INTERNATIONAL STANDARD

ISO/IEC 24727-4

First edition
2008-11-01

AMENDMENT 1
2014-04-01

Identification cards — Integrated circuit card programming interfaces —

Part 4: Application programming interface (API) administration

AMENDMENT 1

Cartes d'identification — Interfaces programmables de cartes à puce —

Partie 4: Administration d'interface de programmation (API)

AMENDEMENT 1

Reference number
ISO/IEC 24727-4:2008/Amd.1:2014(E)

Contents

Foreword

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

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

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

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

Amendment 1 to ISO/IEC 24727-4:2008 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and personal identificaition*.

XML encodings have become more and more used in the field of IAS (Identity, Authentication and (digital) Signature), Identity Management and general networking communication. To enhance interoperability with existing networking systems and federated identification and authorization systems (e.g. SAML, OpenID, etc.) standardization of an XML representation of the API and data structures of ISO/IEC 24727-3 is essential.

In order to support this addition to the ISO/IEC 24727-3 scope, the relevant stack configurations in ISO/IEC 24727-4 will be updated and/or amended. The rules governing the use of various marshalling/un-marshalling procedures will be aligned with the amendment to ISO/IEC 24727-3.

This Amendment has been prepared to:

1. Extend and update as necessary the stack configurations to address the XML representation such that it is compatible with the relevant XML-based standards (e.g. SAML).
2. Clarify use of secure messaging.
3. As a result of Amendments under development for other parts of 24727, portions of this standard may be deleted and referenced.
4. Consider additional forms of secure messaging and consider separating the security of information transferred across a general network versus security of information transferred across the card interface.
5. Refine TC_API to allow channel initiation for various mechanisms; e.g. web service communication (SOAP PAOS), AJAX.
6. Update the current XML specifications to align with ISO and not import 3 party schemas e.g. OASIS.
7. Remove ambiguities by elaborating and re-specifying concepts that may not be clear in the current standard.
8. Incorporate concepts that are captured in other parts of ISO/IEC 24727 but are more relevant for ISO/IEC 24727-4.
9. Include C and Java bindings in a Normative Annex (for C) and an Informative Annex (Java); moved from Part 5

10. Consider relocating data structure generation to the local machine level; e.g. remote ICC Stack

Note: A hybrid numbering scheme for new figures and tables is used (e.g. Figure 7-1) in order to fit more seamlessly with the original document; i.e. to NOT require many modifications to simply change figure and table numbers.

Identification cards — Integrated circuit cards programming interfaces —

Part 4: Application programming interfaces (API) administration

AMENDMENT 1

Page 12

Add the following new subclause before Clause 6:

5.7 Extensions of the Service Access Layer

5.7.1 Extensions under ISO/IEC 7816-15

This clause comprises an enhancement of the functionality assigned to the local platform through which is accessed the card-application. In so doing, it re-uses the existing ISO/IEC 24727-4 middleware stack configurations with only a change bearing upon the location of the component handling ISO/IEC 7816-15 -based registry. The main objective being to alleviate the workload of the client-application platform by supplying it upon request with pre-built data structures for interoperability (ISO/IEC 24727-3).

5.7.2 SAL API Lite

ISO/IEC 24727-2 AMD 1 envisions explicit (normative and informative elements, including the use of the ISO/IEC 7816-15 based Registry. As such, ISO/IEC 24727-2 standardizes the use of ISO/IEC 7816-15 data structures as "discovery information" that is communicated throughout the ISO/IEC 24727 stack. To achieve this discovery process, it is necessary to locate an ISO/IEC 7816-15 based Procedural Elements component within the platform connected to the card-application for the Remote-ICC-stack and the Opaque-ICC-stack (see Figure 7-1) as follows:

- This component is in charge of linking the entities defined at the Service Access Layer (API) (e.g. Differential Identity, DataSet, DSI) with typical "on-card" entities such as keys, files and Access Control Rules. Accordingly enabled with this Registry processing capability, this component is ONLY in charge of retrieving the ISO/IEC 7816-15 based Registry, parsing it, and generating the interoperability data structures representing the objects handled by the SAL API in conformance with ISO/IEC 24727-3.

- this component is part of a SAL API Lite implementation that implements a subset of existing SAL API herein called SAL API Lite

- the SAL API Lite is comprised ONLY of the the SAL API functions that serve to retrieve data structures for interoperability as DataSet, DSI, DID, ACL, etc. to supply it to the client-application upon request.

5.7.3 SAL API Lite Supported Requests:

The SAL API Lite interface provides a subset of the full ISO/IEC 24727-3 API. Specifically included are the following requests:

- **CardApplicationConnect(),**
- **CardApplicationList**(),

- **CardApplicationServiceList**(),
- **CardApplicationServiceDescribe**(),
- **DataSetList**(),
- **DataSetSelect**(),
- **DSIList**(),
- **DSIRead**(),
- **DIDList**(),
- **DIDGet**(),
- **ACLLIst**()

The SAL API Lite requests are limited in the following ways:

5.7.3.1 Registry Delegation

The Service Access Layer implemented on the same platform as the client-application shall delegate the Registry processing capability to the SAL API Lite implementation.

5.7.3.2 Registry Bootstrap

The SAL API Lite implementation shall start the ISO/IEC 7816-15 based Registry processing at bootstrap and save processing time to the Service Access Layer implementation by making available all the data structures defined at SAL API Lite interface.

5.7.3.3 Registry Connection

The SAL API Lite implementation shall provide to the application running on a local machine to which the card-application is connected e.g to allow for GUI to expose the user all or part of his on-card ISO/IEC 7816-15 based Registry.

5.7.3.4 ICC-Resident-Stack Exclustion

The SAL API Lite component does not fit in an ICC-Resident-stack for the following reasons: the SAL-Agent is located on-card and not visible to the outside world, and additionally, on ICC-Resident-stack, the data structures for interoperability can be built-in structures already available on-card, so they do not need to be derived from a ISO/IEC 7816-15 based Registry.

5.7.3.5 Static Library Option

Optionally, the SAL API Lite implementation may rely on a static or dynamic library to explore the on-card ISO/IEC 7816-15 based Registry. To this intent, informative Annex E is proposed with a comprehensive description and binding for an API typically handling on-card ISO/IEC 7816-15 based Registry.

Figure 7-1 illustrates a Full Network Stack that makes use of a SAL API Lite registry facility. In addition, this figure illustrates the use of a procedural element that can translate the formal language representation of the ISO/IEC 24727-3 API functions directly into card specific APDUs. The only required ISO/IEC 24727-2 GCI APDU is the ENVELOPE APDU. The contents of the ENVELOPE APDU can be, for example, DER-TLV represented API functions. Thus, the procedural elements can function as a full Service Access Layer.

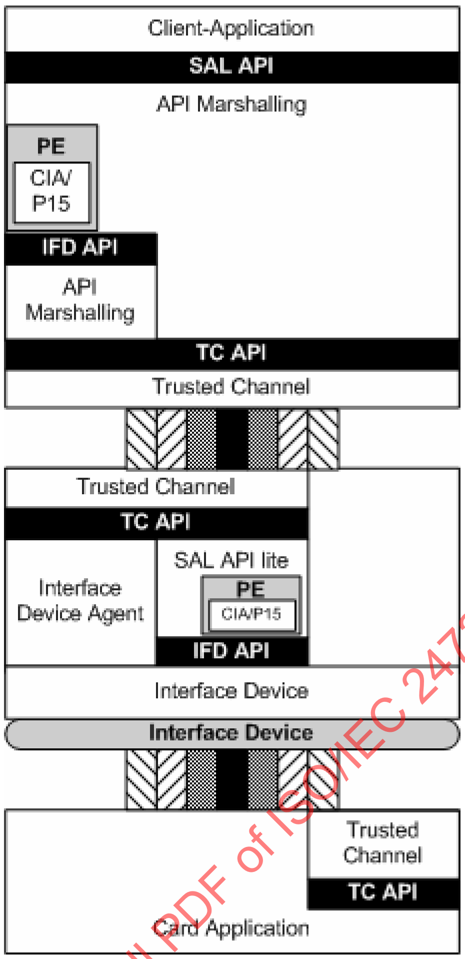

Figure 7-1 —Network stack

5.7.4 Secure Messaging

Within a stack configuration as shown in Figure 7-1, a client-application may establish an end-to-end secure channel with a card-application. The secure APDU commands generated by the SAL implementation are delivered as IFD API marshalled commands to the local host where the Interface Device Agent redirects them to the Interface Device. The SAL API Lite component does not process secure APDUs. Accordingly, whenever the ACL controlling the access to a data structure e.g. DataSet, requires secure messaging, the transaction shall go through the following sequence :

1. client-application sends for the first time during a session a DataSetList call through SAL API
2. the API call is marshalled into either XML or DER-TLV and sent through TC to the SAL API Lite component
3. the SAL API Lite builds the requested data structure (i.e DataSetNameList) as part of the DataSetListResult and returns it through TC layer to the SAL.
4. the SAL hands on the DataSetListResult to the client-application.
5. the client-application sends an ACLList call through SAL API with a parameter targetType denoting a data-set and a parameter targetName related to a named DataSet that was retrieved from the just received list of DataSetNameList (before this call, the client-application may optionally selects a named DataSet and sends a DataSetSelect call through SAL API)
6. the API call is marshalled into either XML or DER-TLV and sent through TC to the SAL API Lite component

7. the SAL API Lite builds the requested data structure (i.e AccessControlList) as part of the ACLListResult and returns it to through TC layer to the SAL.
8. the SAL hands on the ACLListResult to the client-application.
9. the client-application parses the set of AccessRules contained in ACLListResult and figures out which Differential-Identity is to be verified in order to get access to the DataSet contents (with the assumption that the targetted DataSet is protected with a DID controlling access to its contents e.g DIDRead requiring mutual authentication with establishment of secure messasing).
10. the client-application sends a DSIList call through SAL API
11. the API call is marshalled into either XML or DER-TLV and sent through TC to the SAL API Lite component.
12. the SAL API Lite builds the requested data structure (i.e DSINameList) as part of the DSIListResult and returns it to through TC layer to the SAL. in order to allow the client-application to read properly the DSI contents, the SAL API Lite maps the DSI attributes onto on-card attributes; for this purpose the SAL API Lite uses the Cryptographic Information Application (CIA or P15) to figure out the on-card attributes related to each DSI in the DSIListResult. Alternatively, the SAL API Lite may send a SELECT APDU command to the card with parameter P2 requesting the Control Parameter (CP) Template (see ISO/IEC 7816-4); accordingly, a DO'98' within CP denotes DO instances, a DO'9B' within CP denotes EF instances in either DF or ADF, and a DO'97' within CP denotes DF instances in either DF or ADF. To map those on-card attributes onto DSI attributes, an additional parameter is required in DSIListResult definition in ISO/IEC 24727-3.
13. the SAL hands on the DSIListResult to the client-application
14. the client-application sends a DIDAuthenticate (according step 9 above) to fulfil the access rules controlling the reading of DSI contents.
15. the SAL parses the DID requirements (i.e DIDName, DIDAuthenticationData) and starts the authentication protocol with the card i.e SAL generates the appropriate APDU commands and send them out to the card through IFD API and TC layer.
16. Once the end-to-end secure messaging is established between the card and the SAL, the SAL delivers the API return code to the client-application
17. the client-application then sends a DSIRead call through the SAL API.
18. the SAL generates the corresponding APDU command conforming to DSI attributes received in step 12 above; then SAL secures the APDU commands with current session keys and sends it via the IFD API, TC and Interface Device to the Card Application;
19. the SAL recovers the DSI contents and builds the dsiContent as part of the DSIReadResult, then return it to the client-application.

Table 7-1 indicates the messages exchanged between SAL API and SAL API Lite during a transaction with secure messaging :

Table 7-1 – Command Progression

Client-Application	SAL	IFD Proxy	TC	IFD Agent	SAL API Lite	IFD	Card-Application
					Registry processing		
DataSetList	DataSetList	←→	←→	←→	DataSetList		
DataSetSelect	DataSetSelect	←→	←→	←→	DataSetSelect		
ACLList	ACLList	←→	←→	←→	ACLList		
DSILIst	DSILIst	←→	←→	←→	DSILIst		
DIDAuthenticate	APDU C-RP (*)	←→	←→	←→		←→	←→
	← SECURE MESSAGING →						
DSIRead	APDU C-RP	←→	←→	←→		←→	←→

(*) APDU C-RP = APDU command-response pair i.e GET DATA or READ BINARY

Page 47

Add the following new clause before Annex A:

8 Registry Implementations

8.1 ISO/IEC 7816-15 registry implementation

To achieve interoperability among various Service Access Layer implementations, it is necessary to record the mechanisms used to translate between ISO/IEC 24727-3 API functions and ISO/IEC 24727-2 GCI APDUs. This clause defines how to prepare this record in the form of an ISO/IEC 7816-15 formatted Registry.

Included is the representation of ISO/IEC 24727-3 Card-Application discovery information as an ISO/IEC 7816-15 Cryptographic Information Application.

The ISO/IEC 7816-15 representation of an ISO/IEC 24727 card-application contains all the information that the ISO/IEC 24727-2, ISO/IEC 24727-3 implementations need to realize the ISO/IEC 24727 specified interoperable connection between the client-application and the card-application.

Data Structures for Interoperability are discovery information made available to the SAL by the card through the bootstrap mechanism, unless it is provided by other means. The SAL API or the SAL API Lite shall undertake the processing of the discovery information in order to dynamically generate the data structures defined in ISO/IEC 24727-3 as Access Control List, Differential-Identity, Data-Set and Data Structures for Interoperability (DSI), and Card-Application Services and Actions. These data structures shall be surfaced to the client-application upon request through the SAL API or the SAL API Lite.

To allow for coding of access control rules conforming to ISO/IEC 7816-15 implementation, a mapping of SAL API actions onto ISO/IEC 7816-15 *accessMode* attributes is described. This mapping is laid on the extension of accessMode according the second amendment of ISO/IEC 7816-15.

The information in the ISO/IEC 7816-15 representation of an ISO/IEC 24727 card-application is referred to informally in ISO/IEC 24727-1 as the *discovery information* and in ISO/IEC 24727-4 as an *ISO/IEC 7816-15 based Registry.*

To implement the ISO/IEC 7816-15 based Registry, several ASN.1 definitions are available:

- Clause 8.1.3.1 establishes the ASN.1 types for translation of SAL API calls into card-specific APDU; the APDU command parameters rendering each SAL API Action can be so determined by the issuer and DER-TLV encoded then hosted on-card as part of the Cryptographic Information Application or on a remote repository that can be accessible though the interface of which the XML-Binding is described in Annex I [informative] "interoperable access to the repository".

- Annex H[informative] illustrates the actual ASN.1 reference Module implementing clause 8.1.3.1. Common ASN.1 types are imported from ISO 24727-3; only types newly designed in the Amendment are defined within this module.

- Annex G[informative] illustrates application of the rules and guidelines described in clause 8.1.1 "ISO/IEC 24727-3 data structures mapping" to build an ISO/IEC 7816-15 based Registry for a fictitious service called "myService". In G.1.1.2.1, an ASN.1 value notation illustrates an implementation example; accordingly, DER-encoding may be derived from this notation.

8.1.1 ISO/IEC 24727-3 data structures mapping

The data structures that may be surfaced to the client-application upon request are DataSet, DSI, ACL, Differential-Identities, list of Card-Application Services, list of Differential-Identities.

