P1='00'</code> or <code>P1='01'</code> (cf. [ISO7816-4], Section 7.5.10)).
Marker	Contains the new marker for the <code>PinCompare</code> protocol. Details on this element can be found in Section 3.1.1.

3.1.4 DIDGet

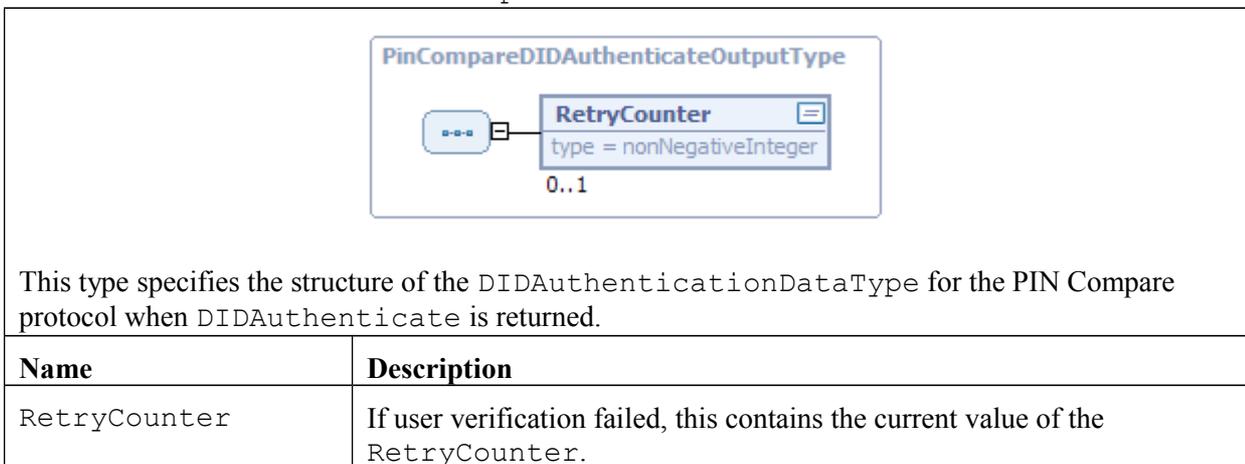
In the case of DIDGet there is a return of DIDDiscoveryData of the type PinCompareMarkerType, although the PinValue element is omitted, and the current reading of any RetryCounter or Usage Counter which may be defined is stated in the first StateInfo element which describes the current status.

3.1.5 DIDAuthenticate

The protocol is processed by a single invocation of DIDAuthenticate with the following entry:



The return to DIDAuthenticateResponse is as follows:



3.1.6 Non-supported functions

The following functions are not supported with this protocol and, when called up, relay an error message to this effect [/resultminor/sal#inappropriateProtocolForAction](#):

- CardApplicationStartSession
- Encipher
- Decipher
- GetRandom

-
- Hash
 - Sign
 - VerifySignature
 - VerifyCertificate

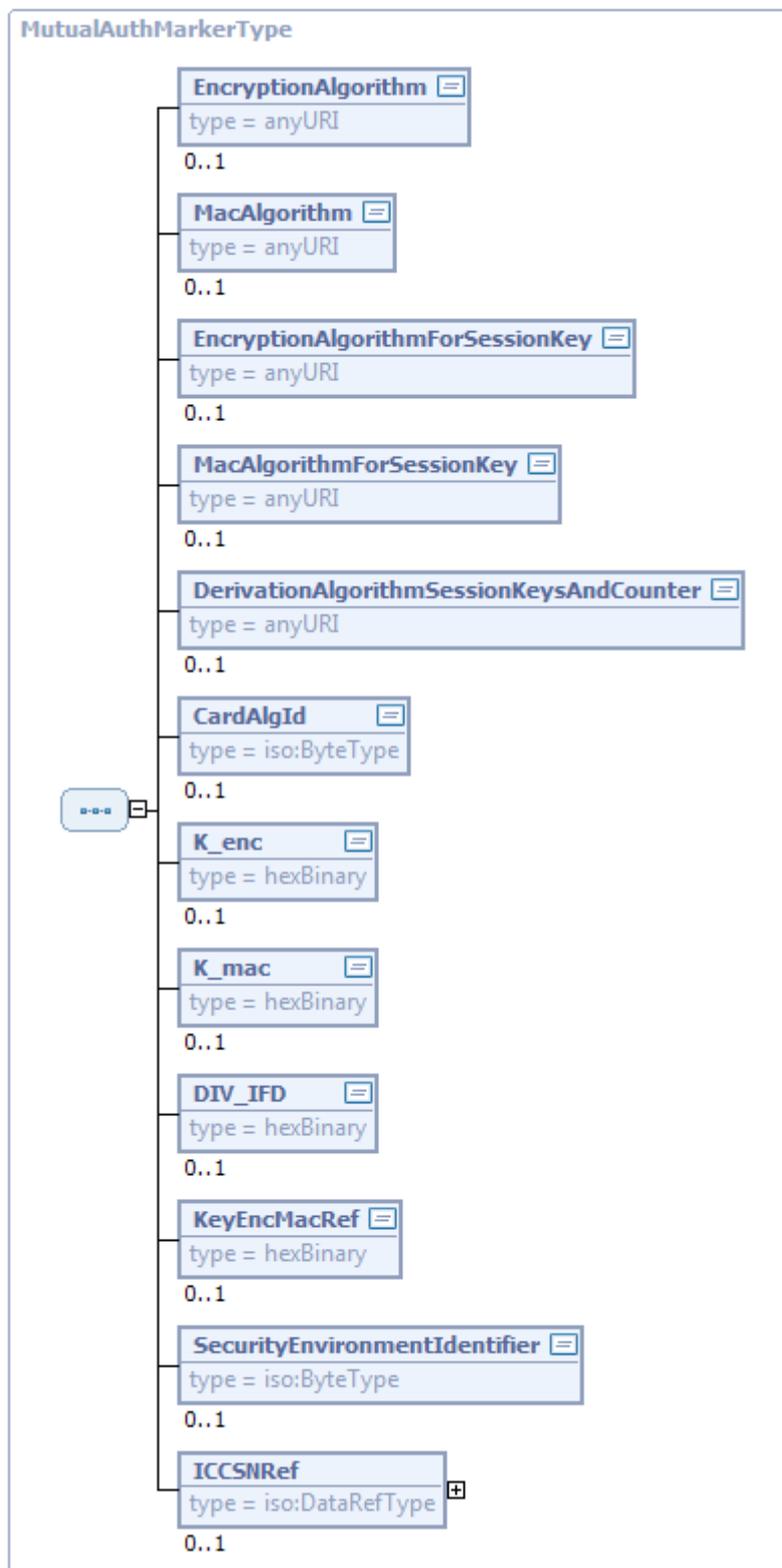
3.2 Mutual authentication

This protocol is specified in similar form in Annex A.12 of [ISO24727-3], Section 16.1.1 [eGK-1] and Section 8.8 of [EN14890-1] and provides the framework for mutual authentication with the exchange of keys using symmetric algorithms.

The identifier for this protocol is [urn:oid:1.3.162.15480.3.0.12](https://nvd.nist.gov/vuln/detail/CVE-2019-11461) for iso(1) identified-organization (3) CEN (162) CEN 15480 (15480) part3(3) annex-a(0) client-application-mutual-authentication-wke(12).

The generic structures from [TR-03112-4] are more closely specified for this protocol by the following types.

3.2.1 Marker



This type specifies the structure of the DID marker for this authentication protocol.

Name	Description
EncryptionAlgorithm	MAY specify the encryption algorithm to be used for connection establishment.
MacAlgorithm	MAY specify the MAC algorithm to be used for connection establishment.
EncryptionAlgorithmForSessionKey	MAY specify the encryption algorithm to be used for connection establishment.
MacAlgorithmForSessionKey	MAY specify the MAC algorithm to be used for secure messaging.
DerivationAlgorithmSessionKeysAndCounter	MAY specify the algorithm required to derive the session key and counters.
CardAlgId	MAY specify the algorithms to be used for this protocol by a single card-specific algorithm identifier.
K_enc	MAY contain the encryption key to be used for connection establishment.
K_mac	MAY contain the MAC key to be used for connection establishment.
DIV_IFD	MAY contain an initialisation vector for the employment of the symmetrical algorithms. If there is no entry, a sequence of 0x00 bytes which is suitable for the respective algorithm is used as an initialisation vector.
KeyEncMacRef	Contains the key reference to the symmetrical key pair (for encryption and MAC computation). The KeyRefType is described on page 16.
SecurityEnvironmentIdentifier	Is an optional element by means of which a security environment which deviates from the standard MAY be specified.
ICCSNRef	<p>MAY contain a reference to the serial number of the card. If this reference is known otherwise, the entry MAY be omitted here.</p> <p>The DataRefType is defined as follows:</p> <div data-bbox="627 1507 1157 1787" data-label="Diagram"> <pre> classDiagram class DataRefType { DataSetName type = iso:DataSetNameType DSIName type = iso:DSINameType } DataRefType "0..1" </pre> </div> <p>If it is a transparent file, the DSIName MAY be omitted.</p>

3.2.2 DIDCreate

DIDCreate uses DIDCreationData of type MutualAuthMarkerType.

3.2.3 DIDUpdate

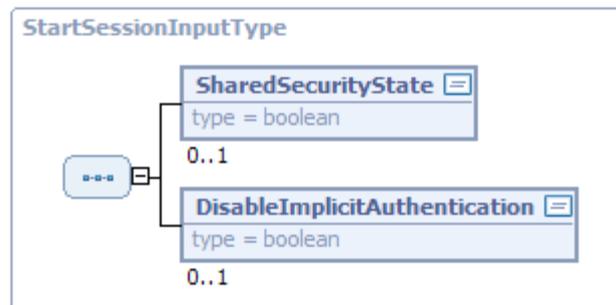
DIDUpdate uses DIDUpdateData of type MutualAuthMarkerType.

3.2.4 DIDGet

DIDGet uses DIDDiscoveryData of type MutualAuthMarkerType.

3.2.5 CardApplicationStartSession

A session to the ICC is established when CardApplicationStartSession is invoked with the optional parameter of the StartSessionInputType, which MAY be subsequently used with the ConnectionHandle returned in the StartSessionOutputType.



This type specifies the structure of the DIDAuthenticationDataType when CardApplicationStartSession is called up.

Name	Description
SharedSecurityState	If this flag is set to True, any channel which may have been established between the client application (or a SAM assigned to it) and the ICC is also used with the returned ConnectionHandle. If no such channel exists then one is established. If this element is missing or if it is False, a new logical channel is always established to the ICC. If the card does not support any logical channels, an error /resultminor/sal#functionNotSupported is returned.
DisableImplicitAuth	If authentication is necessary for access to the specified DIDs, this MAY be implicitly initiated by default (if this element is missing or is False). If this flag is set to True, the necessary authentication is not implicitly initiated and hence the error / resultminor/sal#securityConditionsNotSatisfied MAY occur.



This type specifies the structure of the `DIDAuthenticationDataType` in `CardApplicationStartSessionResponse`.

Name	Description
ConnectionHandle	Enables use of the session opened within <code>CardApplicationStartSession</code> in future function calls. If the call does not create a new logical channel to the ICC (cf. <code>SharedSecurityState</code> above) this element MAY be omitted.

The procedure for setting up a session is approximately as follows:

1. Identify DID information for ICC by means of `DIDGet` and read out ICCSN by means of `DataSetSelect` and `DSIRead`.
2. Request a random number from ICC, form the challenge from the random number and ICCSN and invoke `DIDAuthenticate` for `InternalAuthenticate` on SAM.
3. Invocation of `DIDAuthenticate` for `MutualAuthenticate` on ICC with result from step 2.
4. Invocation of `DIDAuthenticate` for `ExternalAuthenticate` on SAM with result from Step 3.

3.2.6 DIDAuthenticate

`DIDAuthenticate` is used in this protocol for the following purposes:

- To invoke `InternalAuthenticate`
- To invoke `MutualAuthenticate`
- To invoke `ExternalAuthenticate`

3.2.6.1 To invoke InternalAuthenticate



This type specifies the structure of the `DIDAuthenticationDataType` for the Mutual Authentication protocol when `DIDAuthenticate` is invoked to request INTERNAL AUTHENTICATE.

Name	Description
Challenge	Contains the challenge of the communication partner which is to be encrypted and assigned a MAC when INTERNAL AUTHENTICATE is requested.

3.2.6.2 To invoke MutualAuthenticate

The return to `DIDAuthenticateResponse` after INTERNAL AUTHENTICATE, which also serves as the input for the next step (`DIDAuthenticate` for MUTUAL AUTHENTICATE), is as follows in this case:



This type specifies the structure of the `DIDAuthenticationDataType` for the Mutual Authentication protocol when `DIDAuthenticate` is returned after INTERNAL AUTHENTICATE or before MUTUAL AUTHENTICATE.

Name	Description
<code>InternalCryptogram</code>	Contains the cryptogram generated in the previous step.

3.2.6.3 To invoke ExternalAuthenticate

The return to `DIDAuthenticateResponse` after MUTUAL AUTHENTICATE, which also serves as the input for the next step (`DIDAuthenticate` for EXTERNAL AUTHENTICATE), is as follows in this case:



This type specifies the structure of the `DIDAuthenticationDataType` for the Mutual Authentication protocol when `DIDAuthenticate` is invoked to request EXTERNAL AUTHENTICATE.

Name	Description
<code>MutualCryptogram</code>	Contains the cryptogram generated in the previous step.

3.2.7 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when called up:

- Encipher
- Decipher
- Hash
- Sign
- VerifySignature

-
- `VerifyCertificate`

3.2.8 Minimum requirements in terms of algorithms

This authentication protocol MAY be used with various cryptographic algorithms, but the following algorithms MUST be supported as a minimum in accordance with [eGK-1]:

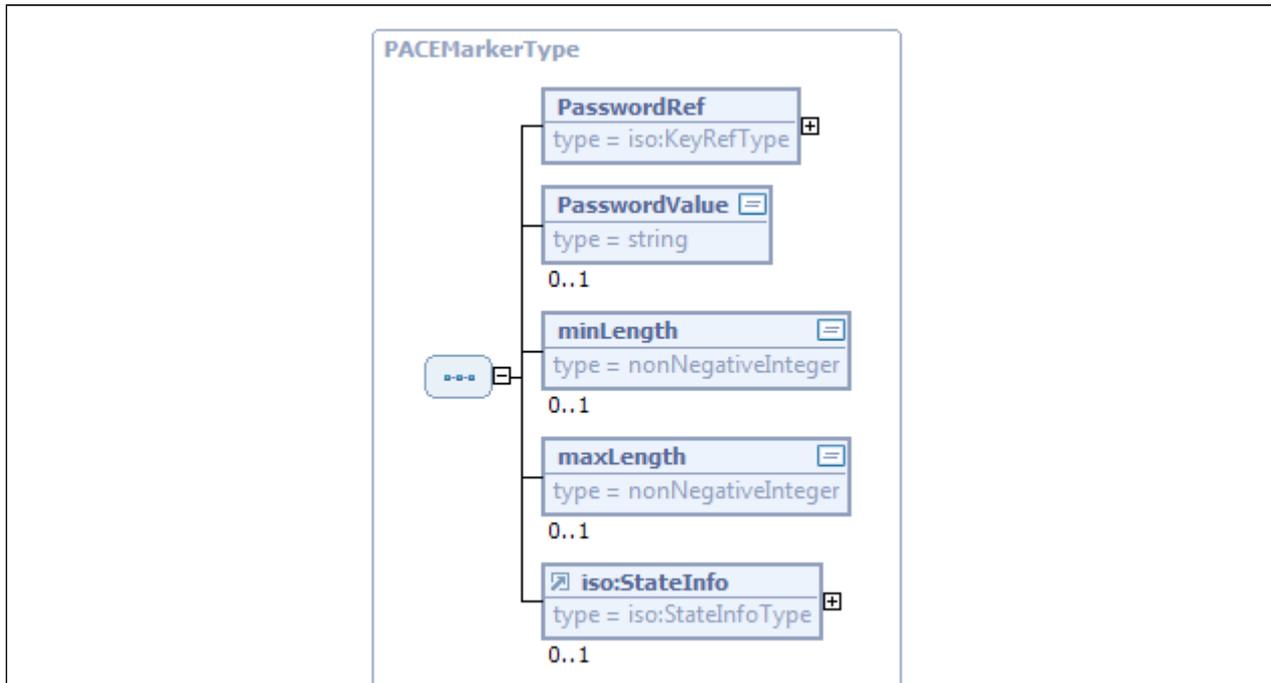
- **EncryptionAlgorithm**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsdsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7) }
- **MacAlgorithm**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsdsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7) }
- **DerivationAlgorithmSessionKeysAndCounter**
[urn:oid:1.2.840.63.0](#) for **x9-63-scheme** ::= { iso(1) member-body(2) US(840) ansi-x9-63(63) schemes(0) }
- **MacAlgorithmForSessionKey**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsdsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7) }
- **EncryptionAlgorithmForSessionKey**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsdsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7) }

3.3 Password Authenticated Connection Establishment

This protocol, defined in [TR-03110], provides a secure connection establishment between terminal and chip card based on weak passwords, e.g. PIN.

This protocol is identified by the URI `urn:oid:0.4.0.127.0.7.2.2.4`.

3.3.1 Marker



This type specifies the structure of the DID marker for the PACE protocol.

Name	Description
PasswordRef	Contains the key reference of the PACE password.
PasswordValue	MAY contain the value of the password. If this element is missing, it is captured at the terminal.
minLength	MAY contain the minimum length of the password.
maxLength	MAY contain the maximum length of the password.
iso:StateInfo	MAY contain information on the designated states if more than one state is defined for the password object. Further details on the iso:StateInfo element are provided on page 17.

The DIDStateQualifier (see [TR-03112-4]) contains the Certificate Holder Authorization Template as defined in [TR-03110], i.e. including tag and length coding.

3.3.2 DIDCreate

DIDCreate uses DIDCreationData of type PACEMarkerType. If the password is missing, it is captured at the terminal or SAL.

3.3.3 DIDUpdate

The DIDUpdate function for the PACE protocol uses DIDUpdateData of the PACEMarkerType and is used to change the status of the password object (e.g. to reset the retry counter) or to change a PACE password.