The SAL generates these data structures from the information available in CardApplicationServiceDescription that is defined as an ASN.1 SEQUENCE of ISO/IEC 7816-15 CIAInfo value along with consecutive ISO/IEC 7816-15 CIOChoice(s) value(s).

The interpretation of CardApplicationServiceDescription value in terms of ACL, Differential-Identities, Services, DataSet or DSI, is based on the mapping described in the following sub-clauses.

8.1.1.1 DataSet

Table 8-1 indicates the mapping of a DataSet onto a DataContainerObjectChoice and describes how the DataSet's ACL can be derived from it. In this and following tables, an “X” in the “ACL items” column indicates that this attribute may be controlled through an Access Control List (ACL) entry.

Table 8-1 — Mapping DataSet to ACL

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**ACL items**	
DataContainerObject-Choice iso7816DO commonObjectAttributes	commonObjectAttributes.label	DataSet Name	X	The label is ASCII encoded and shall evaluate to '**DATA-SET**' used as a prefix denoting the target type, concatenated with the DataSet actual name.
	accessControlRule.accessMode	Action	X[1]	Shall conform to Table 10 and Table 8-8
	accessControlRule.securityCondition	Ref to DID		authId referring to a CIO within the cryptographic information application (DF.CIA)
ClassAttributes	classAttributes.applicationName			Either applicationName or application OID of the card-application owning the DataSet. Shall not be NULL.
TypeAttributes	<unused>			NULL
ISO/IEC 7816 DO container for the DSI(s) within the DataSet				
CommonObjectAttributes	commonObjectAttributes.label	DSIName		The DSI Name shall be ASCII encoded.
	accessControlRule.accessMode	<unused>		DSI's accessRules are controlled by DataSet

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	ACL items	
	accessControlRule.securityCondition	<unused>		DSI's accessRules are controlled by DataSet accessRules
ClassAttributes		<unused>		
TypeAttributes	efidOrTagChoice	location of the actual resource (EF,DF, ADF, DO) hosting the DSI content.		File Id or extended Path (acc. ISO/IEC 7816-15:2004/AM2) to the DSI resource

[1] In case the action is controlled by a logical combination of DIDs, several consecutive securityCondition values may provide different security conditions applying to the same access Mode denoting the SAL API action. The logical operation combining the DIDs is AND or OR depending on the ASN.1 encoding of the securityContitions.

The DIDs referenced by the access rules protecting a DataSet are either Authentication Information Objects or Private Objects or Secret Key. Secret Key and Private Key mapping to Differential-Identities is presented in subsequent tables.

8.1.1.2 CardApplication

Table 8.2 shows the mapping of a CardApplication onto a DataContainerObjectChoice and describes how the CardApplication's ACL can be derived from it.

Table 8-2 — mapping CardApplication to DataContainerObjectChoice

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	ACL items	
DataContainerObject-Choice ISO/IEC 7816 DO commonObjectAttributes	commonObjectAttributes.label	Card-Appli-cation Name	X	The label is ASCII encoded and shall valuate to **'CARD-APPLICATION'** used as a prefix denoting the target type, concatenated with the CardApplication actual name.
	accessControlRule.accessMode	SAL API Action	X [1]	SAL API actions shall conform to value in Table 10 and Table 8-8

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**ACL items**	
	accessControlRule.securityCondition	Ref to DID		May be authId referring to a CIO within the cryptographic information application (DF.CIA)
ClassAttributes	classAttributes.applicationName			Either applicationName or application AID. Shall not be NULL.
TypeAttributes	<unused>			NULL
ISO/IEC 7816 DO container for each target resource of the CardApplication (DataSet Name, DID Name)				
CommonObjectAttributes	commonObjectAttributes.label	DataSet or DID Name		The target name shall be ASCII encoded with a prefix denoting the target type concatenated with the actual target name.
	accessControlRule.accessMode	<unused>		Refer to the named target
	accessControlRule.securityCondition	<unused>		Refer to the named target
ClassAttributes		<unused>		
TypeAttributes	efidOrTagChoice	location of the actual resource (EF,DF, ADF, DO) hosting the named target.		File Id or extended Path (acc. ISO/IEC 7816-15:2004/AM2) to the named target resource

1 In case the Action is controlled by a logical combination of DIDs, several consecutive securityCondition values may provide different security conditions applying to the same access Mode denoting the SAL API Action. The logical operation combining the DIDs is AND or OR depending on the ASN.1 encoding of the securityConditions.

8.1.1.3 Service

Table 8.3 shows a possible alternative for the mapping of a Service onto a DataContainerObjectChoice But since a Named Service is not a target, no ACL is attached to it according ISO/IEC 24727. Deriving ACL from Named Service is not recommended by the present standard.

Table 8-3 — mapping Service to ACL

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**ACL items**	
DataContainerObject-Choice iso7816DO for the Service commonObjectAttributes	commonObjectAttributes.label	Service Name	X	The Service Name shall be comprised of ASCII coded prefix **'SERVICE'** concatenated with the respective Service Name conforming to ISO/IEC 24727-3 naming rules
	accessControlRule.accessMode	SAL API Action	X[1]	two supported Actions for Authorization Service : ACL_LIST, ACL_MODIFY encoding shall conform to Table 10 and Table 8-8
	accessControlRule.securityCondition	Ref to DID		authId. ACL_LIST may valuate to ALWAYS. authId is the cross-reference to an Authentication Object protecting the current CIO
ClassAttributes	applicationName			applicationName or/and application OID. Shall not be NULL. It gives the location of the executable code implementing the named Service within the card application.
	applicationOID			
TypeAttributes	<unused>			NULL

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**ACL items**	
iso7816DO container for the Actions within the Service				
CommonObjectAttributes	commonObjectAttributes.label	<optional> Action Name		The Action Name shall conform to ISO/IEC 24727-3 naming rules. If present, it clears ambiguity for SAL API actions sharing the same AccessMode byte value.
	accessControlRule.accessMode	SAL API Action		Shall conform to Table 10 and Table 8-8
	accessControlRule.securityCondition	Ref to DID	ACL^{2}	**authId**. this is the cross-reference to an Authentication Object protecting the current CIO
ClassAttributes	applicationName			The applicationName or/and application OID shall not be NULL. It gives the location of the executable code implementing the named Service within the card application. The iD may be used to associate the current data container object with some other CIO (e.g.: a private key information object, a public key information object, a certificate information object, a secret key information object)
	applicationOID			
	CommonDataContainerObjectAttributes.iD	Ref to a CIO		

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	**ACL items**	
TypeAttributes	efidOrTagChoice	location of the cryptographic information object executing the Action (e.g : a PrK for a SIGN action)		File Id or Path to the CIA directory file containing the CIO referenced by the **ClassAttributes.iD** above
[1] This ACL is for Authorization Service that applies for all card-application services. [2] This ACL relates to a specific Action within the card-application named Service				

8.1.1.4 Differential-Identity

In the context of ISO/IEC 24727-3, Differential-Identities are used to describe objects for authentication and cryptographic usage. As a consequence, ISO/IEC 7816-15 AuthenticationObjectChoice structures are used to describe differential identities with the focus on authentication.

KeyObjectChoice structures shall be used to qualify differential identities which should mainly be used for the cryptographic methods (e.g. Encipher, Decipher, Sign).

8.1.1.4.1 Differential-Identities for authentication

The structure of a differential identity is heavily depending on the used authentication protocol. The marker and qualifier structures are only known after the authentication protocol has been identified. Therefore, the first step in the discovery mechanism is to retrieve the correct value for the authentication protocol while parsing the service description of the Application Capability Descriptor.

The discovery mechanism for the authentication protocol value is shown in Figure 8-1. The mechanism is starting with parsing the AuthenticationObjectChoice. At the beginning, the TypeAttributes of the AuthenticationObjectChoice can be determined. If the TypeAttributes are PasswordAttributes or BiometricAttributes, the type of the authentication protocol shall be either the PIN Compare or the Biometric Compare protocol.

In case the TypeAttributes are ExternalAuthObjectAttributes, it has to be investigated which choice was made for the ExternalAuthObjectAttributes:

— If CertBasedAuthenticationAttributes are used, the protocol shall be the Asymmetric External Authenticate protocol,

— In the case AuthKeyAttributes are used, it's the Symmetric External Authenticate Protocol unless otherwise specified by the related objId : the corresponding AlgorithmInfo within the CIAInfo structure of the adequate application shall be used to retrieve the objId. If the equivalent non DER-TLV encoded value of this objId is starting with "1 0 24727 3", the value shall be directly used to determine the authentication protocol value.

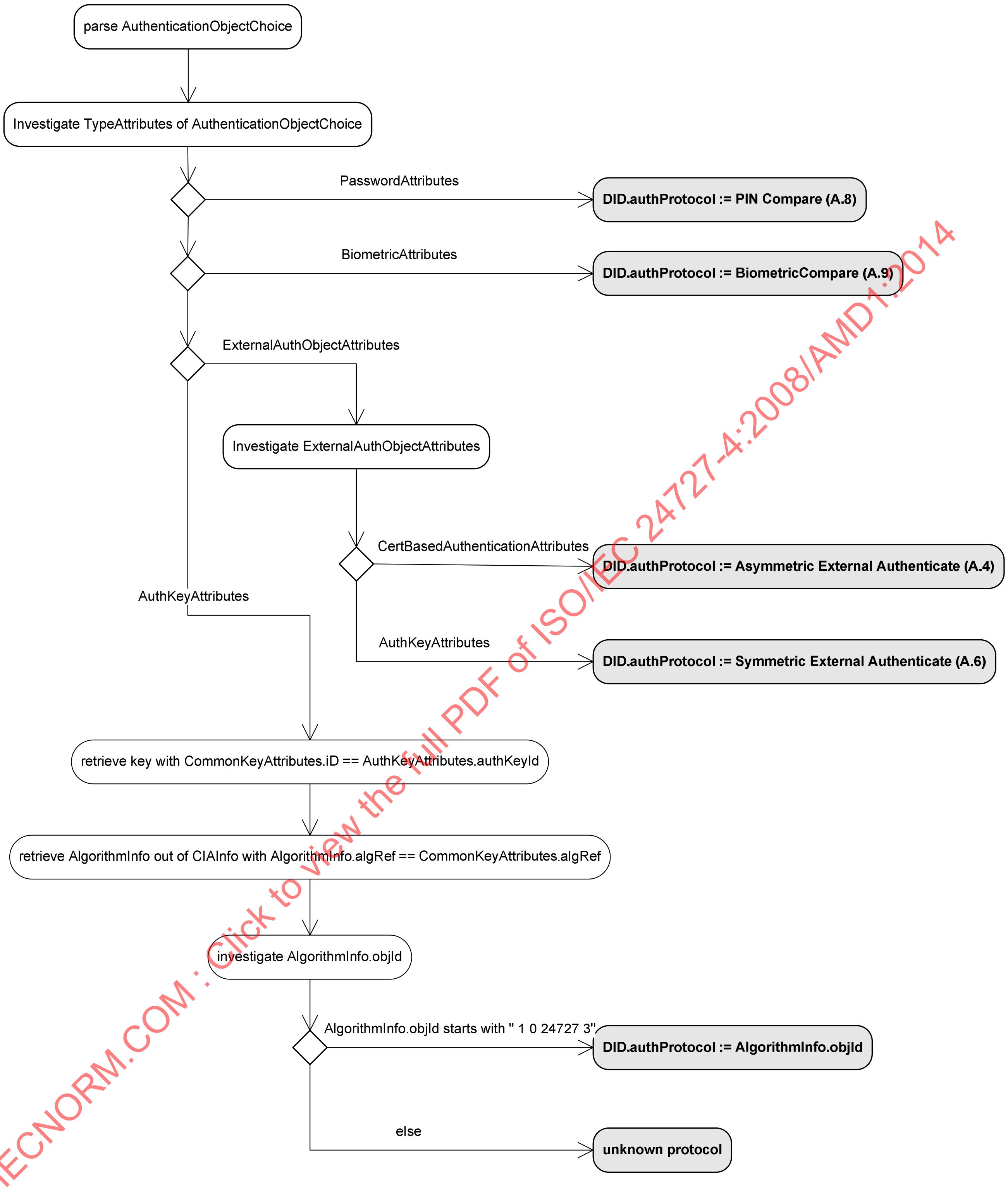
parse AuthenticationObjectChoice
Investigate TypeAttributes of AuthenticationObjectChoice
PasswordAttributes
DID.authProtocol := PIN Compare (A.8)
BiometricAttributes
DID.authProtocol := BiometricCompare (A.9)
ExternalAuthObjectAttributes
Investigate ExternalAuthObjectAttributes
CertBasedAuthenticationAttributes
DID.authProtocol := Asymmetric External Authenticate (A.4)
AuthKeyAttributes
AuthKeyAttributes
DID.authProtocol := Symmetric External Authenticate (A.6)
retrieve key with CommonKeyAttributes.iD == AuthKeyAttributes.authKeyId
retrieve AlgorithmInfo out of CIAInfo with AlgorithmInfo.algRef == CommonKeyAttributes.algRef
investigate AlgorithmInfo.objId
AlgorithmInfo.objId starts with " 1 0 24727 3"
DID.authProtocol := AlgorithmInfo.objId
else
unknown protocol

Figure 8-1 - Discovery of the authentication protocol value

Table 8.4 shows the mapping of an Authentication Object to the related Differential-Identity

Table 8-4 — mapping authObject to DID

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	**DID item**	
AuthenticationObject-Choice commonObjectAttributes	commonObjectAttributes.label	Name denoting the authObject purpose (ASCII-encoded)	DIDName	
	commonObjectAttributes.authId	Cross-reference to the algorithm used to authenticate this DID.	Auth Protocol	Matches the value AlgorithmInfo.algRef in CIAInfo.supportedAlgorithms
	accessControlRule.accessMode	SAL API Action		Shall conform to Table 10 & Table 8-8; Alternatively (not recommended), the access rules controlling the SAL API Actions executable upon this DID may be found in Differential-Identity Service ACLs
	accessControlRule.securityCondition	ISO/IEC 7816-15 compatible		
ClassAttributes	authID	Cross-reference from any CIO to this DID. Shall match for instance authId used as securityCon-ditions attribute for DataSets or Services.		*CommonAuthenticationObjectAttributes.authID* enables DID lookup for private objects or dataContainerObjects.
	authReference	Key reference of the cryptographic object involved in the DID Auth Protocol in security environments	Marker	

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	**DID item**	
	seIdentifier	If present, it identifies the security environment to which the DID Marker and Auth Protocol belong.		
TypeAttributes	pwdFlags.local	If present (for pwd and biometric template CIOs), it denotes whether the object is local or global	Scope	
	Any other TypeAttribute shall comprise the Qualifier (i.g pwdType, minLength, storedLength, maxLength, padChar, path)	If present, shall be part of the Qualifier	Qualifier	once the OID of the authentication protocol is determined by the terminal, the actual Qualifer content shall conform to the definition provided in Annex A of ISO/IEC 24727-3.

Table 8.5 shows the mapping of a Secret Key to a Differential-Identity.