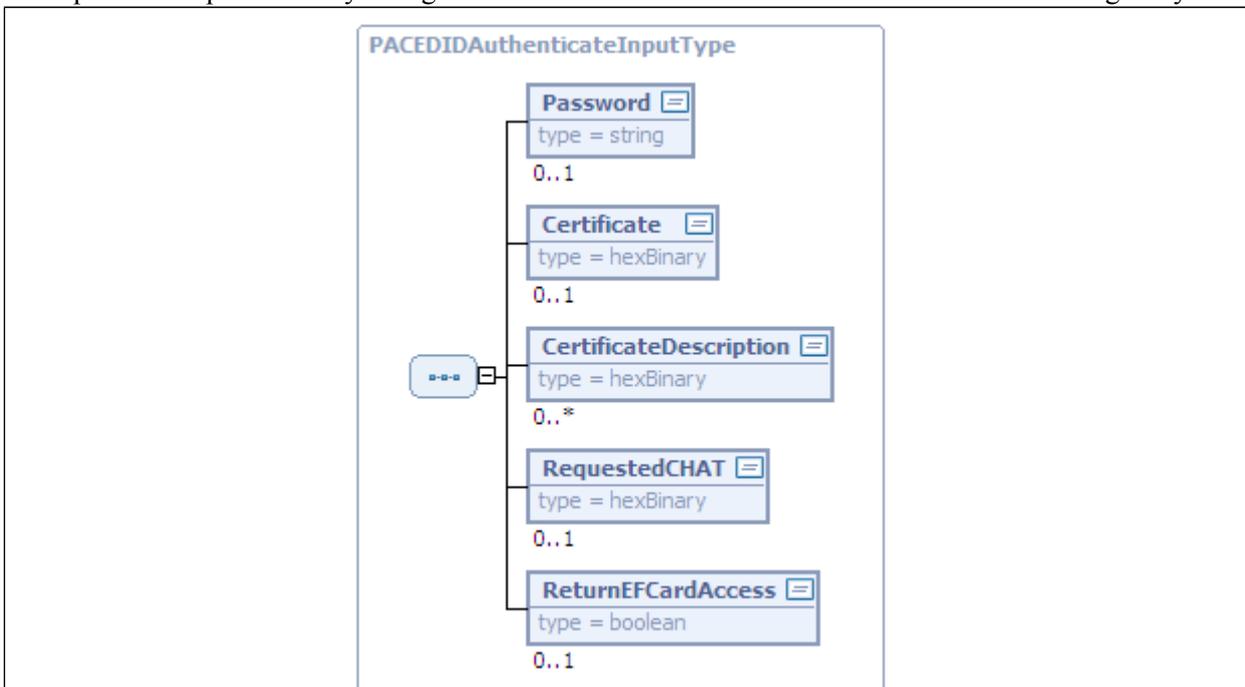
If the current state is not operational it is necessary to initiate appropriate operations (cf. `StateTransition`) to move to an operational state. Once an operational state has been reached, the password MAY be changed. If the `PasswordValue` element is missing it is captured at an appropriate terminal.

3.3.4 DIDGet

In the case of `DIDGet` the `DIDDiscoveryData` of type `PACEMarkerType` are returned. Here the `PasswordValue` element is omitted, and the current value of a `RetryCounter` or `UsageCounter` is provided in the first `State` element.

3.3.5 DIDAuthenticate

The protocol is processed by a single invocation of `DIDAuthenticate` with the following entry:

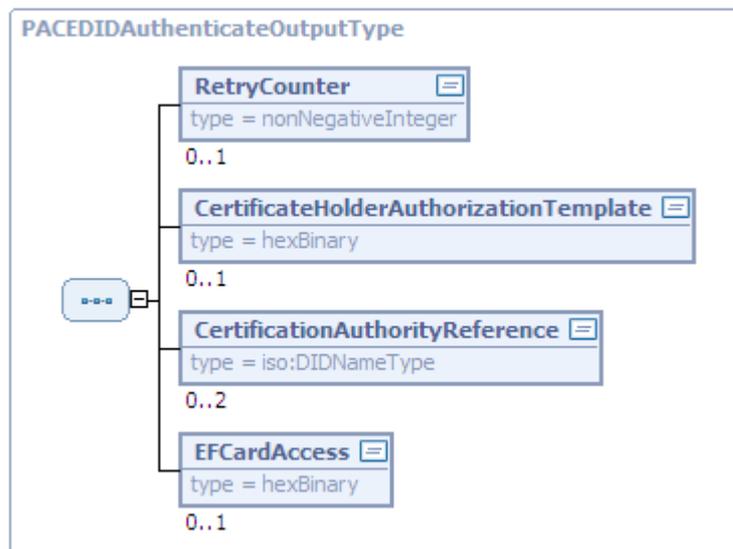


The `PACEDIDAuthenticateInputType` specifies the structure of the `DIDAuthenticationDataType` for the PACE protocol when `DIDAuthenticate` is invoked.

Name	Description
Password	<p>MAY contain the value of the password. If this element is missing, it is captured at the terminal or SAL. In the case of a remote terminal any password transmitted in this element is ignored and a warning (../sal/PACE#PasswordIgnoredWarning) is returned.</p> <p>If the password entry fails, this MUST be duly displayed to the user on the client system so that the user MAY re-enter the password and, if necessary, reset the retry counter (cf. [TR-03110], Section 3.3) or cancel the entry.</p> <p>In the process the user SHOULD also be informed of the remaining</p>

	number of attempts allowed to enter the password and, where applicable, be given the opportunity to reset the operating error counter. If the password entry process is cancelled after an incorrect entry and the number of remaining attempts to enter the password correctly is therefore known in the PICC SAL, this MUST be returned in the <code>RetryCounter</code> element.
<code>Certificate</code>	MAY contain the certificate of the terminal if a Terminal Authentication is to follow PACE. The relevant contents of the certificate (at least the Certification Authority Reference, Certificate Holder Reference, Certificate Holder Authorization Template (CHAT) including user-friendly display of rights, Certificate Effective Date, Certificate Expiration Date and, where present, Certificate Extensions) MUST be duly displayed to the user before the password is entered. The user MUST also be given the opportunity at this point to impose further restrictions on the CHAT and therefore on the effective access rights to the terminal or to cancel the operation without entering a password. The eCard-API-Framework MUST provide an appropriate user interface for these purposes.
<code>CertificateDescription</code>	MAY contain a series of descriptors of the certificate (<code>CertificateDescription</code> , cf. [TR-03110], Annex C.3.1), the hash value of which is contained in the certificate as an extension.
<code>RequestedCHAT</code>	If the full rights specified in the certificate should not be used, a CHAT already restricted by the eService MAY be transferred to the user. It SHOULD be possible, applying the principle of data economy, to configure which CHAT is to be transferred with which certificates and in which cases.
<code>ReturnEFCardAccess</code>	MAY be used to request the return of the ASN.1-encoded <code>SecurityInfos</code> from the <code>EF.CardAccess</code> file (cf. [TR-03110], Table A.1). If this element is absent or <code>FALSE</code> , no <code>SecurityInfos</code> are returned. If this element is set but no <code>EF.CardAccess</code> file is available on the card, a warning is returned (<code>../sal/PACE#EFCardAccessNotFoundWarning</code>).

The return in `DIDAuthenticateResponse` is of the `PACEDIDAuthenticateOutputType` and is as follows:



This type specifies the structure of the `DIDAuthenticateDataType` for the PACE protocol when `DIDAuthenticate` is returned.

Name	Description
RetryCounter	If the user verification process failed, this contains the current value of the <code>RetryCounter</code> .
CertificateHolder AuthorizationTemplate	If the user has imposed further restrictions on the CHAT transmitted by the eService, such that the actual access rights do not correspond with the access rights which might potentially ensue from the certificate, the eService SAL SHOULD be informed of the CHAT restricted by the user in this manner.
CertificationAuthority Reference	Contains up to two references to the certification authority, which MAY be used in the context of the Terminal Authentication to verify the terminal certificate. If two references are returned, the first reference is the more current of the two.
EFCardAccess	MAY contain the ASN.1-encoded <code>SecurityInfos</code> from the <code>EF.CardAccess</code> file (cf. [TR-03110], Table A.1).

3.3.6 CardApplicationStartSession

The execution of the PACE protocol MAY also be initiated by invoking `CardApplicationStartSession`. In this case the `AuthenticationProtocolData` elements are of the `PACEDIDAuthenticateInputType` described above on invocation and of the `PACEDIDAuthenticateOutputType` on their return.

3.3.7 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when invoked:

- Encipher

- Decipher
- GetRandom
- Hash
- Sign
- VerifySignature
- VerifyCertificate

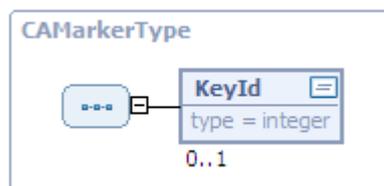
3.4 Chip Authentication

Chip Authentication is defined in [TR-03110]. Two versions of this protocol must be distinguished:

- The Chip Authentication in Version 2 MUST be preceded Terminal Authentication, and therefore the fresh key pair does not have to be generated during Chip Authentication.
- For the generation of secure messaging keys, K_{Enc} and K_{MAC} , a random value $r_{PICC,CA}$ selected by the PICC is included.
- In Version 2 an explicit authentication process occurs, whereas authentication in Version 1 is implicit.

This protocol is identified by the URI `urn:oid:0.4.0.127.0.7.2.2.3`.

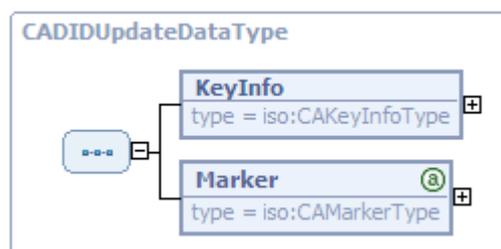
3.4.1 Marker



This type specifies the structure of the DID marker for the Chip Authentication protocol.

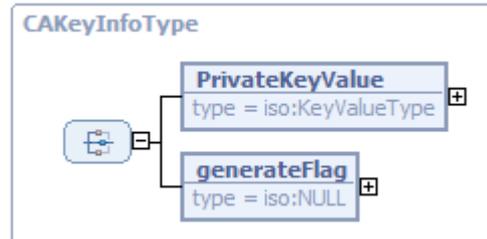
Name	Description
KeyId	MAY contain the local key identifier, if the PICC provides multiple public keys for Chip Authentication..

3.4.2 DIDCreate



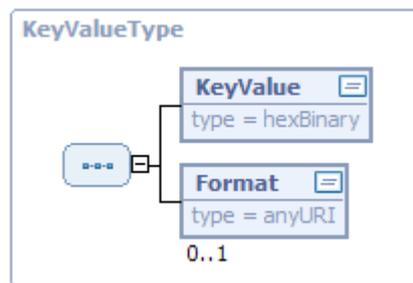
This type specifies the structure of the DIDUpdateDataType for the Chip Authentication protocol.

Name	Description
KeyInfo	Contains information on the private key for the Chip Authentication protocol (details on CAKeyInfoType are given below).
Marker	Contains the marker for the Chip Authentication protocol (see Section 3.4.1). If the generateFlag alternative is selected in the KeyInfo element described below, and if the CAPublicKey element is also contained in the Marker (see above), this is ignored and a corresponding warning is returned (../sal/ChipAuth#PublicKeyIgnoredWarning).



The CAKeyInfoType is used for the definition of DIDUpdateDataType.

Name	Description
PrivateKeyValue	The private key for the Chip Authentication protocol MAY be imported by means of this element. Details on the KeyValueType are provided below.
generateFlag	This element MAY be used to initiate random generation of the private key in the card.



The KeyValueType contains key information and an optional format specification.

Name	Description
KeyValue	Contains the value of the key (in the specified format).
Format	MAY contain the URI of the format employed.

3.4.3 DIDUpdate

In the case of DIDUpdate the DIDUpdateData are of Type CAUpdateDataType defined above.

3.4.4 DIDGet

In the case of DIDGet there is a return of DIDStructure containing data of the type CAMarkerType.

3.4.5 DIDAuthenticate

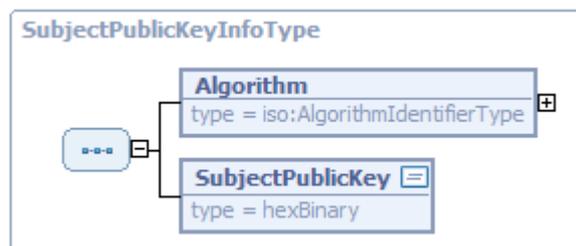
The Chip Authentication protocol is implemented by the following request sequence:

1. DIDGet MAY be used to obtain the Marker (cf. Section 3.4.1) of the card which especially contains the domain parameters and the public key.
2. An ephemeral key pair is then generated in Version 1 of the protocol using the domain parameters. In Version 2 of the protocol the key pair was previously generated in the scope of Terminal Authentication. In each case DIDAuthenticate is now invoked on the PICC SAL by the eService SAL, whereby the AuthenticationProtocolData element is of the CAInputType (see below) and contains the public key $P\tilde{K}_{PCD,CA}$ of the terminal.
3. The card and the terminal are now able to calculate the key that was jointly agreed on. The card also calculates the hash value of the public key of the terminal and, if Version 2 of the Chip Authentication protocol is running, compares this with the key received from the terminal during Terminal Authentication. In this case (Version 2), in response to the DIDAuthenticate request, a AuthenticationProtocolData element of type CAAuthenticationTokenType, which is explained below, is sent by the card in step 2. Otherwise (Version 1) the returned AuthenticationProtocolData is empty.
4. Apart from this, Passive Authentication MUST be performed directly after (Version 1) or before (Version 2) the execution of the Chip Authentication protocol to ensure the authenticity of the card's public key. The SAL (of the eService) is responsible for checking the signature stored in EF.CardSecurity and, where applicable, for checking corresponding certificates up to a trustworthy root.



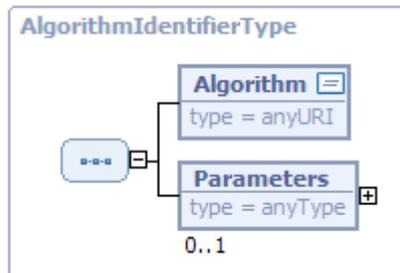
This type specifies the structure of the CAInputType used in step 2.

Name	Description
PublicKey	Contains the public key of the terminal. The structure of the SubjectPublicKeyInfoType is explained below.



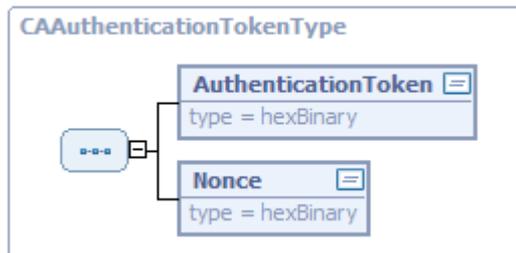
The PublicKey element above and the RootKey element in the TADIDUpdateDataType (cf. page 41) is of the SubjectPublicKeyInfoType.

Name	Description
Algorithm	Specifies the algorithm used. The structure of the AlgorithmIdentifierType is described below.
SubjectPublicKey	Contains the public key.



The AlgorithmIdentifierType is used for the definition of the SubjectPublicKeyInfoType above.

Name	Description
Algorithm	Contains the URI of the algorithm.
Parameters	Contains the parameters required for the respective algorithm.



This type specifies the structure of the CAAuthenticationTokenType used in step 3.

Name	Description
AuthenticationToken	Contains the authentication token (T_{PICC}).
Nonce	Contains the random number ($r_{PICC,CA}$).

3.4.6 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when called:

- CardApplicationStartSession
- Encipher
- Decipher
- GetRandom
- Hash

- Sign
- VerifySignature
- VerifyCertificate

3.5 Terminal Authentication

Terminal Authentication is defined in [TR-03110]. Two versions of this protocol must be distinguished:

- In Version 2 of Terminal Authentication a key pair is generated and committed to for use in a subsequent Chip Authentication Version 2.

This protocol is identified by the URI `urn:oid:0.4.0.127.0.7.2.2.2`.

3.5.1 Marker

There is only an empty DID marker for the Terminal Authentication protocol, because the necessary information is entirely contained in the standardized EF.CardAccess and EF.CardSecurity files.

The `DIDStateQualifier` (see [TR-03112-4]) contains the Certificate Holder Authorization Template as defined in [TR-03110], i.e. including tag and length coding.

3.5.2 DIDCreate



This type specifies the structure of the `DIDUpdateDataType` for the Terminal Authentication protocol.

Name	Description
RootKey	Contains the trusted root key. The structure of the <code>SubjectPublicKeyInfoType</code> is explained on page 39.

3.5.3 DIDUpdate

`DIDUpdate` uses the `DIDUpdateDataType` defined above.

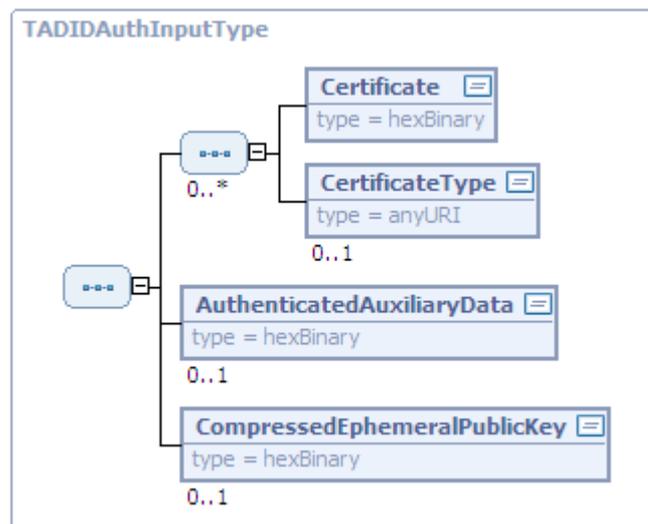
3.5.4 DIDGet

In the case of `DIDGet` there is a return of `DIDStructure` containing data of the type `TAMarkerType`.

3.5.5 DIDAuthenticate

The protocol is implemented by the following requests of DIDAuthenticate from the terminal to the card:

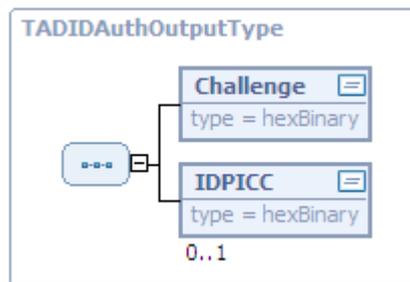
1. When DIDAuthenticate is first invoked with AuthenticationProtocolData of the TADIDAuthInputType, the certificate chain is transmitted along with the hash value (in Version 2) of the newly generated public key as well as any other auxiliary data which may require authentication (e.g. for age verification).
2. The returned AuthenticationProtocolData are of type TADIDAuthOutputType and contain the "card identity" ID_{PICC} (IDPICC) and the random number $r_{PICC,TA}$ (Challenge).
3. When DIDAuthenticate is invoked for the second time with AuthenticationProtocolData of the TADIDAuthExternalAuthType, the signature generated by the terminal is ultimately transmitted to the card for verification.



The type specifies the structure of the TADIDAuthInputType used in the first step.

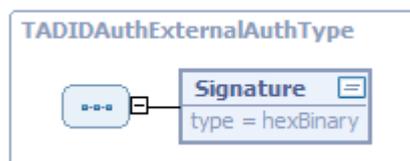
Name	Description
Certificate	Contains a certificate.
CertificateType	MAY specify the type of certificate (cf. VerifyCertificate in [TR-03112-4]).
Authenticated AuxiliaryData	MAY contain additional data which are used to check the validity of the card or for age verification. These data MUST be provided in the form specified in [TR-03110], Table A.13. For each piece of data transmitted for additional verification after successful Terminal Authentication, a Verify command is requested (e.g. with OID 0.4.0.127.0.7.3.1.4.1 (id-auxiliaryData-1) for age verification, with 0.4.0.127.0.7.3.1.4.2 (id-auxiliaryData-2) for the document validity verification or with 0.4.0.127.0.7.3.1.4.3 (id-auxiliaryData-3) to verify municipality citizenship, cf. also [TR-03110], Section B.11.8). The corresponding result of these verification steps is returned in the Result element of the response of the second DIDAuthenticate request (cf. also TADIDAuthExternalAuthType, page 43).

	If the specified DID only supports Version 1 of the Terminal Authentication protocol, the warning .../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning is returned.
CompressedEphemeralPublicKey	MAY contain the compressed public key of the key pair which has been newly generated by the terminal. In accordance with [TR-03110], Table A.2, this MAY either be the SHA-1 hash value (Diffie-Hellman) or the x -coordinate of the group element (Elliptic Curve Diffie-Hellman). If the specified DID only supports Version 1 of the Terminal Authentication protocol, the warning .../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning is returned.



This type specifies the structure of the TADIDAuthOutputType returned by the card in step 2.

Name	Description
Challenge	Contains the random number generated by the card $r_{PICC,TA}$.
IDPICC	MAY contain the "card identity" ID_{PICC} . As stipulated in [TR-03110], Section 4.4, this involves the Doc# from the MRZ in case of BAC or the compressed ephemeral public keys of the PICC in case of PACE. In the case of a loyal-stack configuration, where the card identity is already known to the terminal, this element MAY be omitted.



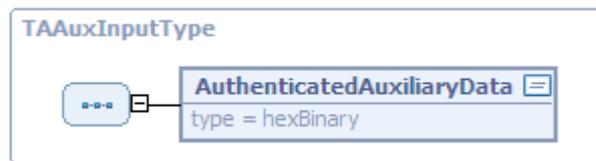
This type specifies the structure of the TADIDAuthExternalAuthType sent in step 3, which is verified by the card.

Name	Description
Signature	Is the signature of the terminal which has to be verified by the card in the scope of Terminal Authentication. The results of the verification of this signature and, where applicable, of the additional verification steps (cf. Authenticated AuxiliaryData, page 42) are returned following this invocation of DIDAuthenticate. If

	<p>these additional verification steps fail, the following warnings</p> <ul style="list-style-type: none"> • .../sal/EAC#AgeVerificationFailedWarning • .../sal/EAC#DocumentValidityVerificationFailedWarning • .../sal/EAC#CommunityVerificationFailedWarning <p>are returned.</p>
--	--

In addition, the VERIFY commands which may be required to check the additional data for the PICC validity check or for the age verification by means of APDUs protected by secure messaging and the Transmit function are sent to the card (cf. [TR-03110], Annex B.11.8).

For cards which – *unlike the German identity card* – would allow to perform the additional data verification outside a secure messaging channel, it *would* be possible to perform these checks with an invocation of DIDAuthenticate using the TAAuxInputType.



This type specifies the structure of the TAAuxInputType which *would* be used in DIDAuthenticate to verify additional data if the card allows to verify additional data without secure messaging.

Name	Description
Authenticated AuxiliaryData	Contains the data also requiring checking in the form specified in Table A.13 of [TR-03110] (cf. Authenticated AuxiliaryData, page 42).

3.5.6 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when invoked:

- CardApplicationStartSession
- Encipher
- Decipher
- GetRandom
- Hash
- Sign
- VerifySignature

3.6 Extended Access Control

This protocol specified in [TR-03110] forms the framework for mutual authentication with keys exchanged using the Extended Access Control protocol.

The identifier for this protocol is [urn:oid:1.3.162.15480.3.0.14](#) for iso(1) identified-organization (3) CEN (162) CEN 15480 (15480) part3(3) annex-a(0) extended-access-control-protocol(14)

The following protocol variants and URIs are available for this purpose:

- [urn:oid:1.3.162.15480.3.0.14.1](#) for EAC Version 1 in accordance with [TR-03110], which comprises the following sub-protocols:
 - Basic Access Control in accordance with [ICAO 9303]
 - Chip Authentication version 1 in accordance with [TR-03110]
 - Terminal Authentication version 1 in accordance with [TR-03110]
- [urn:oid:1.3.162.15480.3.0.14.2](#) for EAC Version 2 in accordance with [TR-03110], which comprises the following sub-protocols:
 - Password Authenticated Connection Establishment (PACE) in accordance with [TR-03110]
 - Chip Authentication version 2 (CA) in accordance with [TR-03110]
 - Terminal Authentication version 2 (TA) in accordance with [TR-03110]
 - Restricted Identification (RI) in accordance with [TR-03110]

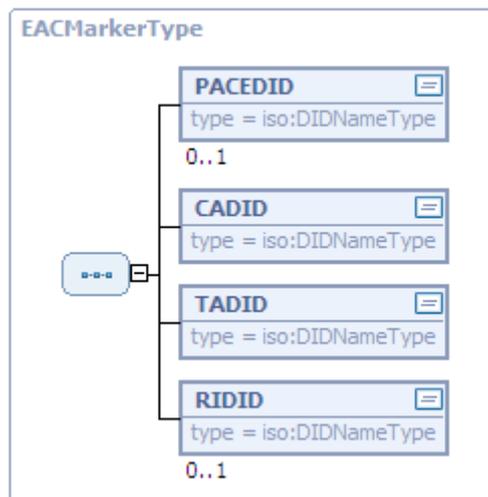
3.6.1 EAC protocol specification

The Extended Access Control (EAC) protocol is specified in the following sections:

- Marker
- Call and return of `CardApplicationStartSession`
- Overview of EAC protocol sequence
- Phase 1 - Extended PACE protocol
- Phase 2 - Combination of Terminal and Chip Authentication
- Phase 3 - Secure messaging with APDU batches

3.6.2 Marker

A DID for the EAC protocol has the following marker structure, making reference only to existing DIDs for the PACE, Chip Authentication, Terminal Authentication and, where applicable, the Restricted Identification protocol.



This type specifies the structure of the DID marker for the EAC protocol.

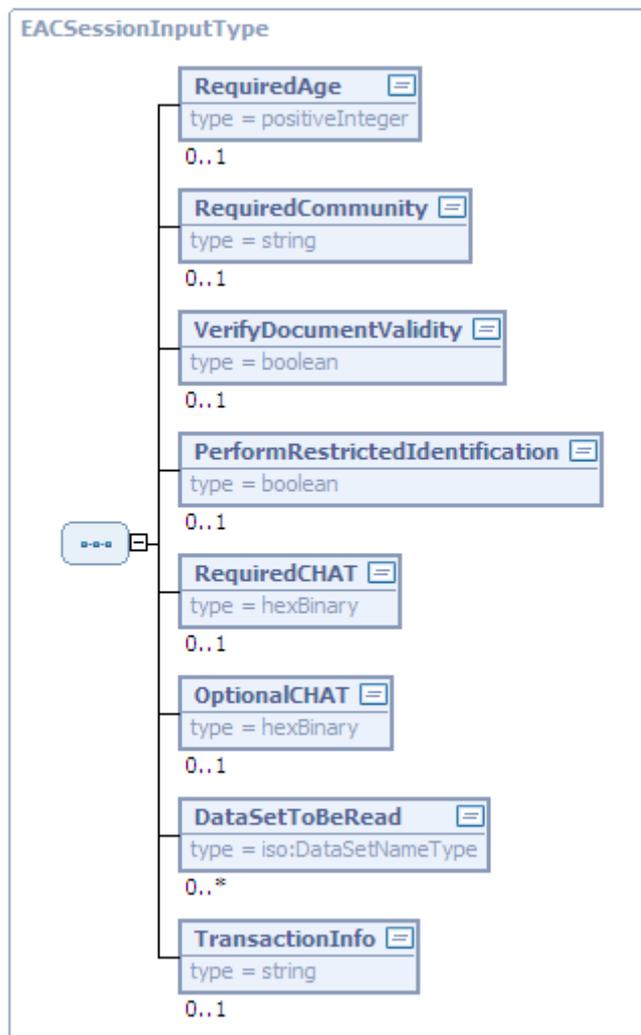
Name	Description
PACEDID	Contains a reference to a DID for the PACE protocol in Version 2 (details on such DIDs are given in Section 3.3). However, in Version 1 of the EAC protocol, the element is omitted and the MRZ is always used to derive the key for the BAC protocol.
CADID	Contains a reference to a DID for the Chip Authentication protocol (details on such DIDs are given in Section 3.4).
TADID	Contains a reference to a DID for the Terminal Authentication protocol (details on such DIDs are given in Section 3.5).
RIDID	MAY contain a reference to a DID for the Restricted Identification protocol (details on such DIDs are given in Section 3.7).

The `DIDStateQualifier` (see [TR-03112-4]) contains the Certificate Holder Authorization Template as defined in [TR-03110], i.e. including tag and length coding.

3.6.3 Call and return of `CardApplicationStartSession`

The protected channel to the card by means of EAC is established by requesting `CardApplicationStartSession` with a corresponding DID for the EAC protocol. In this context the `DIDName` refers to the DID on the PICC with the marker structure defined in Section 3.6.2. The `AuthenticationProtocolData` are of type `EACSessionInputType` explained in more detail below, through which the optional test sequences for age verification, document validity and municipality citizenship MAY be specified and / or the generation of a sector-specific pseudonym MAY be requested.

If required, a differentiation MAY also be made between different eService keys (with different certificates and authorisations) using the `SAMConnectionHandle`. Handles for the keys/certificates, which are currently available to the eService SAL are returned without additional parameters when `CardApplicationPath` is called.

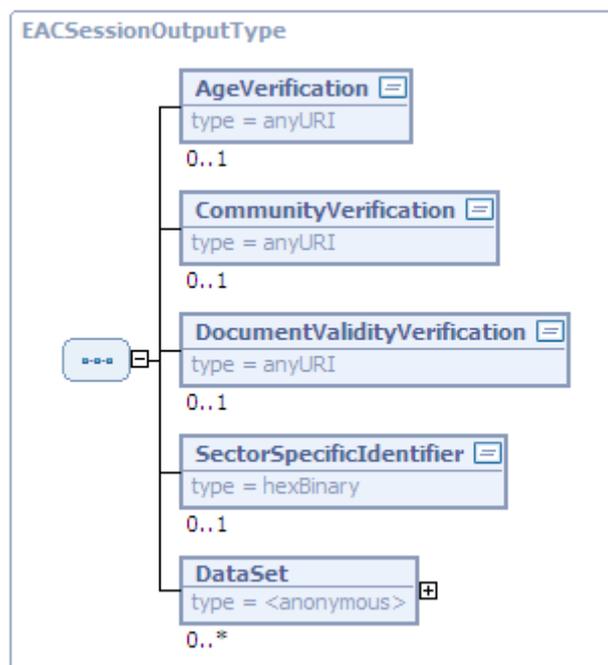


This type specifies the structure of the AuthenticationProtocolData when CardApplicationStartSession is called up with the EAC protocol.

Name	Description
RequiredAge	<p>MAY be used for age verification with a specific minimum age. If this element is missing, the age is not verified. This element is converted by the eService SAL into the format required for the PICC (cf. Authenticated-AuxiliaryData in TADIDAuthInputType in Section 3.5.5 and [TR-03110], Section A.6.5).</p> <p>If the age verification process fails, a warning is returned in the AgeVerification element explained below (../sal/mEAC#AgeVerificationFailedWarning).</p> <p>If the specified DID is provided in EAC Version 1, the warning returned is ../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning.</p>
RequiredCommunity	<p>MAY be used to check whether the citizen is affiliated with a certain municipality.</p> <p>If the community affiliation process fails, a warning is returned in the</p>

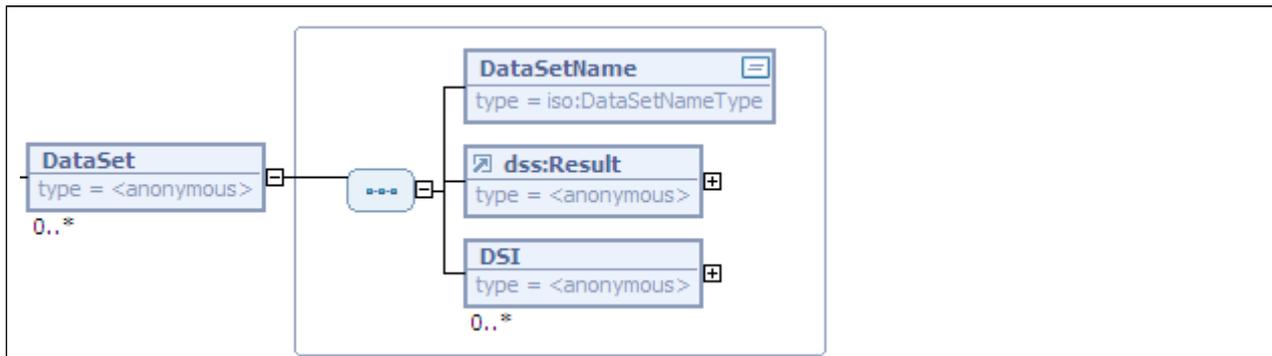
	<p>CommunityVerification element explained below (../sal/mEAC#CommunityVerificationFailedWarning).</p> <p>If the specified DID is provided for EAC Version 1, the warning returned is ../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning.</p> <p>If this element is missing, the citizenship is not checked.</p>
VerifyDocumentValidity	<p>MAY specify whether the current document validity will be checked. If this element is missing or FALSE, the document validity is not checked. This element is converted by the eService SAL into the format required for the PICC (cf. Authenticated-AuxiliaryData in TADIDAuthInputType in Section 3.5.5 and [TR-03110], Section A.6.5).</p> <p>If the document validity check fails, the warning (../sal/mEAC#DocumentValidityVerificationFailed) is returned.</p> <p>If the specified DID is provided for EAC Version 1, the warning returned is ../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning.</p>
PerformRestrictedIdentification	<p>MAY specify whether the sector-specific pseudonym is to be calculated once the trustworthy channel has been established between the eService and PICC with the Restricted Identification protocol.</p> <p>If the specified DID is provided for EAC Version 1, the warning returned is ../sal/FunctionalityByCurrentProtocolVersionNotSupportedWarning.</p>
RequiredCHAT	<p>If the eService does not want to use the full access rights provided by the CHAT of the addressed certificate or leave it up to the configuration of the eService-SAL, it MAY explicitly specify the required CHAT here. If the user does not allow the required rights, the connection establishment fails.</p>
OptionalCHAT	<p>In a similar manner the eService MAY specify optional access rights, which are not strictly necessary to establish the connection.</p>
DataSetToBeRead	<p>In order to minimise the number of messages which have to be sent via the network, it is possible to send the necessary APDUs for the Restricted Identification protocol together with the APDUs for the data readout in one single Transmit request (cf. [TR-03112-6]). To allow this, the data groups which are to be read out must be specified when invoking CardApplicationStartSession, with one DataSetToBeRead element available for each data set to be read out.</p>
TransactionInfo	<p>This element MAY contain transaction-related information, which MUST be displayed in the eID-PIN dialogue before the PACE-protocol is performed.</p>

In response to the CardApplicationStartSession request, a CardApplicationStartSessionResponse is returned with AuthenticationProtocolData of the EACSessionOutputType.



This type specifies the structure of the AuthenticationProtocolData in CardApplicationStartSessionResponse with the EAC protocol.

Name	Description
AgeVerification	If a RequiredAge element was transferred on invocation, the result of the age verification is returned in this element. If the check was successful, the returned URI is ../resultmajor#ok , whereas the error code returned if the age verification failed is ../sal/EAC#AgeVerificationFailedWarning .
Community Verification	If a RequiredCommunity element was transferred on invocation, the result of the citizenship check is returned in this element. If the check was successful, the returned URI is ../resultmajor#ok , whereas the error code returned if the citizenship check failed is ../sal/EAC#CommunityVerificationFailedWarning .
DocumentValidity Verification	If the document validity check was requested with the VerifyDocumentValidity element, which is assigned the status True, the result of this check is returned in this element. If the check is successful, the returned URI is ../resultmajor#ok , whereas the error code ../sal/EAC#DocumentValidityVerificationFailed is returned if the document validity check fails.
SectorSpecific Identifier	If the PerformRestrictedIdentification element was used to request the calculation of the sector-specific pseudonym, this is returned here.
DataSet	In response to each DataSet request either its content or an error message, indicating that readout was unsuccessful is returned. The precise structure of this element is explained in detail below.



For each DataSetToBeRead element in the request, a corresponding DataSet element is returned.

Name	Description
DataSetName	Contains the name of the DataSet.
dss:Result	<p>Contains the result of the request. If the DataSet readout was successful, the URI returned in the ResultMajor element is .../resultmajor#ok.</p> <p>If the process fails, the URI returned in the ResultMajor element is .../resultmajor#error and, in addition, further details are returned in the ResultMinor element as to the cause of the error, distinguishing between the following cases:</p> <ul style="list-style-type: none"> • /resultminor/sal#unknownDataSetName • /resultminor/sal#securityConditionsNotSatisfied • /resultminor/sal#prerequisitesNotSatisfied
DSI	Is available once for each Data Structure for Interoperability contained in the DataSet. See below for details.



The DSI element is part of DataSet.

Name	Description
DSIName	Contains the name of the Data Structure for Interoperability (DSI).
dss:Result	<p>Contains the result of the request. If the DSI readout was successful, the message returned in the ResultMajor element is .../resultmajor#ok.</p> <p>If the process fails, the message returned in the ResultMajor element is .../resultmajor#error and, in addition, further details are returned in the ResultMinor element as to the cause of the error, distinguishing between the following cases:</p>

	<ul style="list-style-type: none"> • /resultminor/sal#unknownDSIName • /resultminor/sal#prerequisitesNotSatisfied • /resultminor/sal#securityConditionsNotSatisfied
DSIContent	The content of the DSI is returned here if the process is successful.

3.6.4 Overview of EAC protocol sequence

The sequence between both SAL instances after invocation of `CardApplicationStartSession` on the eService SAL is shown Figure 3.