Table 8-5 — mapping Secret Key to DID

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**DID Item**	
SecretKeyChoice commonObjectAttributes	commonObjectAttributes.label	Name denoting the secretKey purpose (ASCII-encoded)	DIDName	
	commonObjectAttributes.authId	Cross-reference to the authentication object used to protect this secret key	Auth Protocol	May be employed as cross-reference to the algorithm used to authenticate this DID. If so, it shall matches the value AlgorithmInfo.algRef in CIAInfo.supportedAlgorithms. But it is recommanded to use authId as a cross-reference to the authentication object used to protect this secret key
	accessControlRule.accessMode	SAL API Action		Shall conform to Table 10 and Table 8-8. Alternatively (not recommended), the access rules controlling the ISO/IEC 24727-3 Actions executable upon this DID may be found in Differential-Identity Service ACLs.
	accessControlRule.securityCondition			
ClassAttributes	iD	Unique identifier of the Secret Key in the ISO/IEC 7816-15 structure		Used as cross-reference from access rules of DataSets or Services. iD value shall match authId value.

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	DID Item	
	usage	Indicates the possible usage of the key		Denotes the usage authorized for the DID Marker
	native	<unused> Mandatory attribute but unused for DID building		Denotes whether the algorithm use by the key is implementated in the card hardware
	keyReference	Unique reference	Marker	Reference used in CRT (within security environments) or cryptographic commands to reference the key in the card-Application. Encoded as INTEGER
	algReference	identifies algorithms that may be used with the key by referencing supported-Algorithm values from CIAInfo.	Auth Protocol	Preferably used instead of CommonObjectAttributes.authId. This attribute is Optional but shall be used for DID mapping.
Sub-Class Attributes	CommonSecretKeyAttributes.keyLen	Key length in bytes encoded as INTEGER	Qualifier	This attribute is Optional. If used it shall be figured within the DID Qualifier
TypeAttributes	SecretKeyAttributes.value	The value shall be a Path to a file containing the key or to some representation of the key. If used, shall be part of the Qualifier		This attribute is mandatory

Table 8.6 indicates the mapping of a RSA Private Key to a Differential-Identity

Table 8-6 — mapping private Key to DID

ISO/IEC 7816-15 attribute	**ISO/IEC 7816-15 sub-attribute(s)**	**ISO/IEC 24727-3**		**Comments**
		Description	**DID Item**	
PrivateKeyChoice commonObjectAttributes	commonObjectAttributes.label	Name denoting the PrivateKey purpose (ASCII-encoded)	**DIDName**	
	userConsent	<unused>		
	commonObjectAttributes.authId	Cross-reference to the authentication object used to protect this Private Key	(Auth Protocol)	May be employed as cross-reference to the algorithm used to authenticate this DID. If so, it shall match the value AlgorithmInfo.algRef in CIAInfo.supportedAlgo-rithms. But it is recommanded to use authId as a cross-reference to the authentication object used to protect this Private Key
	accessControlRule.accessMode	SAL API Action		Shall conform to Table 8-10 & Table 8-8. Alternatively (not recommended), the access rules controlling the ISO/IEC 24727-3 Actions executable upon this DID may be found in Differential-Identity Service ACLs.
	accessControlRule.securityCondition			
ClassAttributes	iD	Unique identifier of the Private Key in the ISO/IEC 7816-15 structure		Used as cross-reference from access rules of DataSets or Services. iD value shall match authId value.

ISO/IEC 7816-15 attribute	ISO/IEC 7816-15 sub-attribute(s)	ISO/IEC 24727-3		Comments
		Description	**DID Item**	
	usage	Indicates the possible usage of the key		Denotes the usage authorized for the DID Marker
	native	<unused> Mandatory attribute but unused for DID building		Denotes whether the algorithm use by the key is implementated in the card hardware
	keyReference	Unique reference	Marker	Reference used in CRT (within security environments) or cryptographic commands to reference the key in the card-Application. Encoded as INTEGER
	algReference	identifies algorithms that may be used with the key by referencing supported-Algorithm values from CIAInfo.	(Auth Protocol)	Preferably used instead of CommonObjectAttributes.authId. This attribute is Optional but shall be used for DID mapping.
Sub-Class Attributes	CommonPrivateKeyAttributes.	<unused>. If used, shall be part of the Qualifier	Qualifier	This attribute is Optional. If used it shall be figured within the DID Qualifier
TypeAttributes	PrivateRSAKeyAttributes.value and PrivateRSAKeyAttributes.modulusLength	Shall be part of the Qualifier		This attribute is mandatory

8.1.2 SAL API Action mapping onto ISO/IEC 7816-15 attributes

Table 8.7 provides the mapping of all SAL API actions in terms of ISO/IEC 7816-15 *accessMode* attributes.

The ISO/IEC 7816-15 containers from which the SAL API access control rules can be extracted are provided.

Whenever a SAL API action is always authorized, the ISO/IEC 7816-15 *securityCondition* byte is set to 'ALWAYS' value (see comments in Table 8.7)

Table 8-7 — SAL API action mapping onto ISO/IEC 7816-15 attributes (1/2)

SAL-API action	Targets as defined in ISO/IEC 24727-3 §5.3	read(0)	update(1)	execute(2)	delete(3)	attribute(4)	pso_cds(5)	pso_verify(6)	pso_dec(7)	pso_enc(8)	int_auth(9)	ext_auth(10)	7816-15 container	comments
Card-application-service access														
Initialize				1										ALWAYS
Terminate				1										ALWAYS
CardApplicationPath				1										ALWAYS
Connection service														
CardApplicationConnect	C-App			1										ALWAYS
CardApplicationDisconnect	C-App			1										ALWAYS
CardApplicationStartSession	C-App			1							1	1	DataContainerObjectChoice (1)	see DIDAuthenticate
CardApplicationEndSession	C-App			1										ALWAYS
Card-application-service														
CardApplicationList	C-App	1											DataContainerObjectChoice (1)	(1) : from the DataContainerObjectChoice, the securityCondition attribute applying to the SAL-API Actions can make reference to a cryptographic information object through its identifier authId. This authId shall point to an object that can be an AuthenticationObject representing the DifferentialIdentity protecting the SAL-API Action.
CardApplicationCreate	C-App		1										DataContainerObjectChoice (1)	
CardApplicationDelete	C-App				1								DataContainerObjectChoice (1)	
CardApplicationServiceList	C-App	1											DataContainerObjectChoice (1)	
CardApplicationServiceCreate	C-App		1										DataContainerObjectChoice (1)	
CardApplicationServiceLoad	C-App		1										DataContainerObjectChoice (1)	
CardApplicationServiceDelete	C-App				1								DataContainerObjectChoice (1)	
CardApplicationServiceDescribe	C-App	1											DataContainerObjectChoice (1)	
ExecuteAction	C-App			1									DataContainerObjectChoice (1)	
Named data service														
DataSetList	C-App (3)	1											DataContainerObjectChoice (1)	(3) : the target can be as well DataSet. This means that DataSet access control rules, if any, shall overwrite C-App access control rules.
DataSetCreate	C-App			1	1								DataContainerObjectChoice (1)	
DataSetSelect	DataSet	1											DataContainerObjectChoice (1)	
DataSetDelete	C-App (3)				1								DataContainerObjectChoice (1)	
DSIList	DataSet	1											DataContainerObjectChoice (1)	
DSICreate	DataSet		1	1		1							DataContainerObjectChoice (1)	
DSIDelete	DataSet				1								DataContainerObjectChoice (1)	NEVER
DSIWrite	DataSet		1										DataContainerObjectChoice (1)	
DSIRead	DataSet	1											DataContainerObjectChoice (1)	

Table 8-8 — SAL API Actions mapping onto ISO/IEC 7816-15 attributes (2/2)

SAL-API action	Target as defined by 24727-3 chapter 5.3	read(0)	update(1)	execute(2)	delete(3)	attribute(4)	pso_cds(5)	pso_verify(6)	pso_dec(7)	pso_enc(8)	int_auth(9)	ext_auth(10)	7816-15 container	comments
Cryptographic service														
Encipher	DID			1						1			PrivateKeyChoice (2)	(2): other CIO (PrivateKeyChoice, PublicKeyChoice, CertificateChoice, SecretKeyChoice) can be referred to from the classAttribute.iD of the DataContainerObjectChoice thereby attaching a CIO to the DataContainerObjectChoice (e.g attaching a PrivateKey to the Sign Action)
Decipher	DID			1					1				PrivateKeyChoice (2)	
GetRandom	DID			1									CIAInfo.cardflags.prnGeneration value	ALWAYS
Hash	DID			1									CIAInfo.supportedAlgorithms	ALWAYS
Sign	DID			1			1						PrivateKeyChoice(2)	
VerifySignature	DID			1				1				1	PublicKeyChoice, CertificateChoice(2)	
VerifyCertificate	DID			1				1					PublicKeyChoice, CertificateChoice (2)	
Differential-identity service														
DIDList	DID	1											DataContainerObjectChoice (1)	
DIDCreate	C-App		1								1	1	DataContainerObjectChoice (1)	
DIDGet	DID	1											DataContainerObjectChoice (1)	Not similar to DIDList
DIDUpdate	DID		1										DataContainerObjectChoice (1)	Similar to DIDCreate
DIDDelete	DID				1								DataContainerObjectChoice (1)	NEVER
DIDAuthenticate	DID			1							1	1	DataContainerObjectChoice (1)	May combine ext and int authentication depending on the Auth Protocol. **Shall be distinguished from ExecuteAction()**
Authorization service														
ACLList	C-App/ DataSet/ DID	1				1							present in the DataContainerObjectChoice of any Named Service ("Connection", "CardApplication", "NamedData", "Cryptographic", "DifferentialIdentity", "Authorization")	
ACLModify	C-App/ DataSet/ DID		1			1							present in the DataContainerObjectChoice of any Named Service ("Connection", "CardApplication", "NamedData", "Cryptographic", "DifferentialIdentity", "Authorization")	Similar to DIDCreate and DIDUpdate

8.1.3 Card-specific APDU mapping onto ISO/IEC 7816-15 attributes

The Procedural Element may generate the card-specific APDUs according the CIA-based Registry instructions. This translation mandates a means for interoperability based on ISO/IEC 7816-15 with fixed labels assigned to SAL API Actions.

A dedicated ISO/IEC 7816-15 compliant *DataContainerObject* shall be personalized on the card within EF.DCOD to provide reference of the container (EF or DO) hosting the (complete or partial) list of card-specific APDU command headers, as defined in ISO/IEC 7816-4 for expanded format for access mode DOs. This is illustrated in Table 8-9 and Table 8-10.

Table 8-9 — Attributes of DataContainerObject (ISO/IEC 7816-15 DO)

Attributes	Item	Description	
CommonObjectAttributes	Label	An ASCII interoperable descriptive string (see Table 10) of the EF or DO Tag of the container matching SAL API Actions with card-specific APDUs	Mandatory
	accessControlRules	Shall be ALWAYS readable unless otherwise determined by issuer	Mandatory
CommonDataContainer ObjectAttributes	applicationName	name or the registered object identifier for the application to which the data container object belongs	Optional
	iD	Discretionary Identifer (OCTET STRING)	Optional
DataObjectAttributes	Path	Extended Path (according ISO/IEC 7816-15:2008 AM2) to the EF or DO where matching rules between SAL API Actions and a list of [part of] APDU command headers are available. The respective File identifier value or Data Object tag value is up to the issuer.	Mandatory

Table 8-10 — EF or DO identification

Label	File or DO	meaning
"TranslationCode"	FID or Tag	Establishes the link between the ASCII encoded label "TranslationCode" and the file Identifier or DO Tag of the container matching SAL API Actions with card-specific APDUs

8.1.3.1 ASN.1 definition for "TranslationCode" contents :

```
translationCode TranslationCode ::= SEQUENCE{
    action ActionName,
    cmdHeader CmdHeader}

ActionName ::= CHOICE { -- acc. ISO/IEC 24727-3 COR1:2010
apiAccessEntryPoint            APIAccessEntryPointName,
connectionServiceAction        ConnectionServiceActionName,
cardApplicationServiceAction      CardApplicationServiceActionName,
namedDataServiceAction         NamedDataServiceActionName,
cryptographicServiceAction        CryptographicServiceActionName,
differentialIdentityServiceAction    DifferentialIdentityServiceActionName,
authorizationServiceAction        AuthorizationServiceActionName
}
```

```
CmdHeader ::= SET OF SEQUENCE {
        accessModeDOTag OCTET STRING ('81'H..'8F'H),
    accessModeDOValue       OCTET STRING (SIZE(1..4))
}
```

NOTE 1: the tag of access mode DO is from '81' to '8F', then the access mode data element represents a list of possible combinations (max = 4)of values of the four bytes CLA, INS, P1 and P2 in the command header.

NOTE 2: the entire corresponding ASN.1 module is defined in Annex G in the present document.

8.1.3.2 Example of SAL API action to card-specifc cmd translation instructions

This example matches the SAL API "DSIList" with card-specific APDU command SELECT DATA. Accordingly, the EF file referred to from EF.DCOD under the label "TranslationCode" contains the following :

Hex-string coding	**meaning**
30 16 1A 07 4453494C697374 31 0B 30 09 04 01 8F 04 04 00A5F002	TranslationCode CmdHeader SEQUENCE actionName "**DSIList**" VisibleString[UNIVERSAL 26] CmdHeader SET OF SEQUENCE cmdHeader accessModeDOTag OCTET STRING accessModeDOValue OCTET STRING for : CLA='00' INS='A5' P1='F0' P2='02' meaning : **SELECT DATA** with DIR function and selection of the first occurrence of the DO setting the current template (acc. ISO/IEC 7816-4)

in this example, the PE is assumed to adapt the card-specific APDU command data field to fit with the purpose of the DSIList action : for instance the data field of SELECT DATA can bring a *general reference template* pointing the the actual DF or EF or ADF or DO representing the current DataSet context.

8.1.4 ISO/IEC 24727-3 data structures storage onto the card

According to ISO/IEC 24727-2, the **CardApplicationServiceDescription** data object of a card-application may be nested in the card-application ACD.

From the ACD, the mapping of DataSets and Services may make reference to Differential-Identities that are either encoded as appropriate Information Objects within the ACD or within the relevant EFs of a DF.CIA application present on the card. As example, the authId attributes from the access rules of a Named Service may point to a CommonKeyAttributes.iD of a Secret Key within the EF.SKD associated to a cryptographic information application.

Each CIOChoice encoded in the CardApplicationServiceDescription can be either encoded as Path or Object.

The figure below shows the relationships and indirections between these data structures and Information Objects.