The protected channel between the eService SAL and the PICC is established in the following steps:

1. The eService-SAL invokes `DIDAuthenticate` with the `DIDName` provided for PACE (cf. `PACEDID` element in Section 3.6.2) and `AuthenticationProtocolData` of the `EAC1InputType` explained in more detail below. The eService certificate, the corresponding DV certificate and additional link certificates are transferred in this process and MAY be verified by the client SAL. If the verification fails, the user MUST be informed accordingly and the authentication protocol MUST be aborted. The `DIDAuthenticate`-message also contains corresponding certificate descriptions (see specification of the ASN.1-based `CertificateDescription` structure in [TR-03110], Annex C.3.1), information about the required and optional Card Holder Authorization Template (CHAT) (`RequiredCHAT` and `OptionalCHAT`), the `AuthenticatedAuxiliaryData` prepared for the chip and additional `TransactionInfo`, if required. The contents of the certificate, any existing certificate descriptions

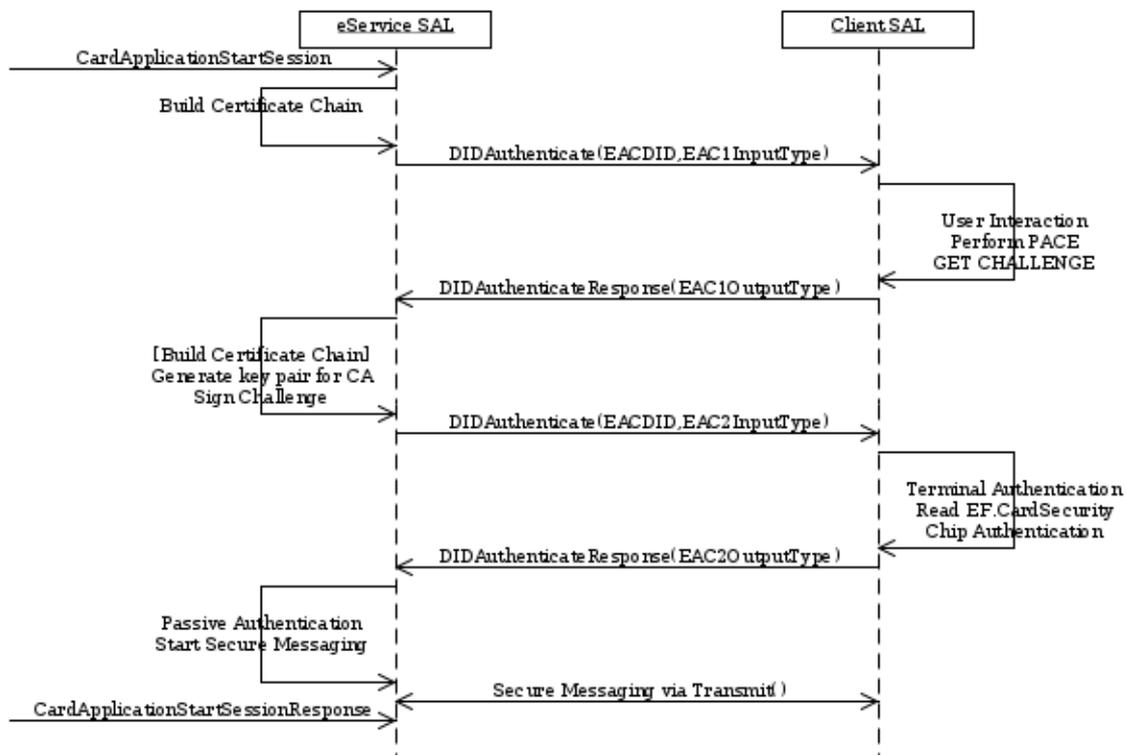


Figure 3: Message Sequence after `CardApplicationStartSession` (EACSession)

(CertificateDescription) and the TransactionInfo -element are duly displayed to the user. The user is also given the opportunity to limit the effective eService access rights before entering the password and thereby signalling consent to the access. The file EF.CardAccess is read out and, following the PACE protocol process, the challenge for the Terminal Authentication is requested by the chip if the PICC is able to check the terminal certificate chain between the challenge request and the signature verification. If this is not possible, the Challenge element is missing from the EAC1OutputType returned and an additional message is required (see step 3). The data described are returned in the AuthenticationProtocolData of type EAC1OutputType, which is explained in more detail below. If this process is successful the ASN.1-encoded SecurityInfo structure from EF.CardAccess and the “card identity” ID_{PICC} (see [TR-03110], Section 4.4) is returned. If the CertificateHolderAuthorizationTemplate (CHAT) has been further restricted by the user it will be returned. If the client SAL has not been able to build a PICC-verifiable certificate chain there will be up to two CertificationAuthorityReference elements, which specify the root keys that are available for the certificate verification on the PICC. The SecurityInfo structure from the EF.CardAccess file notably contains the domain parameters which are used in the next step to generate a fresh key pair.

If the password entry fails, this MUST be duly displayed to the user on the client system so that the user MAY re-enter the password and, if necessary, reset the retry counter (cf. [TR-03110], Section 3.3) or cancel the entry.

In the process, the user SHOULD also be informed of the remaining number of attempts allowed to enter the password and, where applicable, be given the opportunity to reset the retry counter. If the password entry process is cancelled after an incorrect entry and the number of remaining attempts to enter the password correctly is therefore known in the client SAL, this MUST be returned in the RetryCounter element.

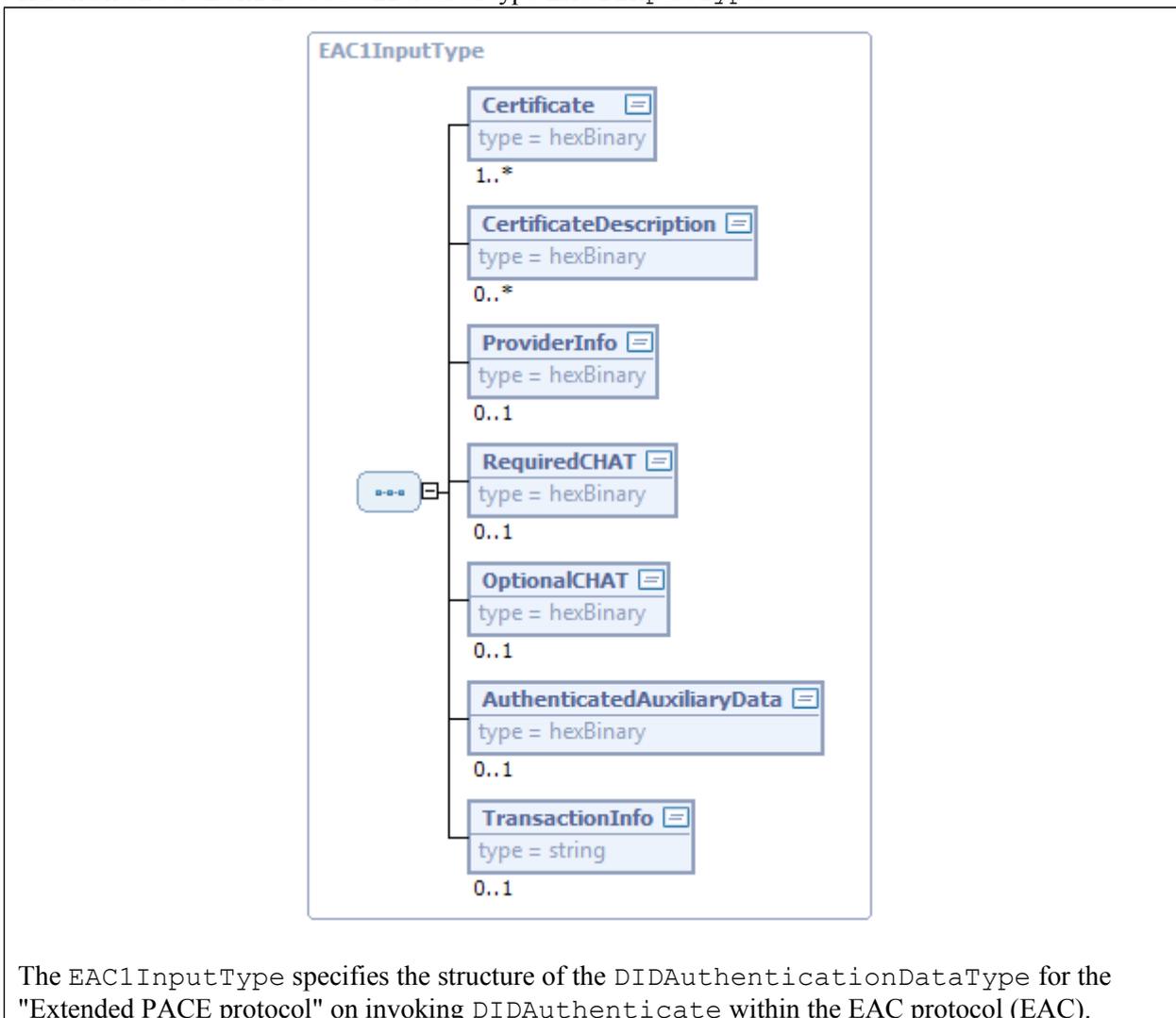
2. Using the Chip Authentication domain parameters (see SecurityInfo structure above), the eService SAL generates a fresh key pair in the next step, forms an appropriate chain of additionally required certificates and finally, where required, signs the Challenge which has been transmitted. The eService SAL then invokes DIDAuthenticate for the CADID (cf. Section 3.6.2) and relays AuthenticationProtocolData of type EAC2InputType, which is described in more detail below, to the PICC SAL. In addition to the certificate chain applicable to the PICC, this element notably contains the newly generated public key EphemeralPublicKey and, if generated, the signature (Signature). The certificate chain is verified by the PICC. Moreover, either the signature is checked by means of EXTERNAL AUTHENTICATE or the Challenge is requested — given that the Challenge could not be requested before the certificate check. If it is already possible at this stage, the file EF.CardSecurity is read out and finally the Chip Authentication is executed by invoking MSE:SET AT and GENERAL AUTHENTICATE. Finally, the result of these actions is returned to the eService SAL in AuthenticationProtocolData of type EAC2OutputType, which is explained in more detail below. This includes either the Challenge -element or the content of the file EF.CardSecurity (EFCardSecurity), the authentication token (AuthenticationToken) and the random number required to check the authentication token (Nonce).
3. (Optional) If the Challenge has not already been returned with the first message, thus preventing the terminal signature from being generated until this time, an additional invocation of DIDAuthenticate with AuthenticationProtocolData of type EACAdditionalMessageType is required to transmit the terminal signature to the PICC where it is checked by invoking EXTERNAL AUTHENTICATE. In this case the file EF.CardSecurity is read out first and finally the Chip Authentication is executed by invoking MSE:SET AT and GENERAL AUTHENTICATE. The result of these actions is then returned to the eService SAL in AuthenticationProtocolData of type EAC2OutputType, which is explained in more detail below. This comprises the content of the file EF.CardSecurity

(EFCardSecurity), the authentication token (AuthenticationToken) and the random number required to check the authentication token (Nonce).

4. If the signature extracted from EFCardSecurity (Passive Authentication) and the authentication token generated in the Chip Authentication process are verified, the eService SAL MAY then communicate with the PICC via APDUs protected by secure messaging in order to — according to the information requested by means of CardApplicationStartSession — request the generation of the sector-specific pseudonym, perform additional checks or read out certain data stored on the PICC. The APDUs required for this MAY be calculated in advance by the eService SAL and transferred as a batch using the Transmit function from [TR-03112-6] via the network to the IFD-Layer on the side of the PICC. The IFD-Layer on the side of the PICC in turn sends the APDUs prepared by the eService SAL to the PICC in sequence and logs the respective response APDUs, which are ultimately sent back to the eService SAL as a collective batch in the TransmitResponse.

3.6.5 Phase 1 - Extended PACE protocol

The "Extended PACE protocol" is realized by one single request of DIDAuthenticate with AuthenticationProtocolData of type EAC1InputType:



The EAC1InputType specifies the structure of the DIDAuthenticationDataType for the "Extended PACE protocol" on invoking DIDAuthenticate within the EAC protocol (EAC).

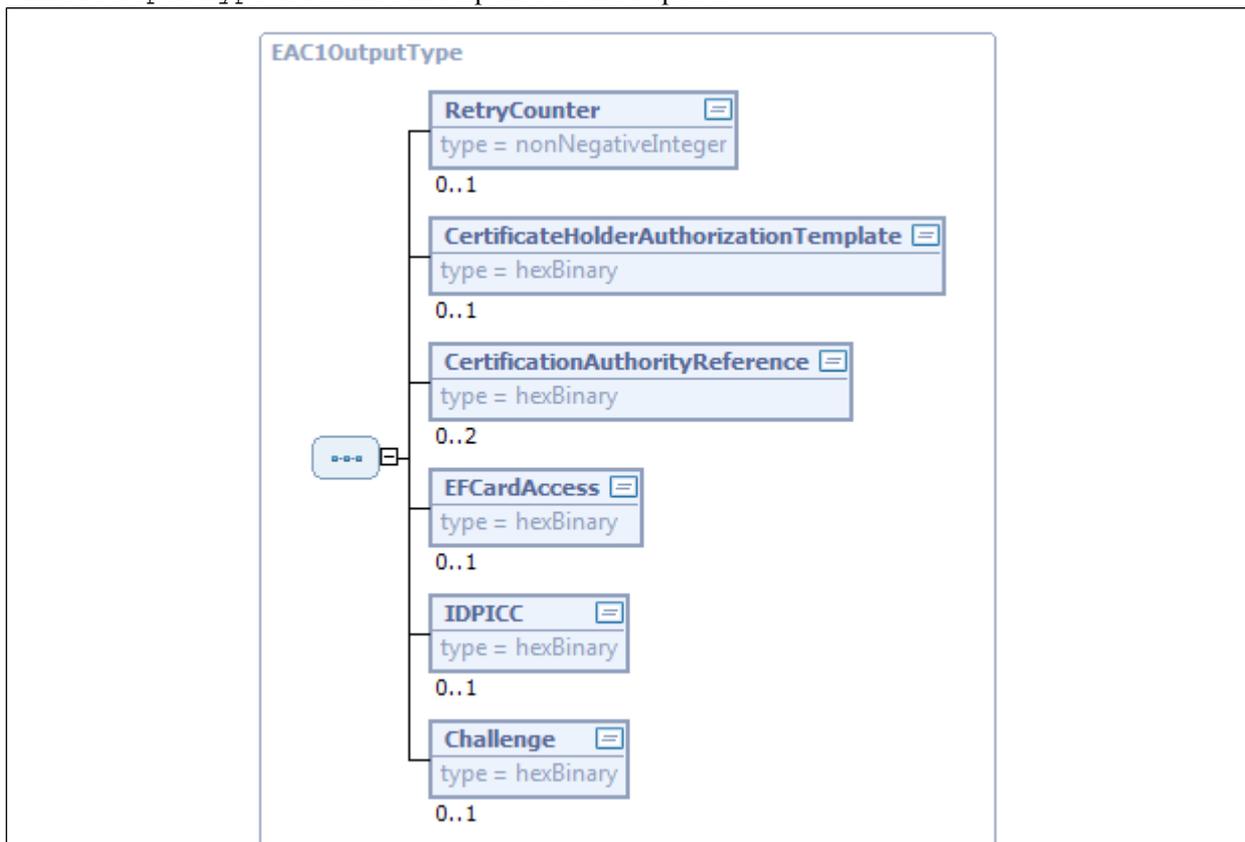
Name	Description
Certificate	<p>MUST contain exactly one eService certificate and the corresponding DV certificate. Additional link-certificates MAY be included. The eService SAL SHOULD include all link-certificates known to the eService. The relevant contents of the eService certificate (at least the Certification Authority Reference, Certificate Holder Reference, Certificate Holder Authorization Template (CHAT) including user-friendly display of rights, Certificate Effective Date, Certificate Expiration Date and, where present, Certificate Extensions) MUST be displayed to the user before the password is entered. The user MUST also be given the opportunity at this point to impose further restrictions on the CHAT and therefore on the effective access rights to the terminal or to cancel the operation without entering a password. The client SAL MUST provide an appropriate user interface for these purposes.</p> <p>The client SAL MAY verify the eService certificate according to [TR-03110], chapter 2.2.5, before display. In this case the client SAL MUST maintain trust point(s) according to [TR-03110], chapter 2.2.5 (the role of MRTD chip must be performed by the client SAL). This includes secure storage of the trust point(s) and update of trust point(s) according to the rules in [TR-03110], chapter 2.2.5. If the verification fails, the user MUST be informed accordingly and the authentication protocol MUST be aborted.</p>
Certificate Description	<p>While the present schema¹ allows an arbitrary number of CertificateDescription- elements there MUST exactly be one such element and the citizen client MUST display the content of this element in a suitable manner before capturing the PIN and performing the PACE-protocol.</p>
ProviderInfo	<p>While the present schema² allows that there is a ProviderInfo -element, this element is deprecated and is ignored if present.</p>
RequiredCHAT	<p>Specifies the data, which are required by the eService.</p> <p>If the full rights specified in the certificate are not supposed to be used, a CHAT already restricted by the eService MAY be transferred to the user. It SHOULD be possible, applying the principle of data-thrift, to configure the eService SAL to dictate which CHAT is transferred with which certificates and in which cases.</p>
OptionalCHAT	<p>Specifies the data, which are requested by the eService, but which transmission may be suppressed by the user.</p>
Authenticated AuxiliaryData	<p>MAY contain additional data which are used to check the validity of the card, verify the age or check municipality citizenship.</p> <p>These data MUST be relayed in the form specified in [TR-03110], Table A.13. For each piece of data transmitted for additional verification after successful Terminal Authentication, a Verify</p>

1 Please note that it is planned to adjust the schema definition in the future such that it allows precisely one CertificateDescription element.

2 Please note that it is planned to adjust the schema definition in the future in order to remove the deprecated ProviderInfo -element.

	command is requested (e.g. with OID 0.4.0.127.0.7.3.1.4.1 (id-auxiliaryData-1) for age verification, with 0.4.0.127.0.7.3.1.4.2 (id-auxiliaryData-2) for the document validity check or with 0.4.0.127.0.7.3.1.4.3 (id-auxiliaryData-3) to check municipality citizenship, cf. also [TR-03110], Section B.11.8.
TransactionInfo	This element MAY contain transaction-related information, which MUST be displayed in the eID-PIN dialogue before the PACE-protocol is performed.

A DIDAuthenticateResponse element with AuthenticationProtocolData of type EAC1OutputType is returned in response to this request:



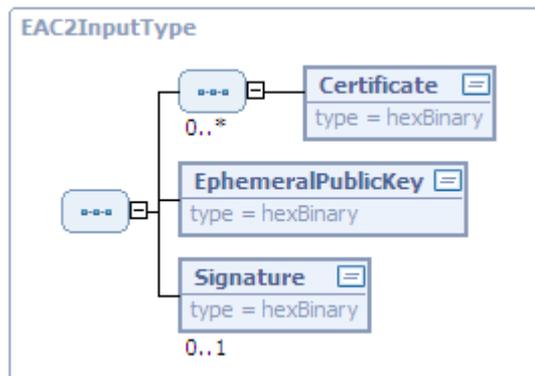
This type specifies the structure of the DIDAuthenticationDataType for the PACE protocol when DIDAuthenticate is returned.

Name	Description
RetryCounter	If the user verification process failed, this element contains the current value of the RetryCounter.
CertificateHolder Authorization Template	If the user has imposed further restrictions on the CHAT transmitted by the eService, such that the actual access rights do not correspond with the access rights which might potentially ensue from the certificate, the eService SAL SHOULD be informed of the CHAT restricted by the user in this manner.
Certification AuthorityReference	MUST be present if the client SAL is not able to build a PICC-verifiable certificate chain from the certificates provided by the eService. In that case the element c contains up to two references to

	the certification authority, which are provided by the chip (see [TR-03110], B.1.4). If two references are returned, the first reference is the more current of the two.
EFCardAccess	If no errors have occurred, the ASN.1-coded SecurityInfos from the EF.CardAccess file (cf. [TR-03110], Table A.1) are returned at this point.
IDPICC	Contains the "card identity" ID_{PICC} . As stipulated in [TR-03110], Section 4.4, this involves the Doc# from the MRZ in case of BAC or the compressed ephemeral public key of the PICC in case of PACE.
Challenge	MAY contain the random number generated by the PICC, $r_{PICC,TA}$, which is signed by the eService SAL during the Terminal Authentication, if and only if the PICC allows the verification of certificates between requesting the Challenge and checking the signature.

3.6.6 Phase 2 - Combination of Terminal and Chip Authentication

The next step involves — if the Challenge was produced in the first phase — a combination of Terminal and Chip Authentication, which is initiated by an invocation of DIDAuthenticate with AuthenticationProtocolData of type EAC2InputType.

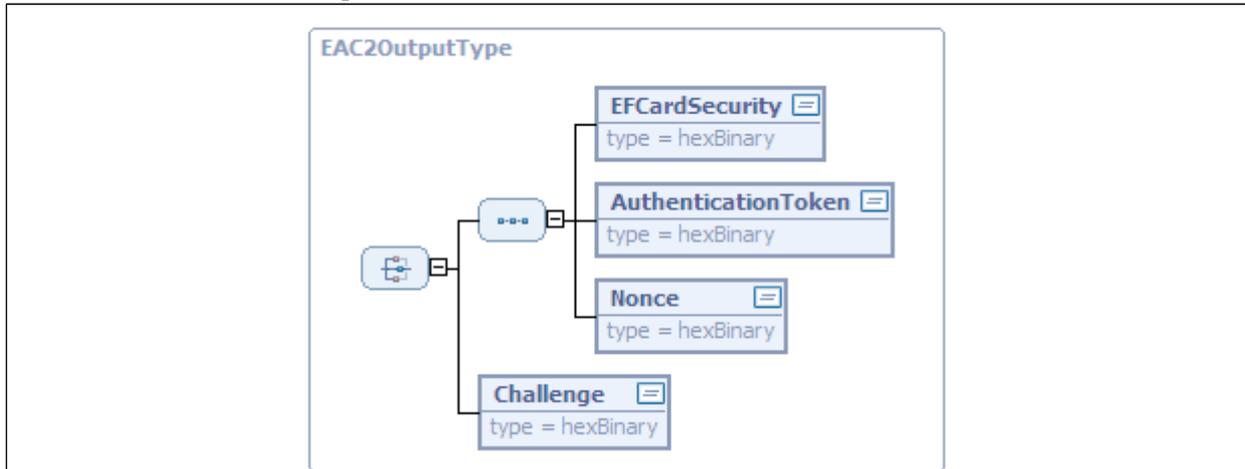


This type specifies the structure of the EAC2InputType which is used in the EAC protocol on the second request of DIDAuthenticate.

Name	Description
Certificate	<p>The Certificate element MAY occur any number of times and contains a certificate in each case so that, together with the eService certificate already transmitted, the resulting overall chain is one which is verifiable by the PICC.</p> <p>This element MUST NOT contain an eService certificate.</p> <p>This element MUST be provided if the element CertificationAuthorityReference in EAC1OutputType is present. If the element CertificationAuthorityReference is not present, the client SAL SHALL build a chain from the certificates transmitted in Phase 1 which is verifiable by the PICC.</p>

EphemeralPublicKey	Contains the public key of the key pair newly generated by the eService.
Signature	Contains, where applicable, the signature generated by the eService SAL during Terminal Authentication.

AuthenticationProtocolData of type EAC2OutputType are returned in the subsequent DIDAuthenticateResponse.

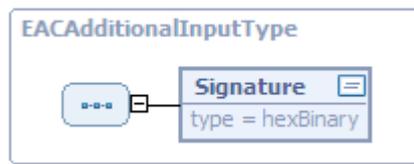


This type specifies the structure of the EAC2OutputType which is used in the EAC protocol on the second request of DIDAuthenticate. If the Challenge has already been returned in the previous message (cf. EAC1OutputType), the elements EFCardSecurity, AuthenticationToken and Nonce are returned. Otherwise the Challenge element is returned at this point.

Name	Description
EFCardSecurity	Contains a SignedData structure in accordance with [RFC3852] which contains the full SecurityInfo structure in the content data (EncapsulatedContentInfo). This signature is checked by the eService SAL during Passive Authentication.
AuthenticationToken	Contains the authentication token (T_{PICC}).
Nonce	Contains the random number ($r_{PICC,CA}$).
Challenge	Contains the Challenge from the PICC. If this element is returned at this point, the additional message MUST be sent with the Signature element in the next step.

3.6.7 Phase 3 - Optional additional message with signature forwarded separately

If the Challenge was produced in Phase 2 instead of Phase 1, an additional invocation of DIDAuthenticate with AuthenticationProtocolData of type EACAdditionalInputType occurs in this optional phase.



This type specifies the structure of the `EACAdditionalInputType` which is used in the optional additional message that is required if the `Challenge` was not included in the first phase.

Name	Description
Signature	Contains the signature generated by the eService SAL during Terminal Authentication.

`AuthenticationProtocolData` of type `EAC2OutputType` (cf. Section 3.6.6) are returned in the subsequent `DIDAuthenticateResponse` and in this case the elements `EFCardSecurity`, `AuthenticationToken` and `Nonce` MUST be included.

3.6.8 Phase 4 - Secure messaging with APDU batches

If the Passive Authentication process (verification of the signature from `EFCardSecurity`) and the verification of the authentication token generated during the Chip Authentication process are successful, the eService SAL MAY now communicate with the PICC with APDUs protected with secure messaging. The APDUs required for this MAY be calculated in advance by the eService SAL and transferred as a batch using the `Transmit` function from [TR-03112-6] via the network to the IFD-Layer on the PICC. The IFD-Layer on the PICC side sends the APDUs prepared by the eService SAL to the PICC in sequence and logs the respective response APDUs which are ultimately sent back to the eService SAL as a collective batch in `TransmitResponse`.

3.6.9 DIDCreate, DIDUpdate and DIDGet

The requests of `DIDCreate`, `DIDUpdate` and `DIDGet` each use an element of the `EACMarkerType` as the input parameter (with `DIDCreate` and `DIDUpdate`) or the output parameter (with `DIDGet`), respectively.

3.6.10 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when invoked:

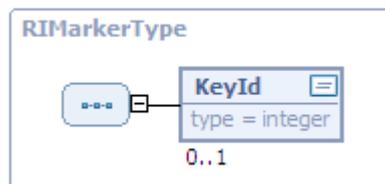
- `DIDAuthenticate`
- `Encipher`
- `Decipher`
- `GetRandom`
- `Hash`
- `Sign`
- `VerifySignature`

- VerifyCertificate

3.7 Restricted Identification

The Restricted Identification protocol is used to generate a sector-specific pseudonym and is defined in [TR-03110].

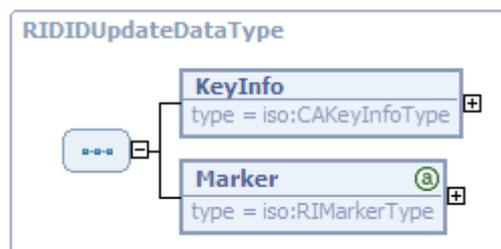
3.7.1 Marker



This type specifies the structure of the DID marker for the Restricted Identification protocol.

Name	Description
KeyId	MAY contain the local key identifier, if the PICC provides multiple keys for Restricted Identification.

3.7.2 DIDCreate



This type specifies the structure of the DIDUpdateDataType for the Restricted Identification protocol.

Name	Description
KeyInfo	Contains information on the key material for the DID. Details on the CAKeyInfoType are given on page 38.
Marker	Contains additional information on the DID. Details on the RIMarkerType are given in Section 3.7.1.

3.7.3 DIDUpdate

DIDUpdate uses the RIDIDUpdateDataType defined above.

3.7.4 DIDGet

DIDGet returns a DIDStructure, which contains a marker of type RIMarkerType.

3.7.5 DIDAuthenticate

For cards, which – *unlike the German eID card* – do not require secure messaging for performing the Restricted Identification protocol, this protocol would be implemented by taking the following steps:

- The sector-specific public key of the terminal for Restricted Identification including the domain parameters is transmitted to the card in a DIDAuthenticate request, whereby the AuthenticationProtocolData is of type RIDIDAuthInputType.
- The card calculates the sector-specific pseudonym and returns it in DIDAuthenticateResponse, which contains AuthenticationProtocolData of type RIDIDAuthOutputType.



The type specifies the structure of the RIDIDAuthInputType used in step 1.

Name	Description
SectorPublicKey	Contains the sector-specific public key including the domain parameters, as specified in [TR-03110], Annex D.3.



The type specifies the structure of the RIDIDAuthOutputType used in step 2.

Name	Description
SectorSpecificIdentifier	Contains the sector-specific pseudonym.

3.7.6 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when called:

- CardApplicationStartSession
- Encipher
- Decipher
- GetRandom

-
- Hash
 - Sign
 - VerifySignature

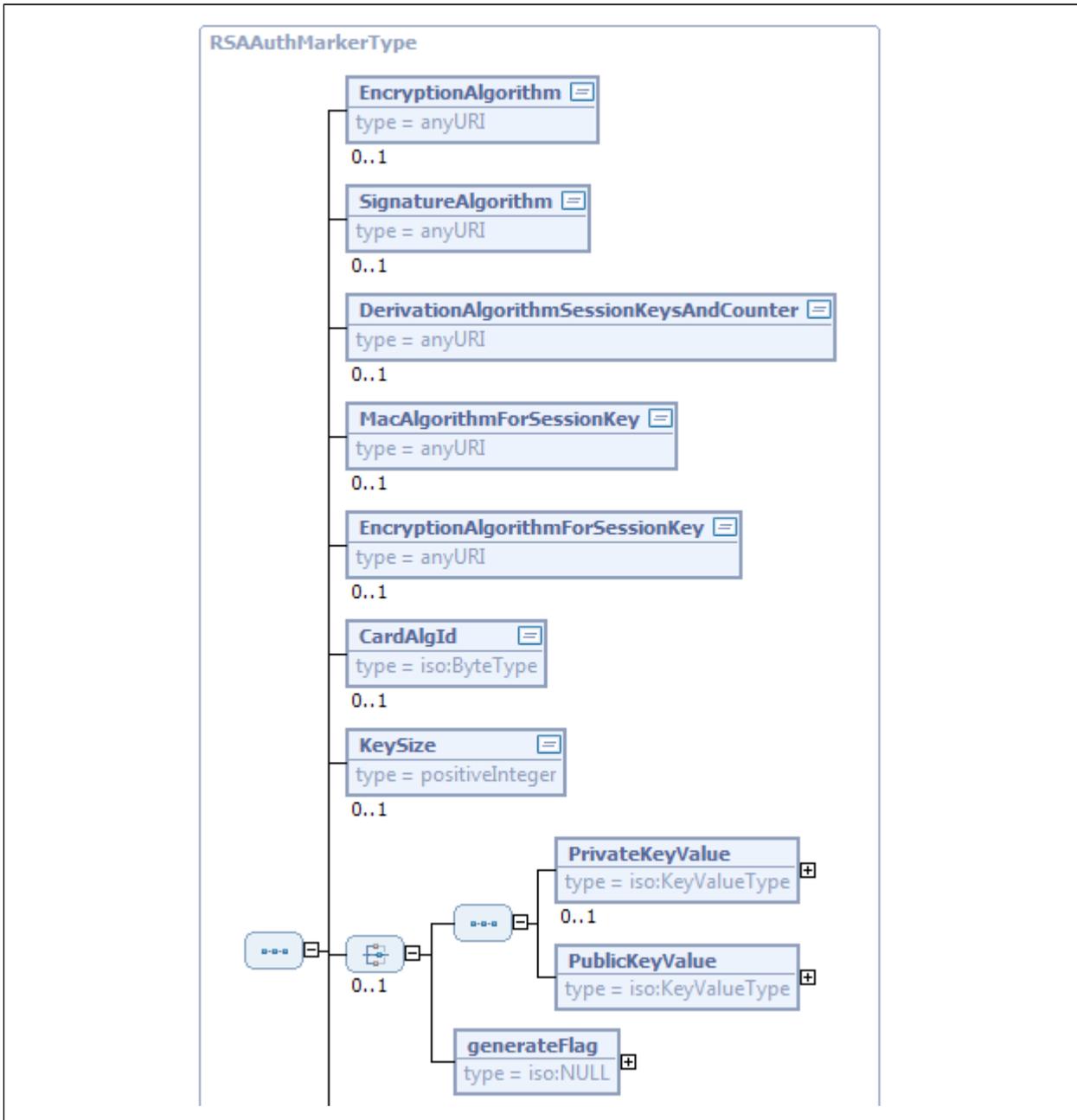
3.8 RSA Authentication

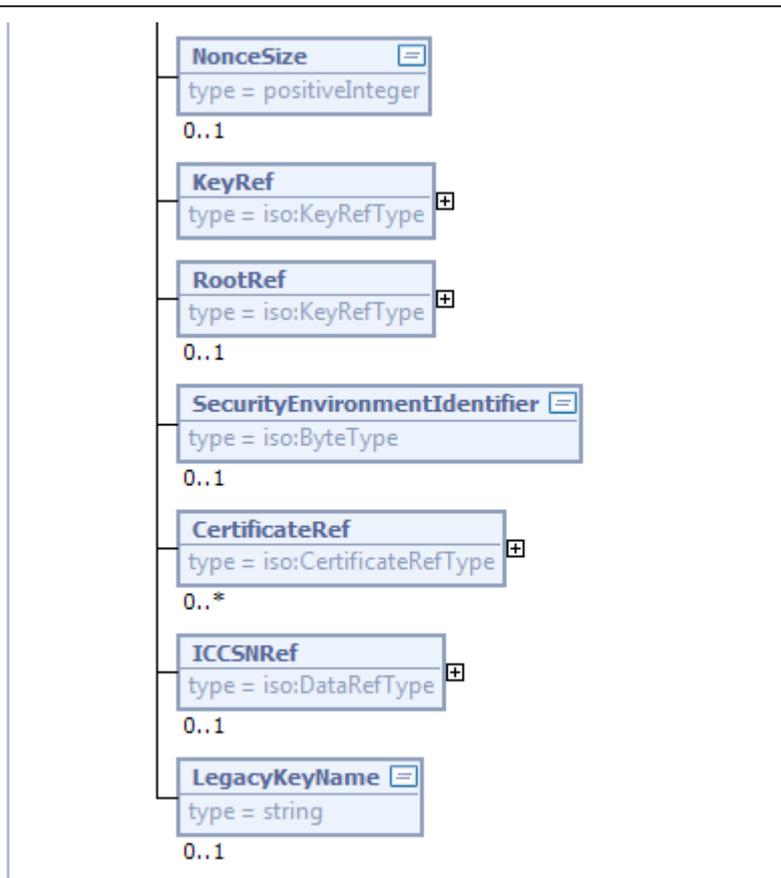
This protocol is specified in a similar form in Annex A.15 of [ISO24727-3], Section 16 of [eGK-1] and Section 8.4 of [EN14890-1] and provides the framework for mutual authentication with an optional exchange of keys using the RSA algorithm.

The identifier for this protocol is [urn:oid:1.3.162.15480.3.0.15](#) for iso(1) identified-organization (3) CEN (162) CEN 15480 (15480) part3(3) annex-a(0) key-transport-with-mutual-authentication(15).

The generic structures from [TR-03112-4] section 3.5 are complemented by the specification of the following types.

3.8.1 Marker

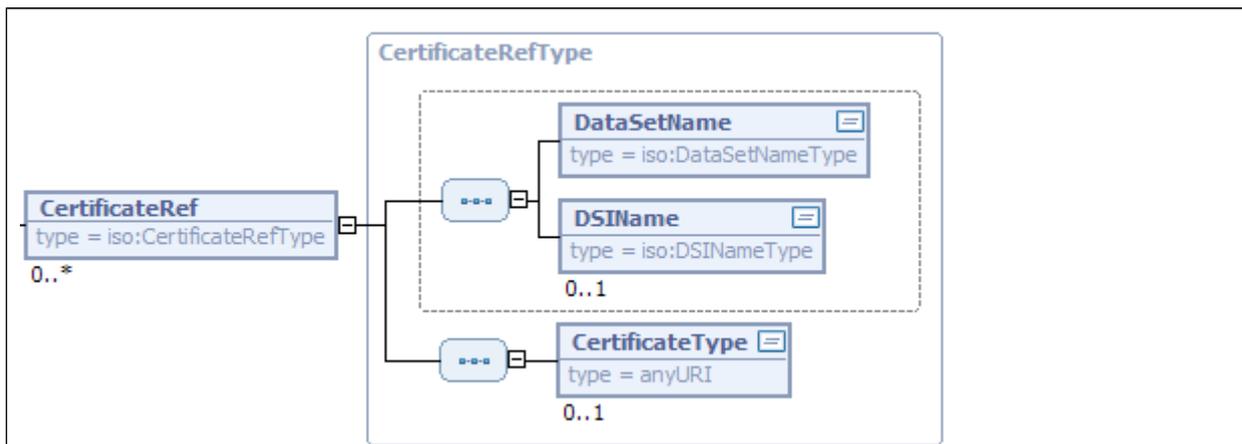




This type specifies the structure of the DID marker for this authentication protocol.

Name	Description
EncryptionAlgorithm	MAY contain a URI for the encryption algorithm to be used in the scope of the optional key exchange. Alternatively it is possible to specify the CardAlgId below, which implies which encryption algorithm is to be used by the card.
SignatureAlgorithm	Specifies the signature algorithm to be used for mutual authentication. Alternatively it is possible to specify the CardAlgId below, which implies which signature algorithm is to be used by the card.
DerivationAlgorithm SessionKeysAndCounter	MAY specify the algorithm required to derive the session key and counters. Alternatively it is possible to specify the CardAlgId below, which implies which derivation algorithm is to be used by the card.
MacAlgorithmForSessionKey	MAY specify the MAC algorithm to be used for secure messaging. Alternatively it is possible to specify the CardAlgId below, which implies which MAC algorithm is to be used for secure messaging by the card.
EncryptionAlgorithm ForSessionKey	MAY specify the encryption algorithm to be used for secure messaging. Alternatively it is possible to specify the CardAlgId below, which implies which encryption algorithm is to be used for secure messaging by the card.