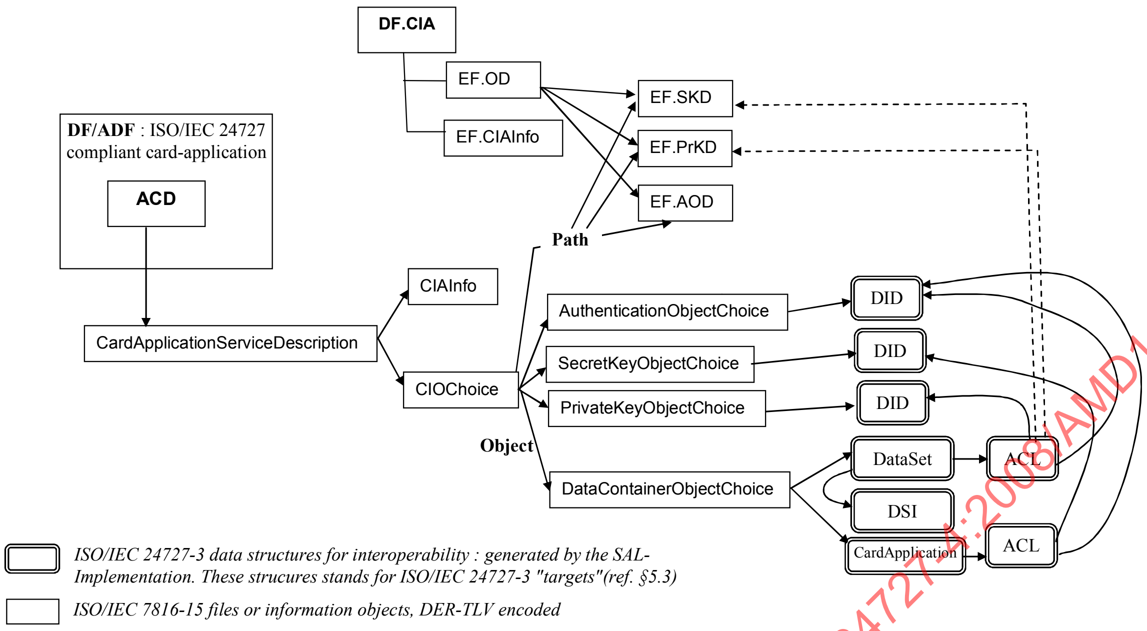
DF.CIA
EF.OD
EF.CIAInfo
EF.SKD
EF.PrKD
EF.AOD
DF/ADF : ISO/IEC 24727 compliant card-application
ACD
Path
CardApplicationServiceDescription
CIAInfo
CIOChoice
AuthenticationObjectChoice
SecretKeyObjectChoice
PrivateKeyObjectChoice
DataContainerObjectChoice
Object
DID
DID
DID
DataSet
DSI
CardApplication
ACL
ACL
ISO/IEC 24727-3 data structures for interoperability : generated by the SAL-Implementation. These strucures stands for ISO/IEC 24727-3 "targets"(ref. §5.3)
ISO/IEC 7816-15 files or information objects, DER-TLV encoded

Figure 8-2 — Relationships between ISO/IEC 7816-15 information objects and ISO/IEC 24727-3 data structures

Page 80

Insert the following new annexes after Annex C:

Annex D
(informative)

Enhanced Use of Procedural Elements

D.1 Background

The objective of ISO/IEC 24727 is to provide interoperability among diverse applications that make use of tokens in the form of ICCs. The services provided by such tokens are termed Identity, Authentication, Signature (IAS) services. Such services are central to the establishment of a trust environment to support distributed, digital interactions. In particular, tokens are used to convey identity information, authentication facilities and signature (non-repudiation) facilities. Within most existing systems in this domain, the top level client-applications are very tightly linked to specific tokens. Through facilities defined in ISO/IEC 24727 it is possible for diverse client-applications to obtain IAS services from diverse tokens.

The operational environment for token based systems in which ISO/IEC 24727 was developed tended to be split along two distinct approaches to achieving standardization from which interoperability could emanate: (a) a standard high-level API that could be used by client applications through a stack that went all the way to the card (SILO systems), and (b) a standard (APDU) command set which would allow a variety of tokens to be utilized as long as they each supported the standard command set; again, primarily SILO systems. The result has been that ISO/IEC 24727 establishes two distinct interfaces: ISO/IEC 24727-3 API and ISO/IEC 24727-2 Generic Command Interface. Each of these interfaces can provide access to distinct software layers that lay between a client-application and an ICC. With several “interface points” available within a stack, it becomes possible to allow middleware to function as a “switch” that connects multiple client-applications to a variety of tokens.

It is assumed that an actual token may not support the standard APDU set specified in ISO/IEC 24727-2. To accommodate such tokens, a procedural element is defined to be included within a software layer accessed through the GCI interface. However, among the stack configurations defined in this part is an ICC Resident Stack. For a token that implements this stack, the only APDU that may be required of the token is the ENVELOPE APDU. This APDU allows an ASN.1 representation of the ISO/IEC 24727-3 API to be conveyed from a client-application onto the token. In essence, the ENVELOPE APDU can be used to convey the ISO/IEC 24727-3 API in the form of ISO/IEC 7816-4 Data Objects (DOs). This facility allows for some of the stack options to avoid the necessity of a strict ISO/IEC 24727-2 software layer. In the following clauses of this annex, some of the implications of this facility are explored.

An illustrative specification of an interface for an ISO/IEC 7816-4 data store is found in Annex E and is designated as the P15API.

D.1.2 Derived Stack Configurations

In order for ISO/IEC 24727 to become more interoperable, efficient and attractive to implement the following is approach is suggested:

a) Define a stack model that can cater for all implementations i.e. the Fully Interoperable Network Stack. This way there is a specification for implementation of a fully interoperable stack model;

b) In a future amendment or revision of ISO/IEC 24727, deprecate and subsequently remove ISO/IEC 24727-2 specified APDUs and abstract the card specific translation for data management, authentication, application management, etc… away from ISO/IEC 24727. E.g. the procedural element can translate ISO/IEC 24727 card management commands to smartcard specific commands, such as those defined by GlobalPlatform;

c) Utilize Procedural Element (PE) functionality directly in the Service Access Layer implementation;

d) Have this PE consume the CIA/P15 if required;

e) The SAL then performs two types of marshalling:

- DER TLV, which is wrapped in an ENVELOPE APDU and sent to the PE; or
- XML that is sent to the PE;

D.1.3 Suggested Procedural Element functionality

The enhanced use of the procedural element component expressed in D1.2 is illustrated in a process flow form in Figure D-1. The use case is the following:

- The distant machine implements the SAL implementation and the client application;
- The local machine handles the card. It also implements the IFD layer, and SAL Lite when present, which are accessed through the IFD API.

Once the SAL implementation has recovered all necessary data (registry, addressing information, etc.), its function is to distinguish two situations, then to perform the appropriate procedure:

- The API call is eligible for the use of SAL API Lite. The SAL implementation transmits the call to the local machine and SAL API Lite, under SOAP encapsulation.
- The API call is not eligible for the use of SAL API Lite. The SAL implementation transmits the call to the PE, which builds the appropriate APDU to be transmitted to the local machine and the IFD layer under SOAP encapsulation.

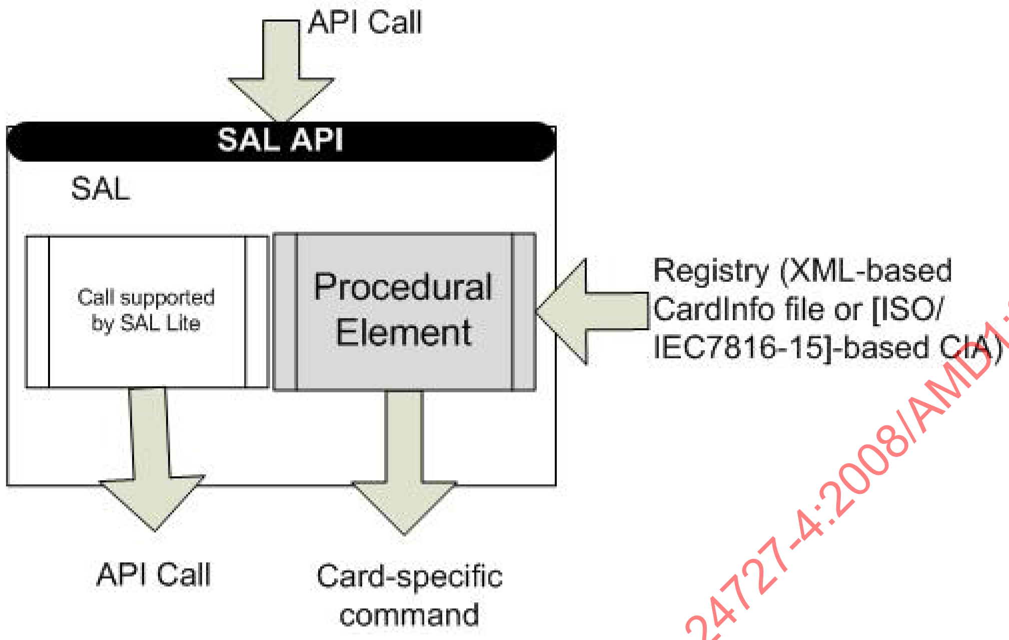
API Call
SAL API
SAL
Call supported by SAL Lite
Procedural Element
Registry (XML-based CardInfo file or [ISO/ IEC7816-15]-based CIA)
API Call
Card-specific command

Figure D1 – Procedural Element functionality

The Procedural Element constructs the card specific commands e.g. APDUs. APDUs no longer need to be specified in ISO/IEC 24727 as these are now handled by the Procedural Element and are smartcard implementation specific. This ensures that all smartcard implementations can become interoperable no matter what APDU commands (or other commands) they support. This is also true for any smartcard application management commands that are supported by the various vendors. Hence there is no requirement to specify these smartcard application management commands in ISO/IEC 24727. All that is required are the existing high level smartcard management commands as specified by ISO/IEC 24727-3.

It is up to each Procedural Element to perform the correct translation from a SAL action into one or more commands that are expected by the chip.

Note: This section only describes the action sent to the chip, but the reverse is applicable for responses from the chip.

D.2 Full Network Stack configuration

A full network stack configuration allows a client-application to be separated across a network from the ICC access point represented by the IFD. This configuration makes use of the proxy-agent architecture specified elsewhere in this part.

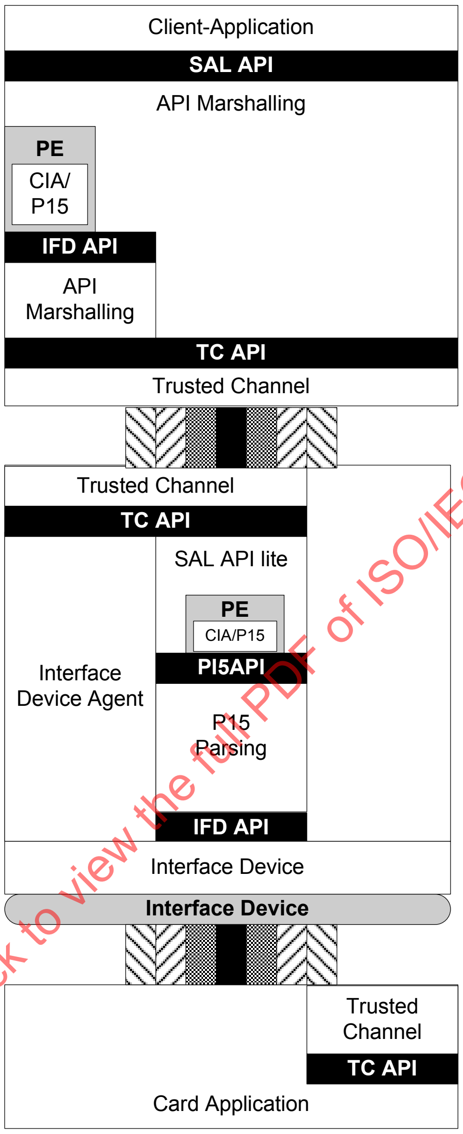

Figure D2 – Fully interoperable Network Stack

Figure D2 illustrates a Full Network Stack configuration that makes use of a procedural element from within a Service Access Layer. This procedural element can access an ISO/IEC 7826-15 Registry in order to translate directly from a DIR-TLV representation of the ISO/IEC 24727-3 API and card specific APDUs.

D.2.1 Scenario 1 – Interfacing with existing smart card applications

f) The Client-Application performs actions on the SAL API, which is a language specific implementation of the SAL API e.g. C, Java, SOAP, etc…

1) The API call is marshaled into DER TLV or XML and sent to Procedural Element (PE) using XML or the ENVELOPE APDU.

2) The PE maps the SAL attributes to on-card attributes using the Cryptographic Information Application (CIA or P15) and then translates the SAL command into card specific APDUs.

3) The request APDU is sent via the IFD API, TC and Interface Device to the Card Application.

4) The response APDU is returned to the PE via the Interface Device, TC and IFD API.

5) The PE either sends another APDU or formats the DER TLV encoded response contained in the response APDU (to the ENVELOPE APDU) to be marshalled for return by the SAL API to the Client-Application.

D.2.2 Scenario 2 – Interfacing on-card applications with DER-TLV encoded SAL

g) The Client-Application performs actions on the SAL API, which is a language specific implementation of the SAL API e.g. C, Java, SOAP, etc…

h) The API call is marshalled into DER TLV and packaged in the ENVELOPE APDU.

i) The request (ENVELOPE) APDU is sent via the IFD API, TC and Interface Device to the Card Application.

j) The response APDU is returned via the Interface Device, TC and IFD API to be marshalled for return by the SAL API to the Client-Application.

D.2.3 Scenario 3 – Interfacing with XML encoded SAL on-card applications

k) The Client-Application performs actions on the SAL API, which is a language specific implementation of the SAL API e.g. C, Java, SOAP, etc…

l) The API call is marshalled into XML (e.g. SOAP) and is sent directly to the smartcard via the TC layer and Interface Device.

m) The smartcard responds with XML (e.g. SOAP) via the Interface Device and TC layer to be marshalled for return by the SAL API to the Client-Application.

D.2.4 Scenario 4 – Interfacing with existing smart card applications using SAL API Lite

n) Secure messaging is specified in Clause 5.7.4.

o)

D.3 Amendments to existing stack models

Use of the more powerful procedural element that is enabled through the use of an ISO/IEC 7816-15 based registry allows the specialization of the various stack configurations described elsewhere in this part. The following sub-clauses examine these specializations where applicable.

D.3.1 Full-network-stack

The purpose of the Full Network Stack is to decouple the client application from the ICC access point and allow each to be present at different points within a complex network.

D.3.1.1 Current configuration

Figure D3 illustrates a very general configuration of the Full Network Stack. It provides for the client-application, the SAL, the GCI and the IFD to be supported on different computer platforms in the network. The greatest utility of this configuration is anticipated to be for testing, but other operational scenarios can be anticipated.

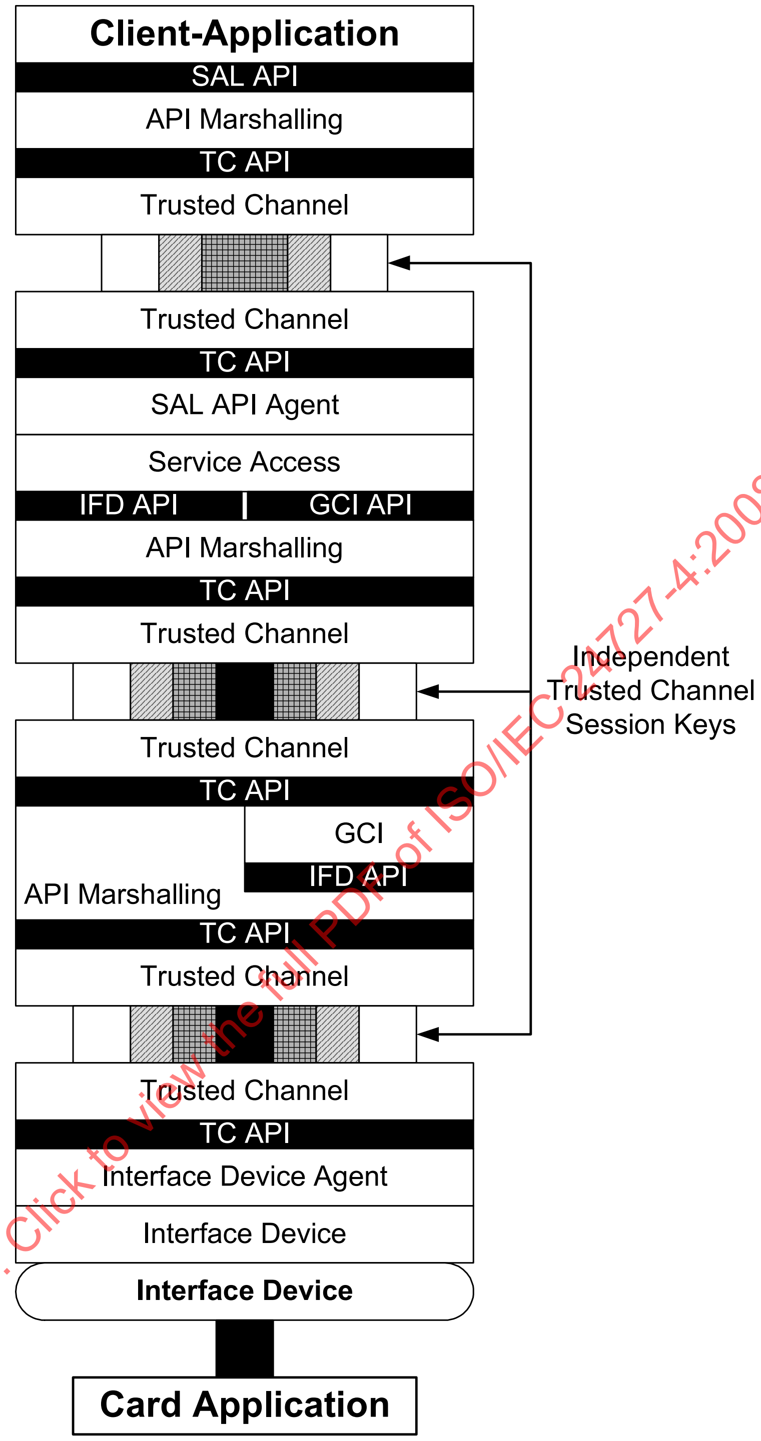

Figure D3 – Current Full Network Stack

D.3.1.2 Proposed configuration

Figure D4 illustrates a slightly more restrictive stack which allows much the same dispersion of the stack components across the network. It also minimizes the GSI layer, allowing for its potential deprecation in the future.

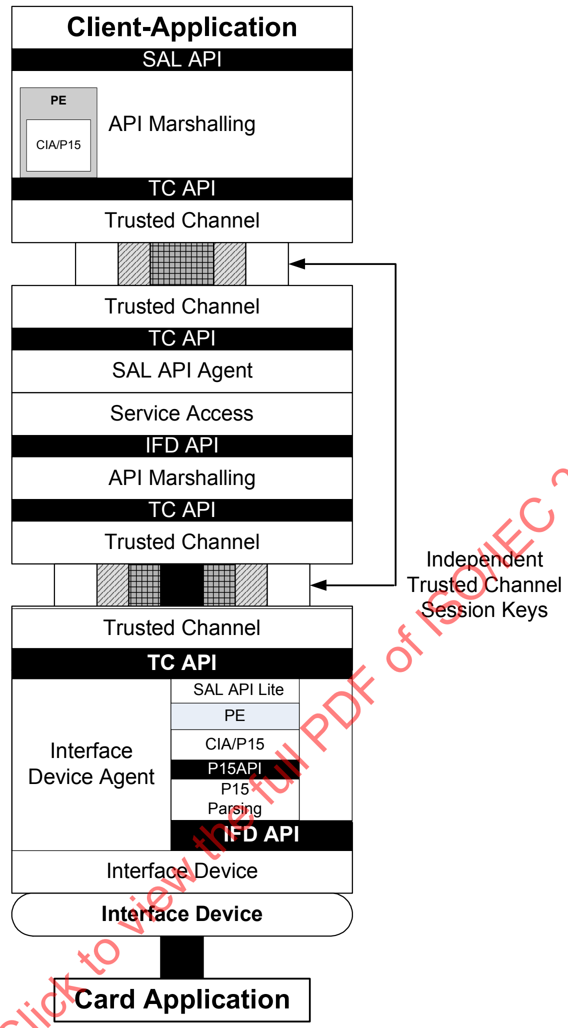

Figure D4 – Proposed Full Network Stack

As noted, this configuration restricts the Full Network Stack slightly by assuming the “GCI equivalent” layer comprised of a more focussed procedural element to be present on the same platform as the IFD. This establishes a compact component that has many of the characteristics of a “terminal” in legacy configurations.

D.3.2 Loyal-stack

A Loyal Stack is representative of a prevalent configuration for legacy systems. In this configuration, the entire stack from the client-application to the IFD is assumed to be present on a single computer platform. This configuration can utilize intrinsic security levels of a single platform as opposed to the more complex mechanisms necessary to achieve equivalent security levels across multiple platforms.

D.3.2.1 Current configuration

Figure D5 illustrates the basic Loyal Stack configuration.

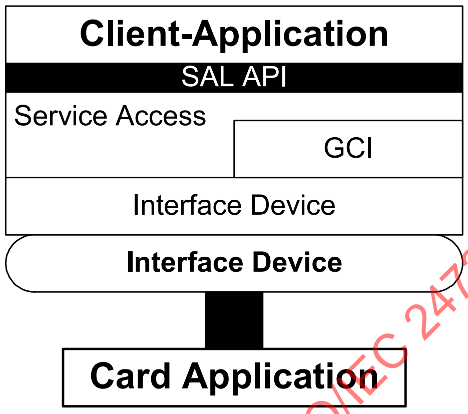

Figure D5 – Current Loyal Stack

D.3.2.2 Proposed configuration

While architecturally, making use of an enhanced procedural element doesn't change the basic Loyal Stack configuration, as illustrated in Figure D6 a potentially more flexible stack is made possible through its use.

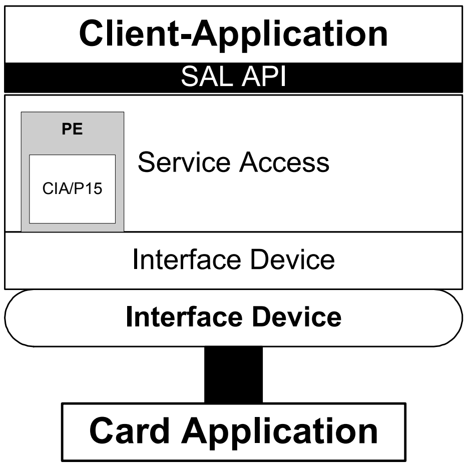

Figure D6 – Proposed Loyal Stack

In this stack configuration, the GCI is never surfaced. This makes its deprecation simpler, and yet the enhanced procedural element allows greater flexibility in supporting a variety of ICCs.

D.3.3 Opaque-ICC-stack

The purpose of the Opaque ICC Stack is to segregate the stack along the lines of a client-application on one platform and all the ICC components on a different platform. This latter platform has many of the characteristics of a “terminal” in legacy systems.

D.3.3.1 Current configuration

Figure D7 illustrates the classic Opaque ICC Stack.

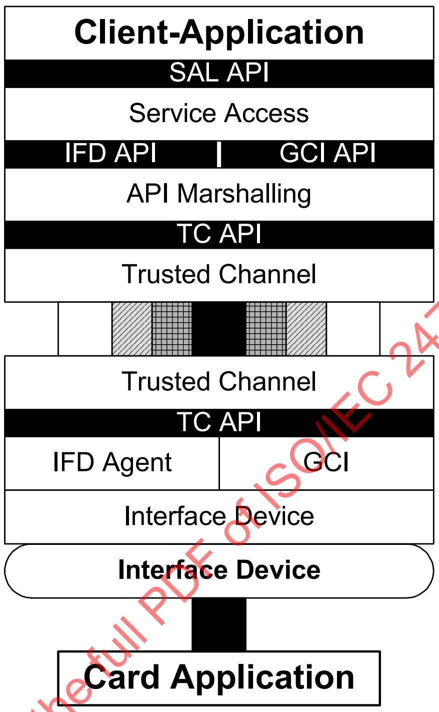

Figure D7 – Current Opaque-ICC Stack

D.3.3.2 Proposed configuration

Figure D8 illustrates a specialization of the Opaque ICC Stack.

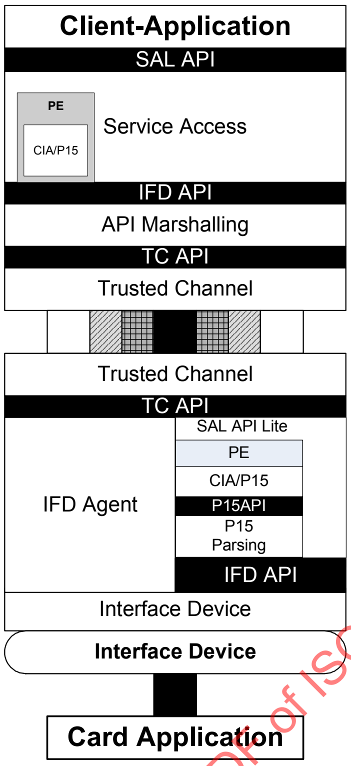

Figure D8 – Proposed Opaque-ICC Stack

This specialized configuration makes use of the enhanced procedural element accessed through the SAL API Lite to minimize the visibility of the GCI and yet provide a configuration able to deal with a wider variety of ICCs.

D.3.4 Remote-loyal-stack

The Remote Loyal Stack allows the bifurcation of the stack such that virtually all of the stack operation is found on a different platform from the client-application. This allows a very light-weight client-application configuration. It also allows the ICC access platform to have more stringent physical security.

D.3.4.1 Current configuration

Figure D9 illustrates the most common variant of the Remote Loyal Stack configuration.

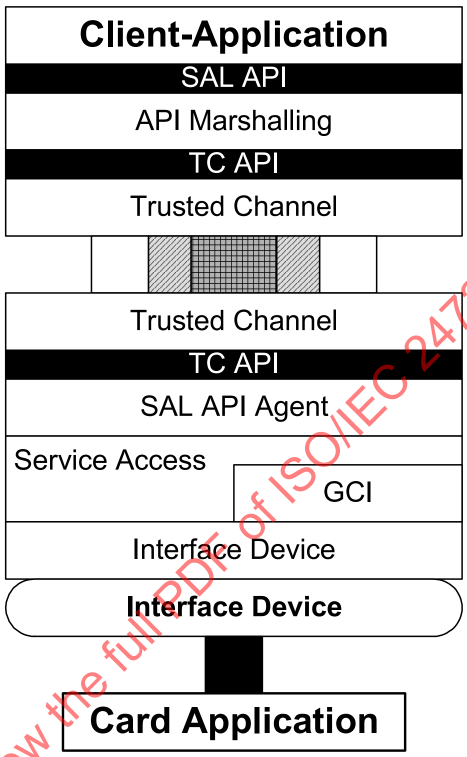

Figure D9 – Current Remote-loyal Stack

D.3.4.2 Proposed configuration

As illustrated in Figure D10, the use of an enhanced procedural element does not appreciably change the Remote Loyal Stack configuration. It does, however, offer the prospect of dealing with a wider range of ICCs.

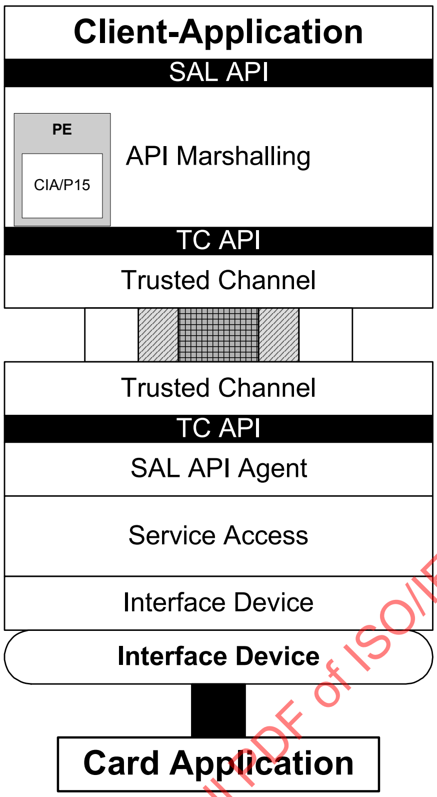

Figure D10 – Proposed Remote-loyal Stack

D.3.5 ICC-resident-stack

There is no off-card Service Access Layer for an ICC-resident-stack configuration. The entire Service Access Layer is present within the Card-Application. The purpose of the off-card stack elements is to transfer the formal language representation of the ISO/IEC 24727-3 API directly to the card-application.

D.3.6 Remote-ICC-stack

The Remote ICC Stack provides a configuration in which the ICC access point is segregated from the rest of the stack. This allows for a very light-weight access point.

D.3.6.1 Current configuration

As illustrated in Figure D11, most the components, and hence the computation complexity of this stack is found on a common platform with the client-application. One use for this is to allow a centralized platform for the client-application and many, distributed ICC access points.

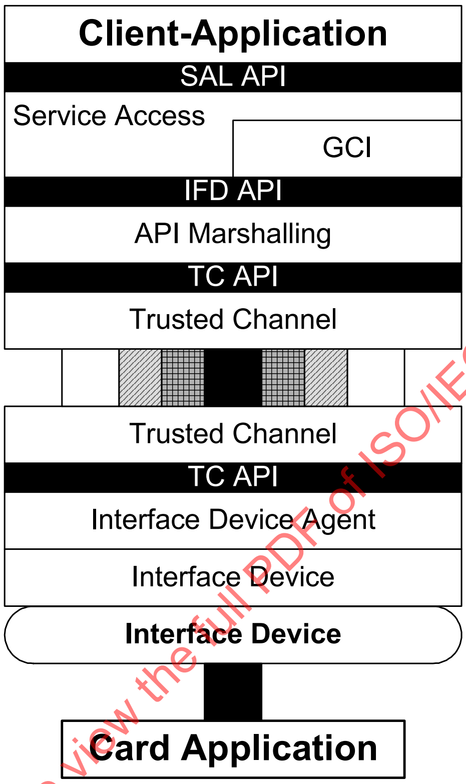

Figure D11 – Current Remote-ICC Stack

D.3.6.2 Proposed configuration

As illustrated in Figure D12, by distributing the SAL API Lite component to the ICC access point, this new configuration offers greater power and flexibility in dealing with a variety of ICCs. This does not necessarily address the same operational characteristics as the basic configuration.

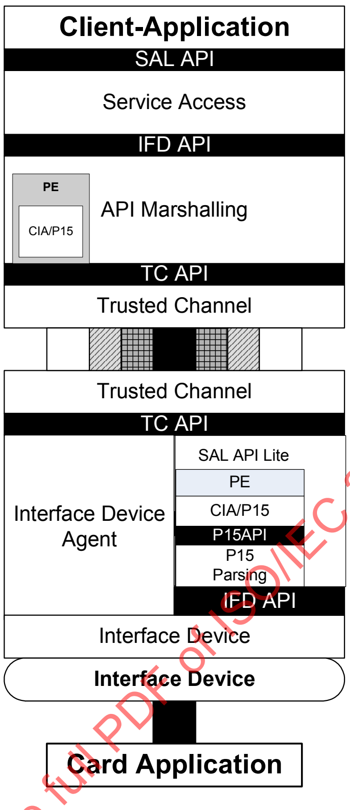

Figure D12 – Proposed Remote-ICC Stack

D.4 Stack Enhancement

The ultimate deprecation and subsequent removal of ISO/IEC 24727-2 APDUs, and the relocation of specialized procedural elements (to ISO/IEC 24727-3), would allow the simplification of ISO/IEC 24727 and the associated stack model configurations without the reduction of functionality. This in turn would simplify ISO/IEC 24727 implementations by reducing the need to cater for a myriad of variations of APDUs in order to be interoperable.