CardAlgId	<p>MAY contain a card-specific algorithm identifier, which implies what kind of algorithms are to be used by the card for asymmetric encryption, signature generation, session key derivation, MAC-calculation and message encryption. Please refer to [eGK-1], Table 168 for the possible algorithm identifiers supported by the German eHealth-card.</p> <p>If this parameter is present, the other algorithm-parameters SHOULD be omitted.</p>
KeySize	MAY contain the bit length of the RSA modulus.
PrivateKeyValue	MAY contain the private key. The structure of the KeyValueType is explained on page 38.
PublicKeyValue	MAY contain the public key of the DID, if the generateFlag alternative has not been selected. The structure of the KeyValueType is explained on page 38.
generateFlag	MAY specify that the key pair belonging to the DID is to be generated on the card.
NonceSize	MAY contain the byte length of the challenges to be used in this authentication protocol.
KeyRef	MUST contain the reference of the private key of the DID.
RootRef	MAY contain the reference of the trustworthy root key of the DID. If this element is omitted, the reference to the root key MUST be provided by the certificate requiring verification.
SecurityEnvironmentIdentifier	Is an OPTIONAL element by means of which a security environment which deviates from the standard environment MAY be specified.
CertificateRef	MAY contain a sequence of references to certificates which are stored on the card. If these are card-verifiable certificates which are to be used to verify the certificate path with the VERIFY CERTIFICATE command from [ISO7816-8] (Section 11.11), they SHOULD be specified in the sequence required to verify the certificate path, i.e. starting from a trustworthy root. Further details on the CertificateRefType MAY be found below.
ICCSNRef	MAY contain a reference to the serial number of the card. The DataRefType is explained on page 28. If this reference is known otherwise, e.g. by a CardInfo file (cf. [TR-03112-4], Annex A), the element MAY be omitted here.
LegacyKeyName	MAY contain a name which can be used to address the DID in a legacy application, e.g. a Microsoft Cryptographic API provider.



CertificateRef is part of the RSAAuthQualifierType and contains a reference to a certificate stored on the card. This type extends the DataRefType by adding the CertificateType element.

Name	Description
CertificateType	<p>MAY specify the type of certificate. The following certificate types are defined here:</p> <ul style="list-style-type: none"> • urn:ietf:rfc:3280 - for X.509 public key certificates in accordance with [RFC3280] • urn:ietf:rfc:3281 - for X.509-based attribute certificates in accordance with [RFC3281] • urn:iso:std:iso-iec:7816:-8:tech:certificate: <aid>: <cpi> for a card-variable certificate, whereby <aid> contains the application identifier of a registered card application and <cpi> contains a corresponding certificate profile identifier (tag '5F29').

3.8.2 DIDCreate

With DIDCreate use is made of DIDCreationData of the RSAAuthMarkerType.

3.8.3 DIDUpdate

With DIDUpdate use is made of DIDUpdateData of the RSAAuthMarkerType.

3.8.4 DIDGet

With DIDGet use is made of DIDDiscoveryData of the RSAAuthMarkerType.

3.8.5 CardApplicationStartSession

The optional parameters on request and return of CardApplicationStartSession are explained in Section 3.2.5.

The procedure for setting up a session is approximately defined as follows (refer to [eGK-1] for details):

1. Determine DID information for ICC by means of `DIDGet` and read out corresponding certificates using `DataSetSelect` and `DSIRead`.
2. Invoke `DIDAuthenticate` to verify certificate path on SAM by successive invocations of `VerifyCertificate` with the certificates determined in step 1.
3. Determine DID information for SAM by means of `DIDGet` and read out corresponding certificates using `DataSetSelect` and `DSIRead`.
4. Invoke `DIDAuthenticate` to verify certificate path on ICC by successive invocations of `VerifyCertificate` with the certificates determined in step 3.
5. Request a Challenge from SAM and invoke `DIDAuthenticate` for INTERNAL AUTHENTICATE on ICC.
6. Invoke `DIDAuthenticate` for EXTERNAL AUTHENTICATE on SAM with result from step 5.
7. Request a Challenge from ICC and invoke `DIDAuthenticate` for INTERNAL AUTHENTICATE on SAM.
8. Invoke `DIDAuthenticate` for EXTERNAL AUTHENTICATE on ICC with result from step 7.

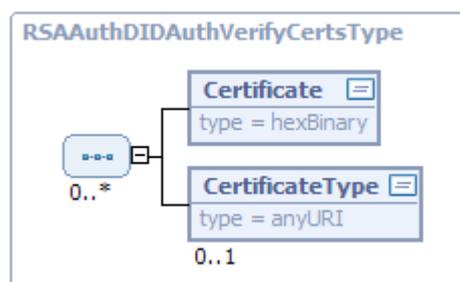
If the ICC under considerations requires that the establishment of a secure session is to be logged in a certain file on the card, this task MUST be performed *within* the call of the `CardApplicationStartSession`-function. Note that this requirement especially exists for the German eHealth-card (refer to [eGK-1] and [eGK-2]) and hence the establishment of a secure session with `CardApplicationStartSession` MUST be logged in the elementary file `EF.Logging` (refer to Section 6.3.4 of [eGK-2]).

3.8.6 DIDAuthenticate

`DIDAuthenticate` is used in this protocol for the following purposes:

- To verify the certificate path
- To invoke INTERNAL AUTHENTICATE
- To invoke EXTERNAL AUTHENTICATE

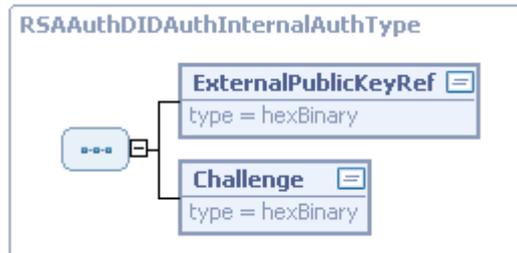
3.8.7 Verification of the certificate path



This type specifies the structure of the `DIDAuthenticationDataType` for the RSA Key Transport protocol on invocation of `DIDAuthenticate` for verification of the certificate path and contains a sequence of certificates which are used in the specified sequence to verify the certificate path — starting from a common root.

Name	Description
Certificate	Contains a certificate.
CertificateType	MAY specify which type of certificate is involved (cf. VerifyCertificate in [TR-03112-4]).

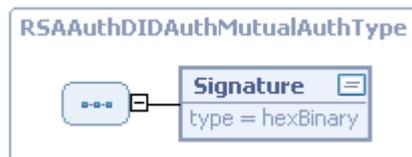
3.8.8 Invocation of INTERNAL AUTHENTICATE



This type specifies the structure of the DIDAuthenticationDataType for the RSA Authentication protocol when DIDAuthenticate is invoked to invoke INTERNAL AUTHENTICATE.

Name	Description
ExternalPublicKeyRef	Contains the key reference of the public key of the message recipient. In the case of the electronic health insurance card (cf. [eGK-2]), for example, this key reference consists of two zero bytes and the counterpart ICCSN which is stored in EF.GDO.
Challenge	Contains the challenge of the communication partner which is to be signed by invoking INTERNAL AUTHENTICATE.

In this case the return to DIDAuthenticateResponse is as follows:



This type specifies the structure of the DIDAuthenticationDataType for the RSA Authentication protocol when DIDAuthenticate is returned after INTERNAL AUTHENTICATE.

Name	Description
Signature	Contains the signature generated during this request.

3.8.9 Invocation of EXTERNAL AUTHENTICATE



This type specifies the structure of the `DIDAuthenticationDataType` for the RSA Authentication protocol when `DIDAuthenticate` is invoked to invoke EXTERNAL AUTHENTICATE.

Name	Description
<code>MutualCryptogram</code>	Contains the signature to be verified by means of EXTERNAL AUTHENTICATE.

3.8.10 VerifyCertificate

The certificate which has been transmitted is checked against the public key of the trustworthy root referenced in `RootCert`.

3.8.11 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when called:

- Encipher
- Decipher
- Hash
- Sign
- VerifySignature

3.8.12 Minimum requirements in terms of algorithms

This authentication protocol MAY basically be used with various cryptographic algorithms but the following algorithms MUST be supported as a minimum in accordance with [eGK-1]:

- **EncryptionAlgorithm**
[urn:oid:1.2.840.113549.1.1.1](#) for **RSA encryption** ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) rsa-encryption(1)}
- **SignatureAlgorithm**
[urn:oid:1.3.36.3.4.3.2.1](#) for **sigS-ISO9796-2rndWithsha1** ::= {iso(1) identified-organization(3) teletrust(36) algorithm(3) signatureScheme(4) sigS-ISO9796-2rnd(3) sigS-ISO9796-2rndWithrsa(2) sigS-ISO9796-2rndWithsha1(1)}
[urn:oid:1.3.36.3.4.3.2.4](#) for **sigS-ISO9796-2rndWithsha256** ::= {iso(1) identified-organization(3) teletrust(36) algorithm(3) signatureScheme(4) sigS-ISO9796-2rnd(3) sigS-ISO9796-2rndWithrsa(2) sigS-ISO9796-2rndWithsha256(4)}

-
- **DerivationAlgorithmSessionKeysAndCounter**
[urn:oid:1.3.162.14890.1.1.1³](#) for **CEN14890-KDF-Simple** ::= { iso(1) identified-organization (3) CEN (162) CEN 14890 (15480) part-1 (1) key-derivation (1) simple-scheme (1)} in accordance with Section 8.4.2 from [EN14890-1]
 - **MacAlgorithmForSessionKey**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7)}
 - **EncryptionAlgorithmForSessionKey**
[urn:oid:1.2.840.113549.3.7](#) for **des-EDE3-CBC** ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) des-ede3-cbc(7)}

3.9 Generic cryptography

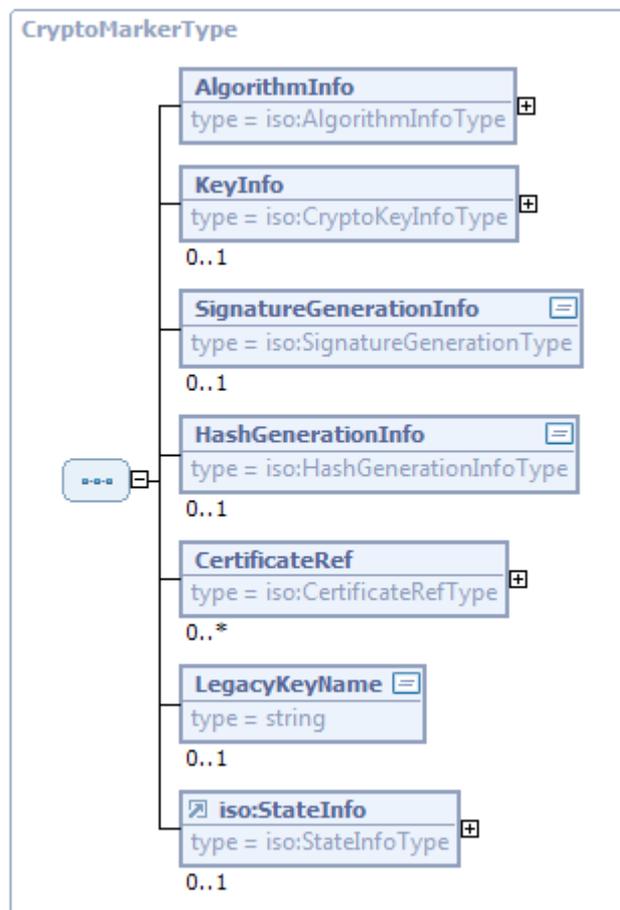
Cryptographic operations can be used independently from specific authentication procedures under this generic protocol.

The identifier for this protocol is [urn:oid:1.3.162.15480.3.0.25](#) for iso(1) identified-organization (3) CEN (162) CEN 15480 (15480) part3(3 annex-a(0) generic-cryptography (25).

The generic structures from [TR-03112-4] are complemented by the definition of the following types.

³ Note that this URI is not yet formally assigned.

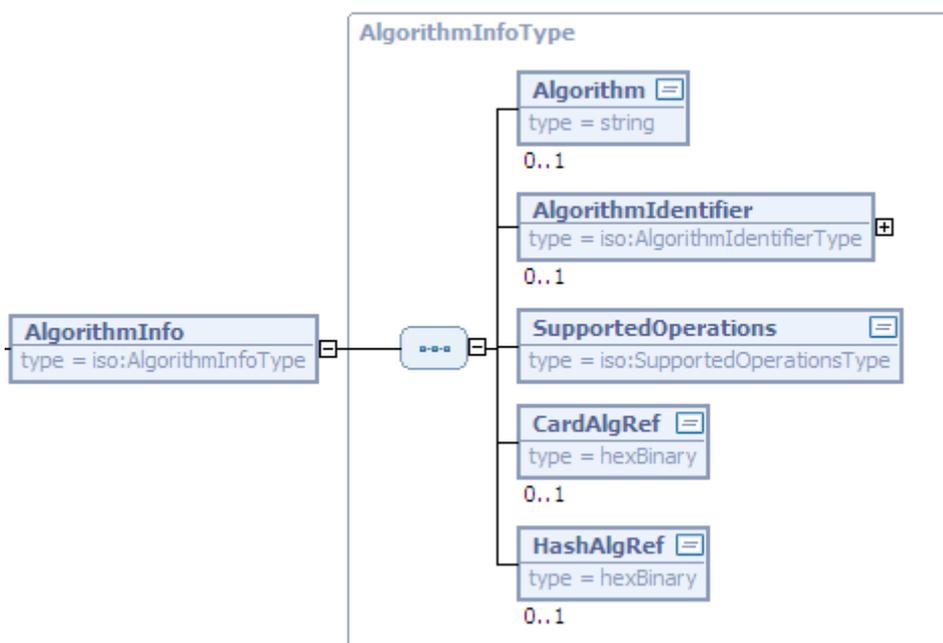
3.9.1 Marker



This type specifies the structure of the DID marker for this generic protocol.

Name	Description
AlgorithmInfo	Contains information about the cryptographic algorithms supported by the DID. See below for details.
KeyInfo	MAY contain information on the DID key material. See below for details.
SignatureGenerationInfo	MAY contain information about the sequence of the smart card commands required for the DID to generate signatures, if the DID corresponds to a signature key. The SignatureGenerationType is defined as follows: <pre> <simpleType name = "SignatureGenerationType"> <list> <simpleType> <restriction base = "token"> <enumeration value = "MSE_RESTORE" /> <enumeration value = "MSE_HASH" /> <enumeration value = "PSO_HASH" /> <enumeration value = "MSE_KEY" /> <enumeration value = "MSE_DS" /> </restriction> </simpleType> </list> </simpleType> </pre>

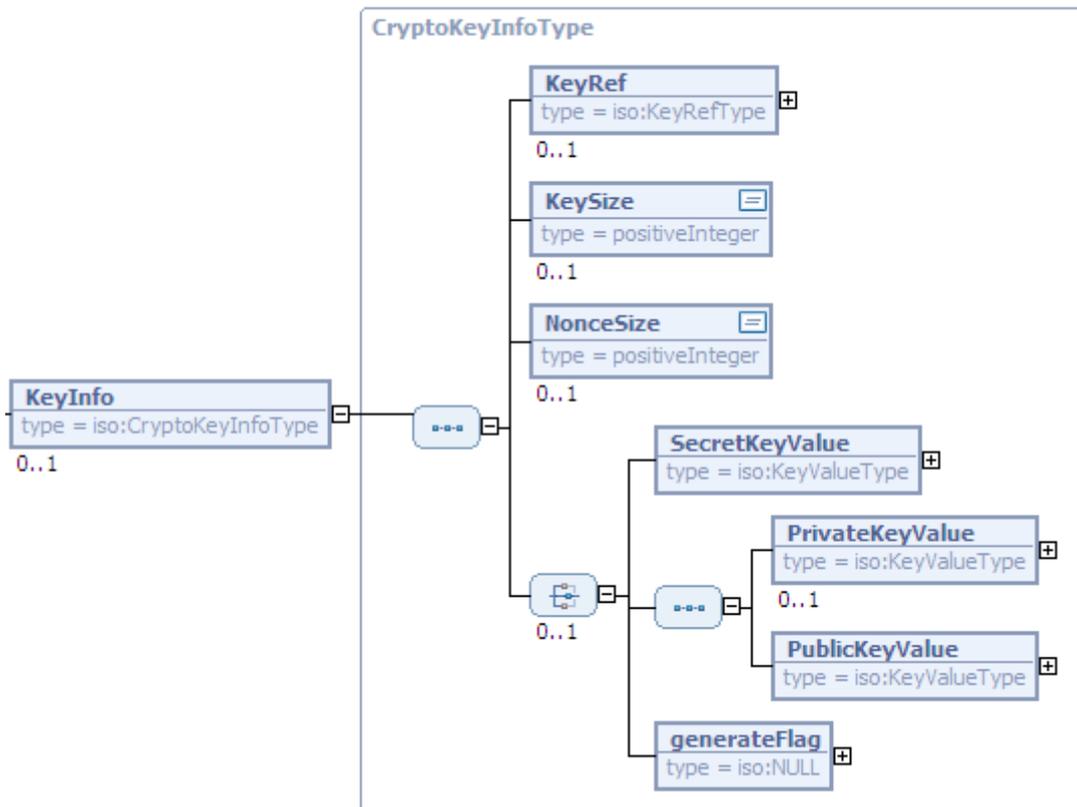
	<pre data-bbox="619 255 1382 472"> <enumeration value = "MSE_KEY_DS" /> <enumeration value = "PSO_CDS" /> <enumeration value = "INT_AUTH" /> </ restriction> </ simpleType> </ list> </ simpleType> </pre> <p data-bbox="608 501 1342 595">The definition of this type allows to specify an ordered list of smart card commands, which are necessary to generate a signature with the specific DID.</p>
HashGenerationInfo	<p data-bbox="608 613 1374 745">MAY contain information about the details required for the DID to calculate hash values, if the DID is meant to calculate hash values. The HashGenerationInfoType is defined as follows:</p> <pre data-bbox="619 763 1382 981"> <simpleType name = "HashGenerationInfoType"> <restriction base = "string"> <enumeration value = "NotOnCard" /> <enumeration value = "CompletelyOnCard" /> <enumeration value = "LastRoundOnCard" /> </ restriction> </ simpleType> </pre>
CertificateRef	<p data-bbox="608 1010 1362 1106">MAY contain a sequence of references to certificates which are stored on the card. Further details about the CertificateRefType are provided on page 65.</p>
LegacyKeyName	<p data-bbox="608 1128 1374 1189">MAY contain a name which can be used to address the DID in a legacy application, e.g. a Microsoft Cryptographic API provider.</p>
iso: StateInfo	<p data-bbox="608 1211 1374 1272">MAY contain information about the designated states of the key object, if more than one state is defined.</p> <p data-bbox="608 1290 1382 1350">More details about the iso: StateInfo -element are provided on page 17.</p>



The AlgorithmInfo element is part of the CryptoMarkerType and contains information about the cryptographic algorithm supported by the DID. The AlgorithmInfoType is based on the AlgorithmInfo structure from [ISO7816-15].

Name	Description
Algorithm	MAY contain a textual descriptor for the algorithm.
AlgorithmIdentifier	MAY contain the unambiguous descriptor for the cryptographic algorithm in the form of a URI and, if required, further parameters for the algorithm. Further information on the AlgorithmIdentifierType is given on page 40.
SupportedOperations	Specifies the cryptographic operations for which the DID MAY be used. The SupportedOperationsType is defined as follows (cf. [ISO7816-15]): <pre> <simpleType name = "SupportedOperationsType"> <union memberTypes = "iso:BitString"> <simpleType> <list> <simpleType> <restriction base = "token"> <enumeration value = "Compute-checksum" /> <enumeration value = "Compute-signature" /> <enumeration value = "Verify-checksum" /> <enumeration value = "Verify-signature" /> <enumeration value = "Encipher" /> <enumeration value = "Decipher" /> <enumeration value = "Hash" /> <enumeration value = "Derive-key" /> </restriction> </simpleType> </list> </simpleType> </union> </simpleType> </pre>

	<pre></ union> </ simpleType></pre>
CardAlgRef	MAY contain the card-specific cryptographic mechanism reference according to [ISO7816-4] (Table 33) and hence the content of the CardAlgRef -element MUST be used in an MSE-command with Tag '80'.
HashAlgRef	MAY contain the card-specific reference for a hash algorithm, if the present AlgorithmInfo -element refers to a signature algorithm and CardAlgRef does not implicitly specify the hash algorithm, which is to be used for the signature generation.