APDU translation would be performed in one software layer instead of two layers. ISO/IEC 24727 would not need to specify any APDUs (apart from ENVELOPE) in order to be compatible with current ICC implementations. This would be achieved by translating the SAL command into card specific APDUs (in the SAL layer) that the chip expects. The SAL layer would perform this translation by using a procedural element. The procedural element would require the CIA to map between on-card and off-card concepts.

Annex E
(informative)

API for ISO/IEC 7816-15 data structures handling

This informative annex describes how the SAL API Lite implementation is able to gain access to the ISO/IEC 7816-15 data structures representing a Registry hosted on a card. This annex defines an API (called P15API) for efficient data handling of the ISO/IEC 7816-15 data structures. This API is defined regardless of card-application implementation. The acronym P15 is a generic reference to standards that derive from PKCS 15 such as ISO/IEC 7816-15.

E.1 C-language Binding for the P15-API

This section describes basic data types, structures and friendly names associated to card content values.

E.1.1 P15RESULT

P15RESULT is the type used for all P15-API functions return codes.

```
typedef unsigned long P15RESULT;
```

Possible values:

P15RESULT_OK	*0x00000000*	*Success*
P15RESULT_ALREADY_INITIALIZED	*0x80000001*	*API Cannot be initialized twice*
P15RESULT_NOT_INITIALIZED	*0x80000002*	*API shall be initialized before any other function is called*
P15RESULT_FUNCTION_FAILED	*0x80000002*	*Internal error while processing the request*
P15RESULT_NOT_ENOUGH_SPACE	*0x80000003*	*Unable to process request due to insufficient provided space*
P15RESULT_NO_CARD	*0x80000004*	*No card present in the reader*

E.1.2 P15CONTEXT

P15CONTEXT is a handle to an initialized P15 API context.

```
typedef unsigned long P15CONTEXT;
```

E.1.3 P15INFO

P15INFO is a structure providing information about P15 API.

```
typedef struct {
       P15VERSION P15APIVersion;
} CK_INFO;
```

E.1.4 P15VERSION

P15VERSION is a structure representing an object version.

```
typedef struct {
        BYTE major;
        BYTE minor;
} P15VERSION;
```

E.1.5 CioChoice

CioChoice is an enumeration of possible P15 object classes. This is particularly useful when parsing an application’s object directory structure (EF.OD). Refer to ISO/IEC 7816-15 *CIOChoice* for a complete description.

```
enum   CioChoice{
   ctPrivateKeys      = 0,
      ctPublicKeys       = 1,
      ctTrustedPublicKeys    = 2,
      ctSecretKeys       = 3,
      ctCertificates     = 4,
      ctTrustedCertificates  = 5,
      ctUsefulCertificates   = 6,
      ctDataContObjects      = 7,
      ctAuthObjects      = 8
};
```

E.1.6 CommonObjectFlags

CommonObjectFlags is an enumeration of possible P15 API object flags, describing object access scope. Refer to ISO/IEC 7816-15 *CommonObjectFlags* for a complete description.

```
enum CommonObjectFlags{
       flagPrivate      = 0x01,
       flagModifiable   = 0x02,
       flagInternal     = 0x04
};
```

E.1.7 SecurityEnvironmentInfo

SecurityEnvironmentInfo is a structure describing a security environment. Refer to ISO/IEC 7816-15 *`SecurityEnvironmentInfo`* for a complete description.

```
struct SecurityEnvironmentInfo {
       int se;
       std::string owner;
       CBlob aid;
};
```

E.1.8 AlgorithmInfo

AlgorithmInfo is a structure describing a supported algorithm. Refer to ISO/IEC 7816-15 *AlgorithmInfo* for a complete description.

```
struct AlgorithmInfo {
       int reference;
       int algorithm;
       CBlob parameters;
       int supportedOperations;
       std::string objId;
       int algRef;
};
```

E.1.9 PasswordType

passwordType is an enumeration of possible password authentication types. Refer to ISO/IEC 7816-15 *PasswordType* for a complete description.

```
enum PasswordType {
       typeBcd           = 0,
       typeAsciiNumeric  = 1,
       typeUtf8          = 2,
       typeHalfNibbleBcd = 3,
       typeIso9564_1     = 4
};
```

E.1.10 Validity

Validity is a structure containing certificate validity period.

```
struct Validity {
        std::string notBefore;
        std::string notAfter;
        };
```

E.1.11 ObjectValueType

ObjectValueType is an enumeration providing possible types of object value storage location (direct or indirect via path, url…). Refer to ISO/IEC 7816-15 AM2:2008 (ref. extended Path) for a complete description.

```
enum ObjectValueType {
       ovtNone,
       ovtDirect,
       ovtPath,
       ovtUrlPrintableString,
       ovtUrlIA5String,
       ovtUrlWithDigest
};
```

E.1.12 FileType

FileType is an enumeration providing possible card file types. Refer to ISO/IEC 7816-4 for a complete description.

```
enum FileType {
       ftNone,
       ftApplication,
       ftDedicated,
       ftTransparent,
       ftLinearFixed,
       ftLinearVariable,
       ftCyclic,
       ftTLV
};
```

E.1.13 FileState

FileState is an enumeration providing possible card file states. Refer to ISO/IEC 7816-4 for a complete description.

```
enum FileState {
       fsNone,
       fsCreation,
       fsInitialisation,
       fsOperationalActivated,
       fsOperationalDeactivated,
       fsTermination,
       fsProprietary
};
```

E.1.14 IdType

IdType is an enumeration providing possible credential identifiers. Refer to ISO/IEC 7816-15 *KeyIdentifiers* for a complete description.

```
enum IdType {
       idNone                        = 0,
       idIssuerAndSerialNumber       = 1,
       idSubjectKeyId                = 2,
       idIssuerAndSerialNumberHash = 3,
       idSubjectKeyHash              = 4,
       idIssuerKeyHash               = 5,
       idIssuerNameHash              = 6,
       idSubjectNameHash             = 7,
       idPgp2KeyId                   = 8,
       idOpenPGPKeyId                = 9
};
```

E.1.15 AccessModes

AccessModes is an enumeration providing possible object access modes flags with the extension according ISO/IEC 7816-15:2004/AM2. Refer to ISO/IEC 7816-15 *AccessMode* for a complete description.

```
enum AccessModes {
       flagRead    = 0x01,
       flagUpdate  = 0x02,
       flagExecute = 0x04,
       flagDelete  = 0x08,
       flagAttribute = 0x10,
       flagPsoCds = 0x14,
       flagPsoVerify = 0x24,
       flagPsoDec = 0x44,
       flagPsoEnc = 0x84,
       flagIntAuth = 0xC4,
       flagExtAuth = 0xA4
};
```

E.1.16 Operations

Operations is an enumeration providing possible key operations. Refer to ISO/IEC 7816-15 *Operations* for a complete description.

```
enum Operations {
       flagOperComputeChecksum    = 0x01,
       flagOperComputeSignature   = 0x02,
       flagOperVerifyChecksum     = 0x04,
       flagOperVerifySignature    = 0x08,
       flagOperEncipher           = 0x10,
       flagOperDecipher           = 0x20,
       flagOperHash               = 0x40,
       flagOperGenerateKey        = 0x80
};
```

E.1.17 ContextTag

ContextTag is an enumeration providing possible object references in **RecordInfo**. Refer to ISO/IEC 7816-15 *RecordInfo* for a complete description.

```
enum ContextTag {
       oDRecordLength  = 0,
       prKDRecordLength    = 1,
       puKDRecordLength    = 2,
       sKDRecordLength = 3,
       cDRecordLength  = 4,
       dCODRecordLength    = 5,
       aODRecordLength = 6
}
```

E.1.17.1 P15Finalize

P15Finalize is called to indicate that an application is finished with the P15 API. It shall be the last P15 API function call made by an application.

```
P15RESULT P15Finalize(P15CONTEXT P15Context)
```

P15Context [in] is the P15 context received on successful call to **P15Initialize**.

Return values:

P15RESULT_NOT_INITIALIZED, P15RESULT_FUNCTION_FAILED, P15RESULT_OK.

E.1.17.2 P15GetInfo – Version information

P15GetInfo returns information about P15 API.

```
P15RESULT P15GetInfo(P15INFO *pP15Info)
```

pP15Info [out] is a structure to receive P15 API information.

Return values:

`P15RESULT_FUNCTION_FAILED, P15RESULT_OK.`

E.1.18 Reader and Card management Functions

E.1.18.1 P15ListReaders

P15ListReaders returns a list of connected card readers.

```
P15RESULT P15ListReaders(P15CONTEXT P15Context, LPSTR mszReaders, LPDWORD
pcchReaders)
```

P15Context [in] is the P15 context received on successful call to **P15Initialize**.
mszReaders [out] Buffer to receive list of connected card readers. Each reader name is finished by NULL character and list is finished by an additional NULL character. If provided value is NULL, **P15ListReaders** ignores the buffer length supplied in *pcchReaders*, writes the length of the buffer that would have been returned if this parameter had not been NULL to *pcchReaders*, and returns a success code.
pcchReaders [in,out] On input contains size of allocated *mszReaders* buffer. On successful execution, this parameter receives the actual length of the *mszReaders* list.

Return values:

```
P15RESULT_NOT_INITIALIZED, P15RESULT_NOT_ENOUGH_SPACE, P15RESULT_FUNCTION_FAILED,
P15RESULT_OK.
```

E.1.18.2 SecurityConditionType

SecurityConditionType is an enumeration providing possible security condition types. Refer to ISO/IEC 7816-15 *SecurityCondition* for a complete description.

```
enum SecurityConditionType {
        sctNone,
        sctAlways,
        sctAuthId,
        sctAuthReference,
        sctNot,
        sctAnd,
        sctOr
}
```

E.2 Interface functions

the functions prototypes are defined with C++ language convention.

E.2.1 General Purposes Functions

E.2.1.1 P15Initialize

P15Initialize initializes P15 library. An application shall call this function prior to any other function, and call **P15Finalize** function when it's done using P15 API.

```
P15RESULT P15Initialize(P15CONTEXT *pP15Context)
```

pP15Context [out] receives an P15 context handle on successful execution.

Return values:

`P15RESULT_ALREADY_INITIALIZED, P15RESULT_FUNCTION_FAILED, P15RESULT_OK.`

E.2.2.2 P15Connect

P15Connect tries to open a session with a card inserted in the specified reader.

```
P15RESULT P15Connect(P15CONTEXT P15Context, LPSTR szReader, CP15Card **ppP15Card)
```

P15Context [in] is the P15 context received on successful call to **P15Initialize**.
szReader [in] is the reader name P15 API shall attempt to connect to.
ppP15Card [out] On successful execution, contains a valid CP15Card object reference.

Return values:

```
P15RESULT_NOT_INITIALIZED, P15RESULT_NO_CARD, P15RESULT_FUNCTION_FAILED,
P15RESULT_OK.
```

NOTE It is possible to create a CP15Card object directly from an application provided PS/SC SCARDHANDLE, previously established in the application context.

NOTE It's application's responsibility to delete returned CP15Card object.

E.3 Objects

E.3.1 Basic objects

E.3.1.2 CP15Card

CP15Card object provides basic ISO/IEC 7816-4 functions for transparent card access and card identification.

```
CP15Card
      CP15Card(SCARDHANDLE hCard)
      void verify (unsigned char p2, CBlob const &data)
      CBlob atr()
      void reset()
      void beginTransaction()
      void endTransaction()
      void sendAPDU (unsigned char cla, unsigned char ins,
             unsigned char p1, unsigned char p2
             CBlob const & data, unsigned long ne,
             CBlob &response, unsigned short &sw12)
```

```
CP15Card(SCARDHANDLE hCard)
```

Applications may use **P15Connect** function to get a new **CP15Card** object connected to a valid card, or may use this constructor, providing a valid PC/SC SCARDHANDLE previously established by the application. This handle shall be closed automatically when the CP15Card object is deleted.

```
void verify (unsigned char p2, CBlob const &data)
```

Allows verifying provided reference data *data* with the one stored in the card and identified by *p2*. On successful verification the associated security level is granted and associated operations are allowed.

```
CBlob atr()
```

Use this function to get connected card ATR (Answer To Reset), allowing identification of card type and model.

```
void reset()
```

Call this function to request card re-initialization.

```
void beginTransaction()
```

Call this function to request exclusive card access mode and guarantee sequentiality of following card commands. Other applications are locked until a call to **endTransaction** is performed.

```
void endTransaction()
```

Call this function to end exclusive card access mode established with **begintransaction** and release other applications.

```
void sendAPDU (  unsigned char cla, unsigned char ins,
        unsigned char p1, unsigned char p2
        CBlob const & data, unsigned long ne,
        CBlob &response, unsigned short &sw12)
```

This command allows sending any APDU command to the card.

E.3.1.2 CP15exception

CP15Exception class encapsulates and describes errors occurred during P15 API functions execution.

```
CP15Exception
       CP15Exception(unsigned long errorCode)
       void raise()
       std::string descriptionA() const
       std::wstring descriptionW() const
       unsigned long error() const
```

```
CP15Exception(unsigned long errorCode)
```
Constructor.

```
void raise()
```
Raise exception.

```
std::string descriptionA() const
```
Get exception description.

```
std::wstring descriptionW() const
```
Get exception description as wide chars.

```
unsigned long error() const
```
Get exception error code.

E.3.1.3 CCiaPath

CCiaPath is an object representation of a CIA object path. This class shall be used to describe where an object can be found in the CIA application (e.g. in which file, at what index and the length allocated to the object). The Path definition shall conform to ISO/IEC 7816-15:2004/AM2 (extended Path)

```
CCiaPath class
        CCiaPath ()
        CCiaPath (CBlob const &blob)
        CCiaPath(CCiaPath const & that)
        CCiaPath & operator=(CCiaPath const & that)
        void setEfidOrPath(CBlob const & efidOrPath)
        void setIndex(int index)
        void setLength(int length)
        CBlob efidOrPath ()
        int index()
        int length()
```

`CCiaPath ()`

This constructor allows creating an empty object. Use **operator =** or **setEfidOrPath** to initialize object path.

`CCiaPath (CBlob const &blob)`

This constructor allows creating an object representation of a file path provided as a **CBlob**. When specifying a direct path, applications may found useful to use _HB macro to directly provide a string path. (e.g. **_HB("2F00")**)

`CCiaPath (CCiaPath const & that)`

This is a copy constructor.

`CCiaPath & operator=(CCiaPath const & that)`

This is a assignment operator to initialize an object from another.

`void setEfidOrPath(CBlob const & efidOrPath)`

Call this function to initialize or change the path this object points to. See **CCiaPath (CBlob const &blob)** for parameter description.

`void setIndex(int index)`

Call this function to initialize or change the offset (or record number) defining a specific starting location in the referenced file. By default the starting offset is set to 0

`void setLength(CBlob const & efidOrPath)`

Call this function to initialize or change the length defining a specific part in the referenced file, starting from the defined index (if any).

`CBlob efidOrPath ()`

Call this function to get the path this object points to.

`int index()`

Call this function to get the specific starting location in the referenced file this object points to.

`int length()`

Call this function to get the location length in the referenced file this object points to.

E.3.1.4 CBlob

CBlob is an object representation of a binary buffer, broadly used by all other classes. Typical application usually manipulates a lot of **CBlob** objects, while it only creates a few on its own. Nevertheless, **CBlob** creation is useful -almost only- for file paths and applications aids, then **_HB** macro may be of interest. Applications willing to display **CBlob** object content may use **_HB** and **AsString** macros.

```
CBlob
        CBlob()
        CBlob(const size_t nCount)
        CBlob(size_t nCount, unsigned char const &val)
        CBlob(unsigned char *pBuffer, size_t nCount)
        CBlob & operator= (unsigned char const &rt)
        unsigned char &    operator[] (size_t nIdx)
        unsigned char &    operator * ()
        bool    operator== (CBlob const &rother) const
        iterator    begin ()
        iterator    end ()
        size_t  size () const
        size_t  length () const
```

`CBlob()`

Default constructor creating an empty blob.

`CBlob(const size_t nCount)`

Creates an empty blob with pre-allocated *nCount* size.

`CBlob(size_t nCount, unsigned char const &val)`

Creates an initialized blob with provided *val* value.

`CBlob(unsigned char *pBuffer, size_t nCount)`

Creates an initialized blob with *nCount* bytes at *pBuffer* address.

`CBlob & operator= (unsigned char const &rt)`

Assignment operator.

`unsigned char &   operator[] (size_t nIdx)`

Array operator, allowing to get byte at index *nIdx*.

`unsigned char &   operator * ()`

Returns blob content.

`bool    operator== (CBlob const &rother) const`

Blob comparison operator.

`iterator   begin ()`

Function returning a blob iterator.

`iterator   end ()`

Function returning a blob iterator pointing to the end of the blob content.

```
size_t  size  ()  const
```

Returns the size of the allocated blob in bytes.

```
size_t  length  ()  const
```

Returns the actual blob length in bytes.

E.3.1.5 CCioSecurityCondition

CCioSecurityCondition is an object representation of card access security condition. Refer to ISO/IEC 7816-15 `SecurityCondition` for a complete description.

```
CCioSecurityCondition
      CCioSecurityCondition ()
      CCioSecurityCondition (CBlob const &blob)
      CCioSecurityCondition (CCioSecurityCondition const &that)
      CCioSecurityCondition & operator= (CCioSecurityCondition const &that)
      SecurityConditionType type() const
      CBlob authId() const
      CCioAuthReference authReference() const
      CCioSecurityCondition not() const
      std::vector< CCioSecurityCondition > and()
      std::vector< CCioSecurityCondition > or()
```

`CCioSecurityCondition ()`

Default constructor.

`CCioSecurityCondition (CBlob const &blob)`

Create and initialize an object with the provided blob.

`CCioSecurityCondition (CCioSecurityCondition const &that)`

Copy constructor.

`CCioSecurityCondition & operator= (CCioSecurityCondition const &that)`

Assignment operator.

`SecurityConditionType` `type() const`

Returns the condition type. Depending on the provided type, methods **authId**, **authReference**, **not**, **and** or **or** may be called to get condition details.

`CBlob authId() const`

Returns id of authorization involved in the object access control.

`CCioAuthReference` `authReference() const`

Returns reference of authorization involved in the object access control.

`CCioSecurityCondition not() const`

Returns security condition involved in the object access control with not operation.

`std::vector< CCioSecurityCondition > and()`

Returns security conditions involved in the object access control with and operation.

`std::vector< CCioSecurityCondition > or()`

Returns security conditions involved in the object access control with or operation.

E.3.1.6 CCioAuthReference

CCioAuthReference is an object representation of card specific authentication method. Refer to ISO/IEC 7816-15 `AuthReference` for a complete description.

```
CCioAuthReference
      CCioAuthReference ()
      CCioAuthReference (CBlob const &blob)
      CCioAuthReference (CCioAuthReference const &that)
      CCioAuthReference & operator= (CCioAuthReference const &that)
      int authMethod() const
      int seIdentifier() const
```

```
CCioAuthReference ()
```

Default constructor.

```
CCioAuthReference (CBlob const &blob)
```

Create and initialize an object with the provided blob.

```
CCioAuthReference (CCioAuthReference const &that)
```

Copy constructor.

```
CCioAuthReference & operator= (CCioAuthReference const &that)
```

Assignment operator.

```
int authMethod() const
```

Returns authentication methods as a bit field. Refer to ISO/IEC 7816-15 AM2:2008 *AuthMethod* for a complete description.

```
int seIdentifier() const
```

Returns id of security environment implied in the object access control rules. Refer to ISO/IEC 7816-15 *AuthReference* for a complete description.

E.3.1.7 CCioAccessControlRule

CCioAccessControlRule is an object representation of card access control rules. Refer to ISO/IEC 7816-15 *`AccessControlRule`* for a complete description.

```
CCioAccessControlRule
        CCioAccessControlRule()
        CCioAccessControlRule & operator= (CCioAccessControlRule const &that)
        AccessModes accessMode ()
        CCioSecurityCondition securityCondition() const
        operator CBlob () const
```

```
CCioAccessControlRule ()
```

Default constructor.

```
CCioAccessControlRule & operator= (CCioAccessControlRule const &that)
```

Assignment operator.

```
AccessModes accessMode ()
```

Returns object access mode as a bitfield. See **AccessModes** for possible masks values.

```
CCioSecurityCondition securityCondition ()
```

Returns object access security conditions.

```
operator CBlob () const
```

Returns binary content of the CCio**AccessControlRule** object. Refer to ISO/IEC 7816-15 for a complete description.

E.3.1.8 CCioObjectValue

CCioObjectValue is an object representation of card object value. Refer to ISO/IEC 7816-15 `ObjectValue` for a complete description.

```
CCioObjectValue
      CCioObjectValue()
      CCioObjectValue(CBlob const &blob)
      CCioObjectValue (CCioObjectValue const &that)
      CCioObjectValue & operator= (CCioObjectValue const &that)
      ObjectValueType type ()
      CBlob value () const
```

```
CCioObjectValue()
```
Default constructor.

```
CCioObjectValue(CBlob const &blob)
```
Create and initialize an object with the provided blob. Refer to ISO/IEC 7816-15 for a complete description.

```
CCioObjectValue (CCioObjectValue const &that)
```
Copy constructor.

```
CCioObjectValue & operator= (CCioObjectValue const &that)
```
Assignment operator.

```
ObjectValueType type ()
```
Returns object value type.

```
CBlob value () const
```
Returns the actual object value.

E.3.1.9 CFileControlParameters

CFileControlParameters is an object representation of card file control parameters (FCP). Refer to ISO/IEC 7816-4 for a complete description.

```
CFileControlParameters
       CFileControlParameters()
       CFileControlParameters(CBlob const &blob)
       CFileControlParameters & operator= (CFileControlParameters const &that)
       operator CBlob () const
       FileType type () const
       int size () const
       CBlob fileIdentifier () const
       CBlob shortFileIdentifier () const
       CBlob dfName () const
       FileState state () const
```

```
CFileControlParameters()
```
Default constructor.

```
CFileControlParameters(CBlob const &blob)
```
Create and initialize an object with the provided blob. Refer to ISO/IEC 7816-4 for a complete description.

```
CFileControlParameters & operator= (CFileControlParameters const &that)
```
Assignment operator.

```
operator CBlob () const
```
Returns raw FCP value as specified in ISO/IEC 7816-4.

```
FileType   type () const
```
Returns file type. See **FileType** enumeration for possible values.

```
int     size () const
```
Returns file size.

```
CBlob   fileIdentifier () const
```
Returns file identifier.

```
CBlob   shortFileIdentifier () const
```
Returns short file identifier.

```
CBlob   dfName () const
```
Returns parent DF name.

```
FileState  state () const
```
Returns file state. See **FileState** enumeration for possible values.

E.3.1.10 CCioUsage

CCioUsage is an object representation of card key object usage. Refer to ISO/IEC 7816-15 *`Usage`* for a complete description.

```
CCioUsage
       CCioUsage()
       CCioUsage(CBlob const &blob)
       CCioUsage & operator= (CCioUsage const &that)
       int keyUsage () const
       std::vector< std::string >    extKeyUsage () const
```

`CCioUsage()`

Default constructor.

`CCioUsage(CBlob const &blob)`

Create and initialize an object with the provided blob. Refer to ISO/IEC 7816-15 for a complete description.

`CCioUsage & operator= (CCioUsage const &that)`

Assignment operator.

`int keyUsage () const`

Returns key usage flags as specified in ISO/IEC 7816-15.

`std::vector< std::string > extKeyUsage () const`

Returns list of extended key usage, as a list of OIDs.

E.3.1.11 CCioCredentialIdentifier

CCioCredentialIdentifier is an object representation of card key or certificate identifier. Refer to ISO/IEC 7816-15 *`CredentialIdentifier`* for a complete description.

```
CCioCredentialIdentifier
      CCioCredentialIdentifier ()
      CCioCredentialIdentifier (CBlob const &blob)
      CCioCredentialIdentifier & operator= (CCioCredentialIdentifier const &that)
      IdType idType() const
      CBlob idValue () const
      CBlob octetStringIdValue () const
```

```
CCioCredentialIdentifier ()
```
Default constructor.

```
CCioCredentialIdentifier (CBlob const &blob)
```
Create and initialize an object with the provided blob. Refer to ISO/IEC 7816-15 for a complete description.

```
CCioUsage & operator= (CCioUsage const &that)
```
Assignment operator.

```
IdType idType() const
```
Returns the type of identifier used to describe the key or the certificate. See **IdType** enumeration for possible values and **idValue** method to get actual identifier value.

```
CBlob idValue () const
```
Returns the actual raw DER encoded identifier value.

```
CBlob octetStringIdValue () const
```
Returns the value part of identifer value, given it is coded as an OCTET STRING

E.3.1.12 CCioKeyInfo

CCioKeyInfo is an object representation of card key information. Refer to ISO/IEC 7816-15 *KeyInfo* for a complete description.

```
CCioKeyInfo
      CCioKeyInfo()
      CCioKeyInfo(CBlob const &blob)
      CCioKeyInfo & operator= (CCioKeyInfo const &that)
      CBlob parameters() const
      Operations operations()const
      bool isNull()const
```

`CCioKeyInfo()`

Default constructor.

`CCioKeyInfo(CBlob const &blob)`

Create and initialize an object with the provided blob. Refer to ISO/IEC 7816-15 for a complete description.

`CCioKeyInfo & operator= (CCioKeyInfo const &that)`

Assignment operator.

`CBlob parameters() const`

Returns details of algorithm parameters. Refer to ISO/IEC 7816-15 for a complete description.

`Operations operations()const`

Returns key operations as a bit field. See **Operations** enumeration for possible masks.

`bool isNull()const`

Returns TRUE when this object has a null value.

E.3.2 File Objects

E.3.2.1 CCiaFile

CCiaFile is an object representation of an EF. This base class can be used to get any card file content in raw format. Specialized subclasses allow specific parsing of main P15 defined files structures (see CApplicationDirectoryFile, CCiaInfoFile, CObjectDirectoryFile and all CxxxFile subclasses).

```
CCiaFile
      CCiaFile()
      CCiaFile(&card, CBlob const &aidCia, CCiaPath const &path)
      void    open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
      const CFileControlParameters & fileControlParameters () const
      void    load (CBlob const &file)
      void load()
      operator CBlob () const
```

`CCiaFile ()`

This is the default constructor. Use **open** to bind this object to a card file.

`CCiaFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)`

This constructor allows creating an object representation of the card file specified by *path*, relative to the specified application *aidCia*.

`void   open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)`

This function binds the **CCiaFile** object to a card EF specified by *path*, relative to the specified application *aidCia*.

`void load(CBlob const &file)`

This function allows parsing a buffer instead of file content.

`void load()`

Call this function to actually read the EF file content.

`const CFileControlParameters & fileControlParameters () const`

Returns information about the card EF referenced by the current object.

`operator CBlob () const`

Returns raw content of EF referenced by the current object. A call to **load** function shall be done prior to calling this function.

E.3.2.2 CApplicationDirectoryFile

CApplicationDirectoryFile is an object representation of EF.DIR. This class can be used to get the list of declared applications.

```
CApplicationDirectoryFile : public CCiaFile
      CApplicationDirectoryFile ()
      CApplicationDirectoryFile (CCiaCard &card)
      CApplicationDirectoryFile (CCiaCard &card, CCiaPath const &path)
      void    load (CBlob const &file)
      void load()
      operator CBlob () const
      std::vector< CApplicationTemplate > & applicationTemplates()
```

```
CApplicationDirectoryFile ()
```

This is the default constructor. Use **CCiaFile::open** to bind this object to a real card file.

```
CApplicationDirectoryFile (CCiaCard &card)
```

This constructor allows accessing default application registry (EFdir), with standard ID 2F00.

```
CApplicationDirectoryFile (CCiaCard &card, CCiaPath const &path)
```

This constructor allows accessing application registry (EFdir) with specified *path*.

```
void load(CBlob const &file)
```

This function allows parsing a buffer compliant with EFdir file format.

```
void load()
```

Call this function to actually read the EFdir file content.

```
operator CBlob () const
```

Returns raw content of EFdir associated with the current object. A call to **load** function shall be done prior to calling this function.

```
std::vector< CApplicationTemplate > & applicationTemplates()
```

Calling application should call this method to get a vector of **CApplicationTemplate** objects, each element being an object representation of an application (or an EFdir record). A call to **load** function shall be done prior to calling this function.

E.3.2.3 CCiaInfoFile

CCiaInfoFile is an object representation of EFCiaInfo. This class can be used to get attributes of specified application. Refer to ISO/IEC 7816-15 `CiaInfo` for a complete description.

```
CciaInfoFile : public CCiaFile
       CCiaInfoFile (byte [])
       CCiaInfoFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
       void    load (CBlob const &file)
       void    load ()
       operator CBlob () const
       int version ()
       CBlob serialNumber ()
       CBlob manufacturerID ()
       CBlob label ()
       int cardflags ()
       std::vector< SecurityEnvironmentInfo >   seInfo ()
       std::vector< std::pair< int, int > > recordInfo () const
       std::vector< AlgorithmInfo >  supportedAlgorithms () const
       CBlob issuerId () const
       CBlob holderId () const
       std::string  lastUpdate () const
       CBlob preferredLanguage () const
       std::vector< std::pair< int, std::string > >    profileIndication ()
```

`CCiaInfoFile ()`

This is the default constructor. Use **CCiaFile::open** to bind this object to a card file.

`CCiaInfoFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)`

This constructor allows creating an object representation of the EFciaInfo file specified by *path*, relative to the specified *aidCia* application.

`void   load (CBlob const &fileContent)`

This function allows parsing a buffer compliant with EFciaInfo format.

`void   load ()`

This function requests actual reading of EFciaInfo content.

`operator CBlob () const`

Returns raw content of EFciaInfo associated with the **CCiaInfoFile** object. A call to **load** function shall be done prior to calling this function.

`int version ()`

Returns CIA specification version.

`CBlob serialNumber ()`

Returns application serial number.

`CBlob manufacturerID ()`

Returns card manufacturer ID.

`CBlob label ()`

Returns application label, allowing application identification.