The KeyInfo element is part of the CryptoMarkerType and contains information about the DID key material.

Name	Description
KeyRef	MAY contain the key reference of the DID on the card. Details on the KeyRefType are given on page 16.
KeySize	MAY contain the relevant bit length of the cryptographic key.
NonceSize	MAY contain the length of the random numbers in bytes which can be requested via GetRandom.
SecretKeyValue	Contains the value of a secret key. The structure of the KeyValueTypes is explained on page 38.
PrivateKeyValue	MAY contain the private key. The structure of the KeyValueTypes

	is explained on page 38.
PublicKeyValue	Contains the public key of the DID if the generateFlag alternative has not been selected. The structure of the KeyValue type is explained on page 38.
generateFlag	Specifies that the key material belonging to the DID is to be generated on the card. If this alternative is not selected, either the SecretKeyValue element is present or at least the PublicKeyValue element and, where applicable, (if the DID is to be used for signature generation or decryption) the PrivateKeyValue element as well.

3.9.2 DIDCreate

DIDCreate uses DIDCreationData of type CryptoMarkerType.

3.9.3 DIDUpdate

DIDUpdate uses DIDUpdateData of type CryptoMarkerType.

3.9.4 DIDGet

DIDGet uses DIDDiscoveryData of type CryptoMarkerType.

3.9.5 Encipher

If this operation is supported by the DID (cf. SupportedOperations element, page 72) the plain text is encrypted with the public or secret key assigned to the DID.

3.9.6 Decipher

If this operation is supported by the DID (cf. SupportedOperations element, page 72) the cipher text is decrypted with the private or secret key assigned to the DID.

3.9.7 GetRandom

Generates a random number of the required length (NonceSize bytes) and returns it:

```
random = RNG (NonceSize)
```

3.9.8 Hash

If this operation is supported by the DID (cf. SupportedOperations element, page 72) a hash value is generated from the message using the relevant algorithm. For this purpose the calculation is either performed entirely on the card, partly on the card or not on the card (cf. HashGenerationInfo on page 71).

Note that hash values for data streams MAY be calculated using DIDAuthenticate (cf. Section 3.9.12).

3.9.9 Sign

If this operation is supported by the DID (cf. SupportedOperations element, page 72) a signature is generated for the provided message using the specified DID. For this purpose the series of smart card commands specified in the SignatureGenerationInfo element (see page 70) is transmitted to the smart card.

3.9.10 VerifySignature

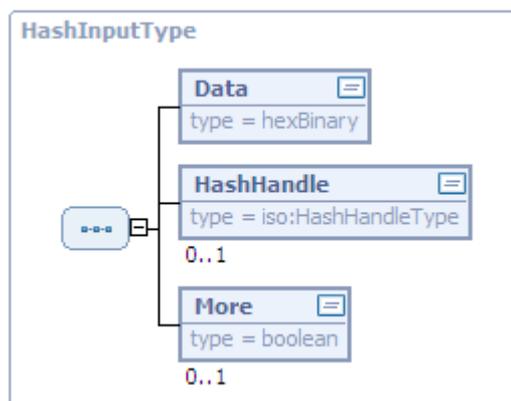
The provided signature is verified using the public key referenced with DIDName.

3.9.11 VerifyCertificate

The certificate which has been transmitted is checked against the public key of the trustworthy root key referenced in RootCert.

3.9.12 DIDAuthenticate

The DIDAuthenticate function of the Generic Crypto protocol MAY be used in an iterative manner to calculate hash values of data streams or large volumes of data.

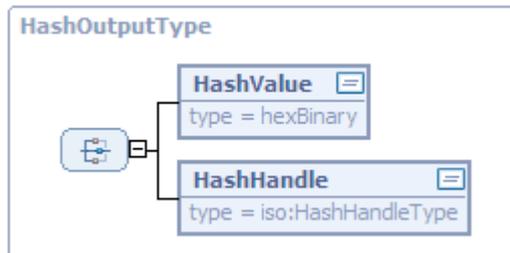


This type specifies the structure of the HashInputTypes which is used when invoking DIDAuthenticate.

Name	Description
Data	Contains the data which are used for the calculation of the hash value.

HashHandle	<p>Is an optional element, which is used to be able to link several requests of DIDAuthenticate. This might be needed, for example, to calculate hash values for large volumes of data or data streams with several requests from DIDAuthenticate.</p> <p>It is necessary to differentiate between the following four cases:</p> <ul style="list-style-type: none"> • No HashHandle and More=FALSE (or not available) – the HashValue calculated from the data is the only output parameter. • No HashHandle and More=TRUE – a new HashHandle is returned. • HashHandle exists and More=FALSE (or not available) – the data transmitted are also included in the calculation of the hash value and the completely calculated HashValue is returned. • HashHandle exists and More=TRUE – the data transmitted are also included in the calculation of the hash value and the transmitted HashHandle is returned.
More	<p>More=TRUE specifies that the hash value is not yet to be returned, because further calls of DIDAuthenticate are expected and therefore a HashHandle is returned.</p> <p>If this element is missing, this is equivalent to the default value More=FALSE and, as a result, any existing HashHandle will lose its validity and a HashValue will be returned.</p>

The DIDAuthenticateResponse returns data of the HashOutputType in the AuthenticationProtocolData element:



This type specifies the structure of the HashOutputType, which is returned in the AuthenticationProtocolData element of the DIDAuthenticateResponse.

Name	Description
HashValue	Is the calculated hash value.
HashHandle	Is a handle which MAY be transmitted with the next invocation of DIDAuthenticate so that the hash value MAY be calculated iteratively.

3.9.13 Non-supported functions

The following functions are not supported with this protocol and return a corresponding error message [/resultminor/sal#inappropriateProtocolForAction](#) when invoked:

- CardApplicationStartSession

4 Protocols for GetCertificate

4.1 GetCertificate by means of Simple Enrollment Protocol

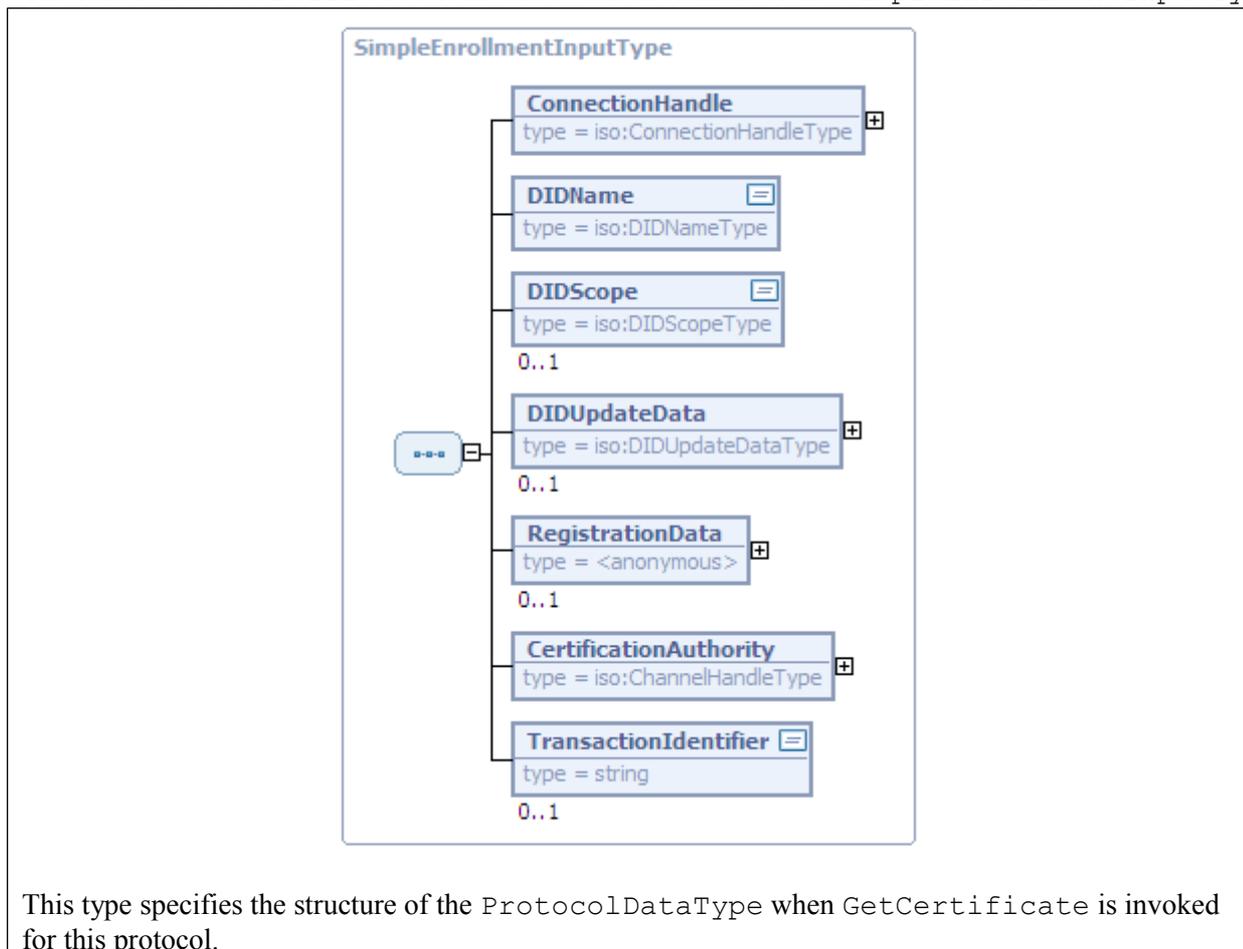
In the scope of this protocol, a key pair is generated on the card and used with other information about the subsequent certificate holder to generate a self-signed certificate request in accordance with [RFC2314] (PKCS #10), which is sent by means of the Simple Enrollment Protocol specified in [ISIS-MTT-2] to a specific URL of the certification authority.

If the process is successful, the expected response from the certification authority, in addition to the status message, is a certificate chain in a CMS container [RFC3369] with MIME type *application/pkcs7-mime*.

This protocol has the following URI:

<http://www.bsi.bund.de/ecard/api/1.0/protocols/GetCertificate#SimpleEnrollmentProtocol>

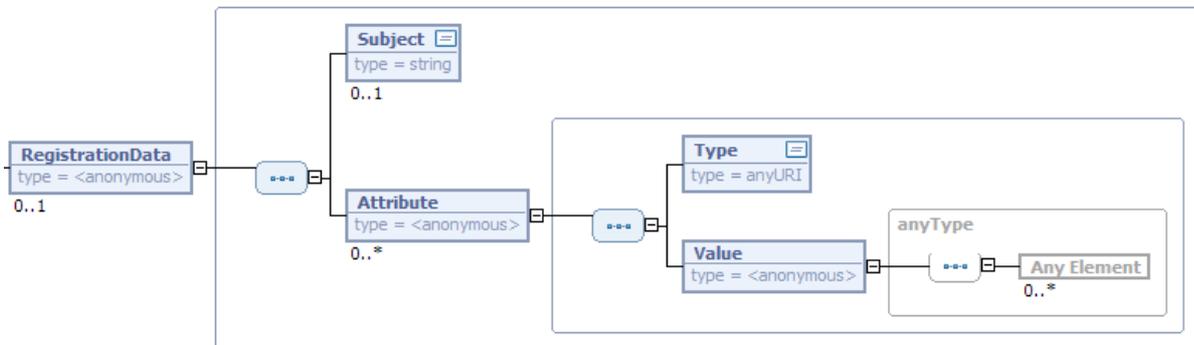
The function `GetCertificate` is invoked with an element of the `SimpleEnrollmentInputType`.



This type specifies the structure of the `ProtocolDataType` when `GetCertificate` is invoked for this protocol.

Name	Description
ConnectionHandle	Contains a handle that is used to address the connection which has

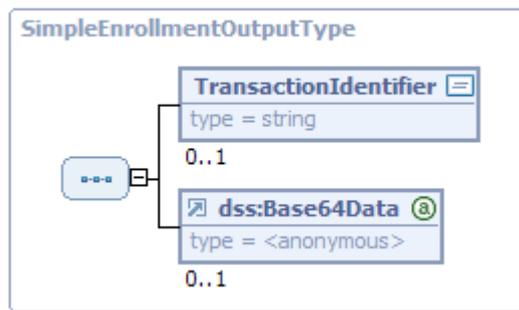
	been established to a card application.
DIDName	In general addresses a certain differential identity (DID) in the card application or the alpha card application specified via the <code>ConnectionHandle</code> . In this case it MUST be a DID for a public key protocol (cf. Section 3.8 and 3.9). Otherwise an error message is returned /resultminor/sal#inappropriateProtocolForAction .
DIDScope	Is an optional parameter which resolves any ambiguity between local and global DIDs with the same name. If the DID is already unambiguously specified by the <code>DIDName</code> , this element MAY be omitted.
DIDUpdateData	MAY contain information which is required for the implicit generation of the corresponding differential identity with <code>DIDCreate</code> . If the only requirement is to create a new certificate for an existing DID then this element is omitted. See <code>DIDCreate</code> in [TR-03112-4] and the relevant detailed specifications in section 3.8.2 and 3.9.2.
RegistrationData	MAY contain additional data which are to be included in the certificate. See below for details.
CertificationAuthority	Contains the address of the certification authority (cf. <code>CardApplicationPath</code> in [TR-03112-4]).
TransactionIdentifier	MAY contain a "ticket", which establishes the logical connection to a previous (unsuccessful) request of <code>GetCertificate</code> .



`RegistrationData` is part of the `SimpleEnrollmentInputType` and contains data which are to be included in the certificate.

Name	Description
Subject	MAY contain the name of the certificate holder. If this element is missing, a corresponding error message or warning /resultminor/il/certificateRequest#subjectMissing MAY be returned and/or a random pseudonym MAY be selected for this purpose.
Attribute	MAY contain any sequence of attributes, which are represented by a pair consisting of <code>Type</code> and <code>Value</code> . The attributes defined in [ISIS-MTT-2] SHOULD be supported.

The `GetCertificate` function returns an element of the `SimpleEnrollmentOutputType`.



This type specifies the structure of the ProtocolDataType when GetCertificate is returned for this protocol.

Name	Description
TransactionIdentifier	If the required certificate can not be returned immediately this element MAY be used in subsequent requests.
dss:Base64Data	In case of success this element contains the certificates generated by the certification authority. These certificates MUST be returned as a CMS Container [RFC3369] with MIME type <i>application/pkcs7-mime</i> . Furthermore the returned certificates SHOULD be stored in the DataSets on the card, if applicable, or the certificate database of the eCard-API-Framework (cf. AddCertificate in [TR-03112-3]).

When GetCertificate is called with an element of the SimpleCertificateEnrollmentType, the following errors and warnings MAY occur in addition to the errors which MAY occur due to the DIDCreate, DIDUpdate, DataSetSelect, DSIWrite and AddCertificate functions:

- [/resultminor/il/certificateRequest#unknownAttribute](#)
- [/resultminor/il/certificateRequest#creationOfCertificateRequestFailed](#)
- [/resultminor/il/certificateRequest#submissionFailed](#)
- [/resultminor/il/certificateRequest#unknownTransactionID](#)
- [/resultminor/il/certificateRequest#certificateDownloadFailed](#)
- [/resultminor/il/certificateRequest#subjectMissing](#)

5 Basic Update Protocol

When `FrameworkUpdate` (cf. [TR-03112-3]) is invoked, the “Basic Update Protocol” is performed with the update server specified in the default configuration. As shown in Figure 4, this protocol MAY use a trusted channel in order to invoke the function `CheckFrameworkUpdate` on the update server, whereby the information described in more detail below is transferred to the modules currently installed (`InstalledModule` elements). In response the update server sends the information set out in greater detail below to the new modules available which are finally returned in response to `FrameworkUpdate` to the client application (`NewModule` elements).

Next the eCard-API-Framework downloads the files required for the update, verifies the authenticity and integrity of the update files and finally runs the update installer. In the last step, the results of the update procedure are returned to the client application.

The input and output parameters of the `CheckFrameworkUpdate` -function are specified on the basis of the `ModuleInfoType` given as follows:

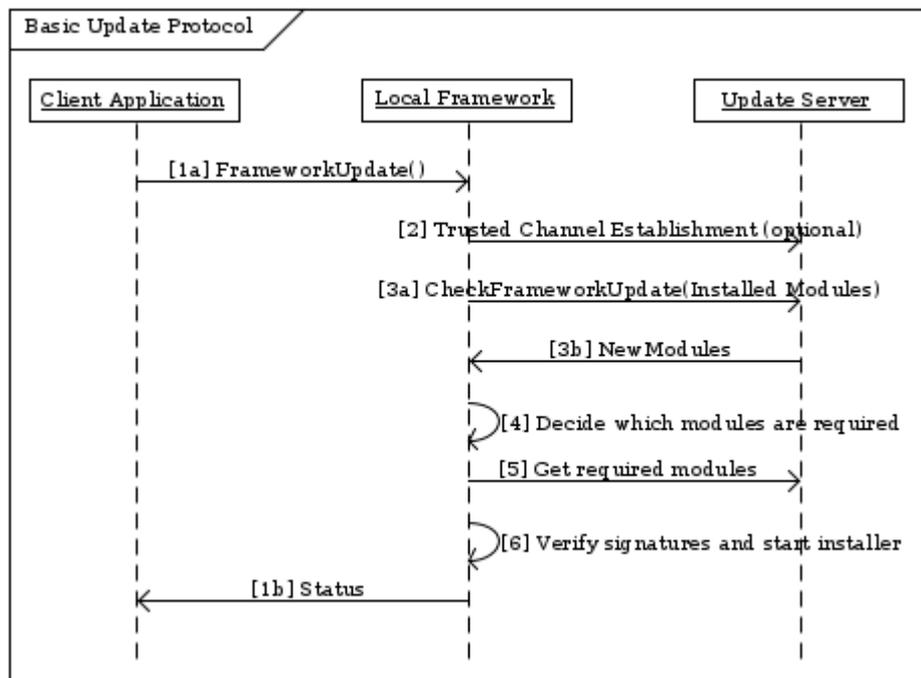
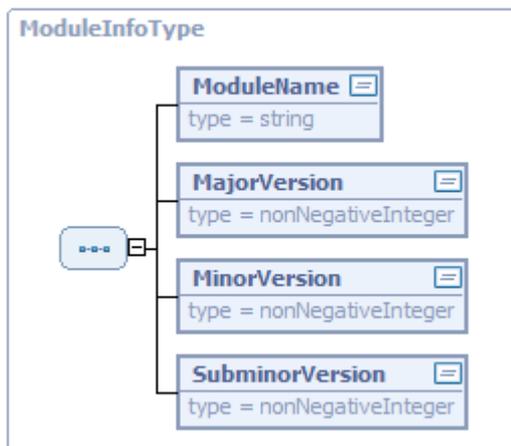


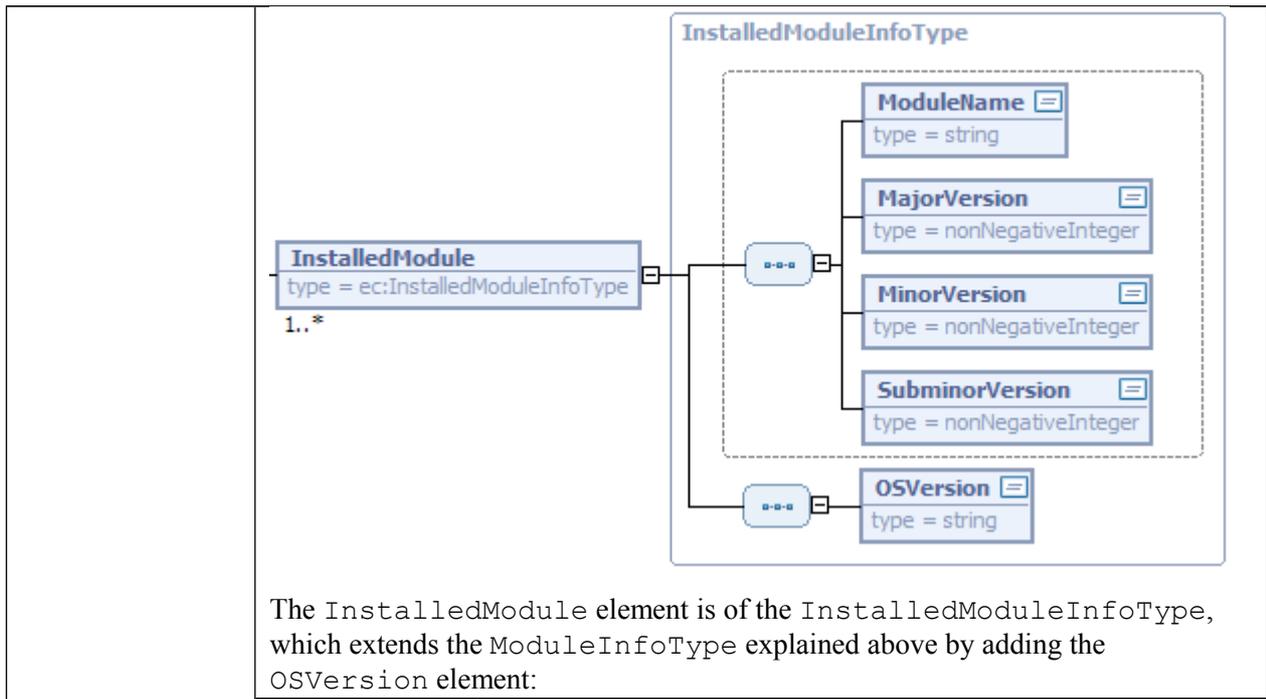
Figure 4: Basic Update Protocol



The ModulInfoType acts as a basic type for the messages exchanged within the Basic Update Protocol.

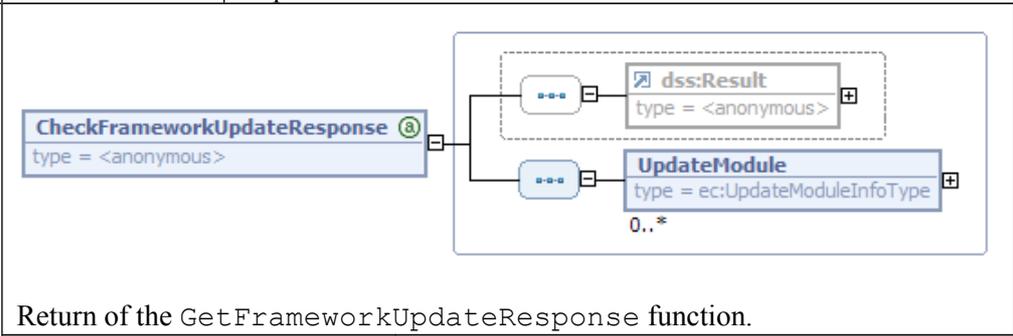
Name	Description
ModuleName	Contains the name of the module.
MajorVersion	Contains the main version number of the module.
MinorVersion	Contains the secondary version number of the module.
SubminorVersion	Contains the lower-level secondary version number of the module.

Name	CheckFrameworkUpdate	
Description	The CheckFrameworkUpdate function is used to determine which updates are available for the existing installation of the eCard-API-Framework.	
Invocation parameters		
	Request of the GetFrameworkUpdate function.	
Name	Description	
InstalledModule	Is available for each module currently installed and provides a more detailed description (see below for details).	

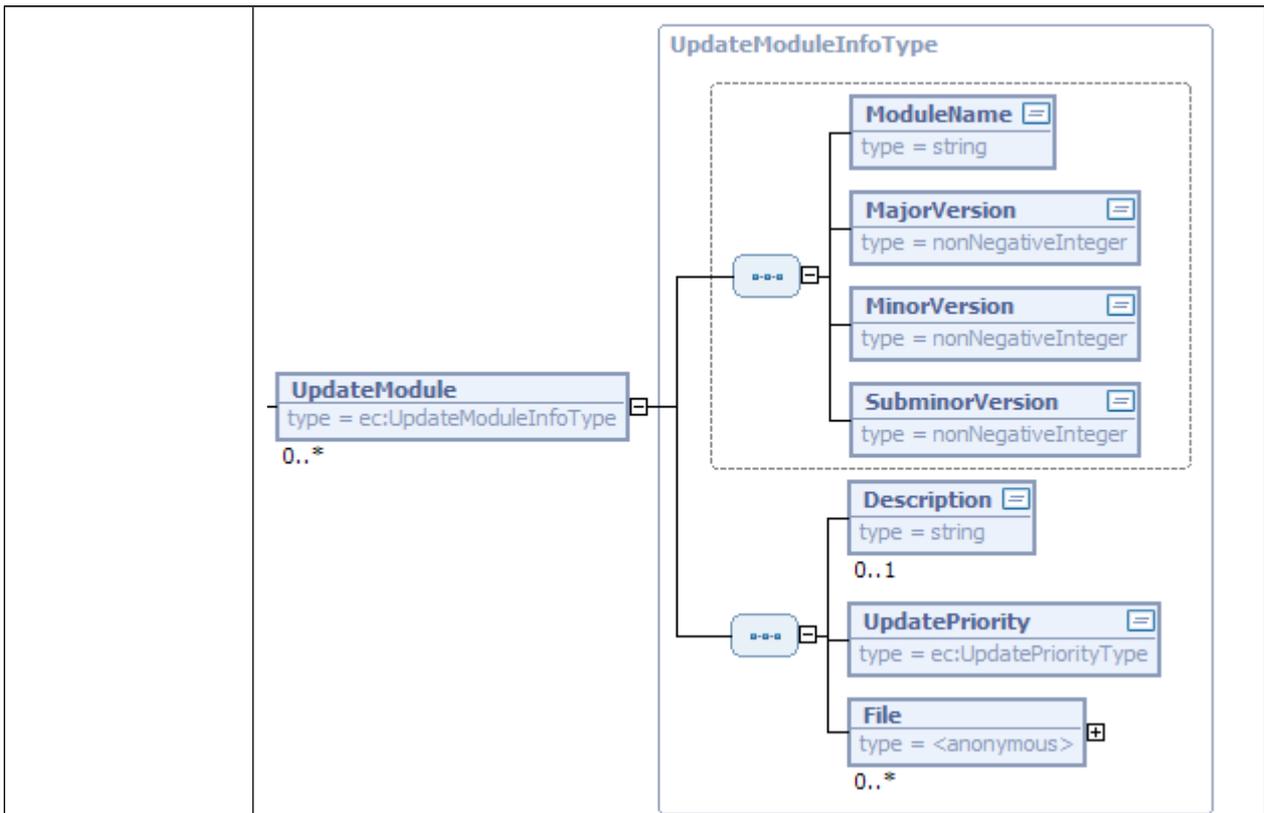


Name	Description
OSVersion	Specifies the version of the operating system used to run the respective module of the eCard-API-Framework.

Return

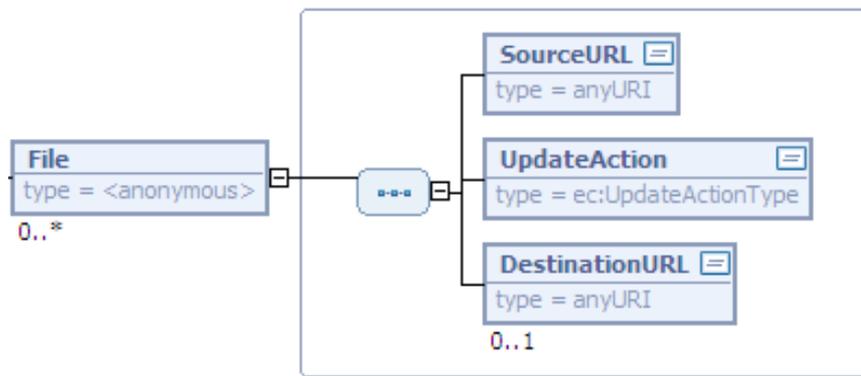


Name	Description
dss:Result	Contains the status information and the errors relating to an executed action. This element is described in more detail below.
UpdateModule	Contains information on the available update modules (see below for details).



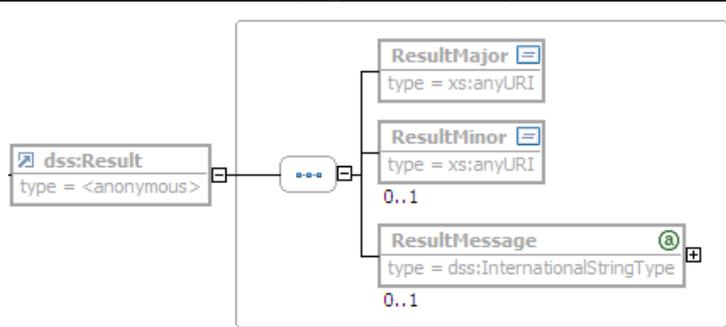
The UpdateModule element is of the UpdateModuleInfoType, which extends the ModuleInfoType explained above by adding the following element:

Name	Description
Description	MAY contain a description of the module requiring updating.
UpdatePriority	Specifies how urgent the update is. The UpdatePriorityType is defined as follows: <pre><simpleType name = "UpdatePriorityType"> <restriction base = "string"> <enumeration value="Mandatory" /> <enumeration value="Recommended" /> <enumeration value="Optional" /> </ restriction> </ simpleType></pre>
File	Contains information on the individual files which are required for the first installation or the update of this module (see below for details).



The File element is part of the UpdateModuleInfoType.

Name	Description
SourceURL	Specifies the URL of the update file under consideration. Note that it MAY be a local address and hence it would be possible to run an update of the framework with the mechanism presented here <i>without</i> any network connection.
UpdateAction	Specifies what will happen with this file. The UpdateActionType is specified as follows: <pre> <simpleType name = "UpdateActionType"> <restriction base = "string"> <enumeration value = "Execute" /> <enumeration value = "Copy" /> </ restriction> </ simpleType> </pre>
DestinationURL	Specifies the target address, if the file is to be copied.



Status information and errors in connection with CheckFrameworkUpdate (cf. [TR-03112-1] sections 3.1 and 3.2).

Name	Error code
ResultMajor	<ul style="list-style-type: none"> /resultmajor#ok /resultmajor#error /resultmajor#warning

	ResultMinor	<ul style="list-style-type: none"> • /resultminor/al/common#noPermission • /resultminor/al/common#internalError • /resultminor/al/common#parameterError • /resultminor/dp#unknownChannelHandle • /resultminor/dp#communicationError • /resultminor/dp#trustedChannelEstablishmentFailed • /resultminor/dp#unknownProtocol • /resultminor/dp#unknownWebserviceBinding • /resultminor/al/FrameworkUpdate#serviceNotAvailable • /resultminor/al/FrameworkUpdate#unknownModule • /resultminor/al/FrameworkUpdate#invalidVersionNumber • /resultminor/al/FrameworkUpdate#operationSystemNotSupported
	ResultMessage	MAY, if required, contain more detailed information on the error which has occurred.
Precondition		
Postcondition		
Note	The address to be used to invoke CheckFrameworkUpdate, the web service binding and corresponding PathSecurity- parameters are specified by the configuration of the default parameters (cf. GetDefaultParameters in [TR-03112-3]).	

A PAOS Example

An exemplary invocation of StartPAOS is sketched below:

```
POST /eService?SessionIdentifier=12345 HTTP/1.1
Host: example.com
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:paos:2006-08"; "urn:iso:std:iso-iec:24727:tech:schema"
action="StartPAOS"
<S:Envelope xmlns:S=http://schemas.xmlsoap.org/soap/envelope/
              xmlns:iso=urn:iso:std:iso-iec:24727:tech:schema>
  <S:Body>
    <iso:StartPAOS>
      <iso:SessionIdentifier>12345
    </iso:SessionIdentifier>
      <iso:ConnectionHandle>...
    </iso:ConnectionHandle>
      ...
      <iso:ConnectionHandle>...
    </iso:ConnectionHandle>
    </iso:StartPAOS>
  </S:Body>
</S:Envelope>
```

Using the PAOS Request-Response Message Exchange Patterns (cf. Section 4 of [PAOSv2.0]), the Server-SAL is able to send webservice requests.

The Server-SAL MAY for example need to invoke CardApplicationPath and CardApplicationConnect in order to establish a connection with a card available on the client system, if this has not been implicitly done by the client. In this case the StartPAOS-element contains appropriate <iso:ConnectionHandle>-elements. An exemplary invocation is sketched below:

```
HTTP 200
Content-Type: application/vnd.paos+xml
Content-Length: ...
<soap:Envelope xmlns:soap="http:// ...">
  <soap:Header>
    <paos:Request xmlns:paos="urn:liberty:paos:2006-08"
                  responseConsumerURL="/eService?SessionIdentifier=12345"
                  service="urn:iso:std:iso-iec:24727:tech:schema" mustUnderstand="1"
                  messageID="4711" actor="http://schemas.xmlsoap.org/soap/actor/next" />
  </soap:Header>
  <soap:Body>
    <soap-env:CardApplicationPath ...>
  ...
```

The client sends the CardApplicationPathResponse back to the Server with a http POST call, at the same time requesting the next command:

```
POST /eService?SessionIdentifier=12345 HTTP/1.1
Host: example.com
Accept: text/html; application/vnd.paos+xml
PAOS: ver="urn:liberty:paos:2006-08"; "urn:iso:std:iso-iec:24727:tech:schema"
Content-Type: application/vnd.paos+xml
Content-Length: ...
<Soap:Envelope ...>
  <soap:Header>
    <paos:Response xmlns:paos="urn:liberty:paos:2003-08" refToMessageID="4711"
      mustUnderstand="1" actor="http://schemas.xmlsoap.org/soap/actor/next" />
  </soap:Header>
  <Soap:Body ...>
    <soap-env:CardApplicationPathResponse ...>
      ...
  </Soap:Body ...>
</Soap:Envelope ...>
...
HTTP 200
  Content-Type: application/vnd.paos+xml
  Content-Length: ...
  <soap:Envelope
    ...
  </soap:Envelope
    ...
  >
  >
```

References

- [TR-02102] BSI: TR-02102: Kryptographische Verfahren: Empfehlungen und Schlüssellängen
[TR-03110] BSI: TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents
- [TR-03112-1] BSI: TR-03112-1: eCard-API-Framework – Part 1: Overview and Generic Mechanisms
[TR-03112-2] BSI: TR-03112-2: eCard-API-Framework – Part 2: eCard-Interface
[TR-03112-3] BSI: TR-03112-3: eCard-API-Framework – Part 3: Management-Interface
[TR-03112-4] BSI: TR-03112-4: eCard-API-Framework – Part 4: ISO24727-3-Interface
[TR-03112-5] BSI: TR-03112-5: eCard-API Framework – Part 5: Support-Interface
[TR-03112-6] BSI: TR-03112-6: eCard-API-Framework – Part 6: IFD-Interface
[TR-03112-7] BSI: TR-03112-7: eCard-API-Framework – Part 7: Protocols
[TR-03116] BSI: TR-03116: Technische Richtlinie für die eCard-Projekte der Bundesregierung
[EN14890-1] CEN: EN14890-1: Application Interface for smart cards used as secure signature creation devices - Part 1: Basic services
- [eGK-1] gematik: Specification of the electronic health insurance card - Part 1: Commands, algorithms and functions of the operating system platform, Version 2.2.2
[eGK-2] gematik: Specification of the electronic health insurance card - Part 2: Applications and application-specific structures, Version 2.2.1
- [ICAO 9303] ICAO: Doc 9303, Machine Readable Travel Documents, Part 3
[RFC2314] IETF: RFC 2314: B. Kaliski: PKCS#10: Certification Request Syntax
[RFC3280] IETF: RFC 3280: R. Housley, W. Polk, W. Ford, D. Solo: Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile
[RFC3281] IETF: RFC 3281: S. Farrell, R. Housley: An Internet Attribute Certificate Profile for Authorization
- [RFC3369] IETF: RFC 3369: R. Housley: Cryptographic Message Syntax (CMS)
[RFC3852] IETF: RFC 3852: R. Housley: Cryptographic message syntax (CMS)
[RFC4279] IETF: RFC 4279: P. Eronen, H. Tschofenig: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)
- [RFC4346] IETF: RFC 4346: T. Dierks, E. Rescorla: The Transport Layer Security (TLS) Protocol, Version 1.1
[RFC4492] IETF: RFC 4492: S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, und B. Moeller: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)
- [ISO24727-3] ISO: ISO/IEC 24727-3: Identification Cards — Integrated Circuit Cards Programming Interfaces — Part 3: Application Interface
[ISO7816-15] ISO: ISO/IEC 7816-15: Identification cards - Integrated circuit(s) cards with contacts — Part 15: Cryptographic information application
[ISO7816-4] ISO: ISO/IEC 7816-4: Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange
[ISO7816-8] ISO: ISO/IEC 7816-8: Identification cards — Integrated circuit cards — Part 8: Security related interindustry commands
[ISO7816-9] ISO: ISO/IEC 7816-9: Identification cards - Integrated circuit cards - Part 9: Commands for card management
- [PAOSv1.1] Liberty Alliance Project: Liberty Reverse HTTP Binding for SOAP Specification, Version v1.1
[PAOSv2.0] Liberty Alliance Project: Liberty Reverse HTTP Binding for SOAP Specification, Version v2.0
[ISIS-MTT-2] TeLeTrust: ISIS-MTT Specification - Part 2: PKI Management, Version 1.1
[SOAPv1.1] W3C: W3C Note: Simple Object Access Protocol (SOAP) 1.1