`int cardflags ()`

Returns card capabilities flags.

```
std::vector< SecurityEnvironmentInfo >   seInfo ()
```
Returns a vector of security environment information applicable to current application.

```
std::vector< std::pair< int, int > > recordInfo () const
```
Returns indication of whether elementary files are linear record files or transparent files. This information is provided as a vector of pairs: contextTag, recordLength.
ContextTag enumeration provides possible values for *contextTag*, to identify object type whose structure is linear record, and where *recordLength* provides related record length.

```
std::vector< AlgorithmInfo >   supportedAlgorithms () const
```
Returns a vector of algorithms applicable to current application.

```
CBlob  issuerId () const
```
Returns card issuer ID.

```
CBlob  holderId () const
```
Returns card holder ID.

```
std::string   lastUpdate () const
```
Returns date of last application update.

```
CBlob  preferredLanguage () const
```
Returns card holder preferred application language.

```
std::vector< std::pair< int, std::string > >  profileIndication ()
```
Returns a list of profile the card is compliant with. This information is provided as a vector of pairs : profileType, profileName. Profile type may be 0 to indicate profile name is provided as an OID, or 1 as a profile name.

```
void   open (CCiaCard &card, CBlob const &aidCia, CCiaPath const &path)
```
This function binds the **CCiaInfoFile** object to a card EF specified by *path*, relative to the specified application *aidCia*.

```
const CFileControlParameters & fileControlParameters () const
```
Returns information about the card EF referenced by the current **CCiaInfoFile**.

E.3.2.4 CObjectDirectoryFile

CObjectDirectoryFile is an object representation of EFod (Object Directory file). This class can be used to get relative path (from CIA application root) of application supported files and their associated object class. Refer to ISO/IEC 7816-15 *EFod* for a complete description.

```
CobjectDirectoryFile : public CCiaFile
        CObjectDirectoryFile ()
        CObjectDirectoryFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const
&path)
        void    load (CBlob const &file)
        void    load ()
        operator CBlob () const
        std::vector< std::pair< CioChoice, CCiaPath > > &cioFiles ()
```

```
CObjectDirectoryFile ()
```

This is the default constructor. Use **CCiaFile::open** to bind this object to a card file.

```
CObjectDirectoryFile (CCiaCard &card, CBlob const &aidCia, CCiaPath const
&path)
```

This constructor allows creating an object representation of the EFod specified by *path*, relative to the specified *aidCia* application root.

```
void    load (CBlob const &fileContent)
```

This function allows parsing a buffer compliant with EFod file content format.

```
void    load ()
```

This function requests actual reading of EFod content.

```
operator CBlob () const
```

Returns raw content of EF associated with the **CObjectDirectoryFile** object. A call to load function shall be done prior to calling this function.

```
std::vector< std::pair< CioChoice, CCiaPath > > &cioFiles ()
```

This function returns a vector of pairs listing all application supported files. Each pair describes a file and is made of an object class indicator and a path to the file, relative to the specified application root.

E.3.2.5 CAuthObjectFile

CAuthObjectFile is an object representation of a card file containing Authentication objects. Refer to ISO/IEC 7816-15 *`AuthObjects`* for a complete description.

```
CAuthObjectFile : public CCiaFile
       CAuthObjectFile()
       CAuthObjectFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
       void    load (CBlob const &file)
       void    load ()
       operator CBlob () const
       std::vector< CCioAuthentication * > & objects ()
```

CAuthObjectFile ()

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

`CAuthObjectFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)`

Use this constructor to instantiate an object bind to file identified with *path* in *aid* application context.

`void    load (CBlob const &file)`

This function allows parsing a buffer compliant with EF class format.

`void    load ()`

This function requests actual reading of card EF content referenced by *CCiaPath*

`operator CBlob () const`

Use this function to get raw file content.

`std::vector< CCioAuthentication * > &    objects ()`

Use this function to get file content parsed as a list of **CCioAuthentication** objects. Applications should use CCioAuthentication::choice() to determine exact derived object class.

E.3.2.6 CCertificateFile

CCertificateFile is an object representation of a card file containing Certificates. Refer to ISO/IEC 7816-15 *`Certificates`* for a complete description.

```
CCertificateFile : public CCiaFile
      CCertificateFile ()
      CCertificateFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
      void    load (CBlob const &file)
      void    load ()
      operator CBlob () const
      std::vector< CCioCertificate * > & objects ()
```

```
CCertificateFile ()
```

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

```
CCertificateFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Use this constructor to instantiate an object bind to file identified by *path* in *aid* application context.

```
void    load (CBlob const &file)
```

This function allows parsing a buffer compliant with EF class format.

```
void    load ()
```

This function requests actual reading of card EF content referenced by *path* (provided in the object constructor)

```
operator CBlob () const
```

Use this function to get raw file content.

```
std::vector< CCioCertificate * > &    objects ()
```

Use this function to get file content parsed as a list of **CCioCertificate** objects. Applications should use CCioCertificate::choice() to determine exact derived object class.

E.3.2.7 CDataContainerFile

CDataContainerFile is an object representation of a card file containing data. Refer to ISO/IEC 7816-15 *`DataContainerObjects`* for a complete description.

```
CDataContainerFile : public CCiaFile
      CDataContainerFile ()
      CDataContainerFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
      void    load (CBlob const &file)
      void    load ()
      operator CBlob () const
      std::vector< CCioDataContainer * > & objects ()
```

```
CDataContainerFile ()
```

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

```
CDataContainerFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Use this constructor to instantiate an object bind to file identified by *path* in *aid* application context.

```
void    load (CBlob const &file)
```

This function allows parsing a buffer compliant with EF class format.

```
void    load ()
```

This function requests actual reading of card EF content referenced by *path* (provided in the object constructor)

```
operator CBlob () const
```

Use this function to get raw file content.

```
std::vector< CCioDataContainer * > & objects ()
```

Use this function to get file content parsed as a list of **CCioDataContainer** objects. Applications should use CCioDataContainer::choice() to determine exact derived object class.

E.3.2.8 CPrivateKeyFile

CPrivateKeyFile is an object representation of a card file containing private keys. Refer to ISO/IEC 7816-15 *PrivateKeys* for a complete description.

```
CPrivateKeyFile : public CCiaFile
      CPrivateKeyFile ()
      CPrivateKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
      void    load (CBlob const &file)
      void    load ()
      operator CBlob () const
      std::vector< CCioPrivateKey * > & objects ()
```

```
CPrivateKeyFile ()
```

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

```
CPrivateKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Use this constructor to instantiate an object bind to file identified with *path* in *aid* application context.

```
void    load (CBlob const &file)
```

This function allows parsing a buffer compliant with EF class format.

```
void    load ()
```

This function requests actual reading of card EF content referenced by *path* (provided in the object constructor)

```
operator CBlob () const
```

Use this function to get raw file content.

```
std::vector< CCioPrivateKey * > &     objects ()
```

Use this function to get file content parsed as a list of **CCioPrivateKey** objects. Applications should use CCioPrivateKey::choice() to determine exact derived object class.

E.3.2.9 CPublicKeyFile

CPublicKeyFile is an object representation of a card file containing public keys. Refer to ISO/IEC 7816-15 `PublicKeys` for a complete description.

```
CPublicKeyFile : public CCiaFile
      CPublicKeyFile ()
      CPublicKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
      void    load (CBlob const &file)
      void    load ()
      operator CBlob () const
      std::vector< CCioPublicKey * > & objects ()
```

`CPublicKeyFile ()`

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

`CPublicKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)`

Use this constructor to instantiate an object bind to file identified with *path* in *aid* application context.

`void    load (CBlob const &file)`

This function allows parsing a buffer compliant with EF class format.

`void    load ()`

This function requests actual reading of card EF content referenced by *path* (provided in the object constructor)

`operator CBlob () const`

Use this function to get raw file content.

`std::vector< CCioPublicKey * > &  objects ()`

Use this function to get file content parsed as a list of **CCioPublicKey** objects. Applications should use CCioPublicKey::choice() to determine exact derived object class.

E.3.2.10 CSecretKeyFile

CSecretKeyFile is an object representation of a card file containing secret keys. Refer to ISO/IEC 7816-15 *SecretKeys* for a complete description.

```
CSecretKeyFile : public CCiaFile
       CSecretKeyFile ()
       CSecretKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
       void    load (CBlob const &file)
       void    load ()
       operator CBlob () const
       std::vector< CCioSecretKey * > & objects ()
```

```
CSecretKeyFile ()
```

Default constructor. Use **CCiaFile::open** to actually bind this object to a card file.

```
CSecretKeyFile (CCiaCard &card, CBlob const &aid, CCiaPath const &path)
```

Use this constructor to instantiate an object bind to file identified with *path* in *aid* application context.

```
void    load (CBlob const &file)
```

This function allows parsing a buffer compliant with EF class format.

```
void    load ()
```

This function requests actual reading of card EF content referenced by *CCiaPath*

```
operator CBlob () const
```

Use this function to get raw file content.

```
std::vector< CCioSecretKey * > &  objects ()
```

Use this function to get file content parsed as a list of **CCioSecretKey** objects.

E.3.3 Data Objects

E.3.3.1 CApplicationTemplate

CApplicationTemplate is an object representation of a card application, or EFDIR record. This class can be used to get attributes of any registered application. Refer to ISO/IEC 7816-15 `Application template` for a complete description.

```
CApplicationTemplate
       CApplicationTemplate ()
       CApplicationTemplate (CBlob const &blob)
       CApplicationTemplate & operator= (CApplicationTemplate const &that)
       CBlob aid()
       CBlob path()
       CBlob label()
```

`CApplicationTemplate ()`

Default constructor.

`CApplicationTemplate (CBlob const &blob)`

This constructor allows parsing a buffer compliant with EFdir record format.

`CApplicationTemplate & operator= (CApplicationTemplate const &that)`

Assignment operator.

`CBlob aid()`

Use this method to get registered application identifier.

`CBlob path()`

Use this method to get registered application path.

`CBlob label()`

Use this method to get registered application label.

E.3.3.2 CCioObject

CCioObject is an abstract base class for all Cio objects**.** Refer to ISO/IEC 7816-15 *`CommonObjectAttributes`* for a complete description.

```
CCioObject
      CBlob label () const
      CommonObjectFlags flags () const
      CBlob authIdReference () const
      int userConsent () const
      std::vector< CCioAccessControlRule > accessControlRules () const
```

```
CBlob label () const
```
Provides the label associated to the object.

```
CommonObjectFlags flags () const
```
Returns access information about the object.

```
CBlob authIdReference () const
```
Returns ID of authentication associated with the object.

```
int userConsent () const
```
Indicates if authentication is required before performing a signature.

```
std::vector< CCioAccessControlRule > accessControlRules () const
```
Returns information on access mode and associated security mechanisms.

Note: this class cannot be instantiated.

E.3.3.3 CCioAuthentication

CCioAuthentication is an abstract base class for object representations of card Authentication objects. Refer to ISO/IEC 7816-15 `CommonAuthenticationObjectAttributes` for a complete description.

```
CCioAuthentication : public CCioObject
      virtual int choice() const=0
      CBlob authId () const
      int authReference () const
      int seIdentifier () const
```

```
virtual int choice () const
```

Virtual abstract method to be implemented by subclasses. Allows applications determining which kind of object is returned by **CCioAuthObjectFile::objects()** method. Refer to ISO/IEC 7816-15 *AuthenticationObjectChoice* for possible values (untagged value is -1).

```
CBlob authId () const
```

Returns authentication identifier.

```
int authReference () const
```

Returns authentication reference.

```
int seIdentifier () const
```

Return SecurityEnvironment reference.

NOTE this class cannot be instantiated.

E.3.3.4 CCioPasswordAuth

CCioPasswordAuth is a class for object representation of a card Password Authentication object. Refer to ISO/IEC 7816-15 `PasswordAttributes` for a complete description.

```
CCioPasswordAuth: public CCioAuthentication
      CCioPasswordAuth ()
      CCioPasswordAuth (CBlob const &blob)
      int choice() const
      int pwdFlags () const
      PasswordType   pwdType () const
      int minLength () const
      int storedLength () const
      int maxLength () const
      int pwdReference () const
      unsigned char  padChar () const
      bool    padCharDefined () const
      std::string lastPasswordChange () const
      CCiaPath path () const
```

`CCioPasswordAuth ()`

Default constructor.

`CCioPasswordAuth (CBlob const &blob)`

Constructor with object initialization from a buffer compliant with EF file content.

`int choice () const`

Base class abstract method implementation. Allows applications determining if a **CCioAuthentication** object is an instance of **CCioPasswordAuth**. Refer to ISO/IEC 7816-15 *AuthenticationObjectChoice* for related value (untagged value -1).

`int     pwdFlags () const`

Returns password flags indicating how a password can be manipulated. Refer to ISO/IEC 7816-15:2004/AM2 for a complete description of *PasswordFlags* definition.

`PasswordType   pwdType () const`

Returns password type.

`int     minLength () const`

Returns minimum length of new password (if change is allowed).

`int     storedLength () const`

Returns actual password length.

`int     maxLength () const`

Returns maximum length of new password (if change is allowed).

`int     pwdReference () const`

Returns password reference, equivalent to *p2* parameter of **CP15Card::verify** command.

`unsigned char     padChar () const`

Returns padding character to pad passwords, if indicated in *pwdFlags.*

`bool    padCharDefined () const`

Returns true if *pwdFlags* attribute *needs-padding* is set, false otherwise.

```
std::string   lastPasswordChange () const
```

Returns last password change date.

```
CCiaPath   path () const
```

Returns path to the DF in which the password is located.

E.3.3.5 CCioCertificate

CCioCertificate is an abstract class for object representation of a card certificate object.

```
CCioCertificate : public CCioObject
      virtual int choice() const=0
      CBlob id () const
      bool authority ()
      CBlob certHash ()
      CCioUsage trustedUsage ()
      std::vector< CCioCredentialIdentifier > identifiers ()
      Validity validity ()
```

```
virtual int choice () const
```

Virtual abstract method to be implemented by subclasses. Allows applications determining which kind of object is returned by **CCioCertificateFile::objects()** method.

```
CBlob id () const
```

Returns id of certificate, common with the private and public keys related to the certificate.

```
bool authority ()
```

Returns true if the certificate is for an authority (e.g Certification Authority).

```
CBlob certHash ()
```

Returns the hash of the certificate.

```
CCioUsage trustedUsage ()
```

Indicates for which purpose(s) the certificate is trusted by the cardholder; if no trustedUsage componet is present, all usage is possible for the certificate.

```
std::vector< CCioCredentialIdentifier > identifiers ()
```

Provides identifiers (or aliases) from which the certificate can be found or referenced.

```
Validity validity ()
```

Provides information about the certificate’s validity period.

E.3.3.6 CCioX509Certificate

CCioX509Certificate is a class for object representation of a card X509 certificate object. Refer to ISO/IEC 7816-15 *`X509CertificateAttributes`* for a complete description.

```
CCioX509Certificate : public CCioCertificate
      CCioX509Certificate()
      CCioX509Certificate(CBlob const &blob)
      int choice() const
      CCioObjectValue value () const
      CBlob subject () const
      CBlob issuer () const
      CBlob serialNumber () const
```

```
CCioX509Certificate()
```

Default constructor.

```
CCioX509Certificate(CBlob const &blob)
```

Constructor with object initialization from a buffer compliant with EF file content.

```
int choice () const
```

Base class abstract method implementation. Allows applications determining if a **CCioCertificate** object is an instance of **CCioX509Certificate**.

```
CCioObjectValue value () const
```

Returns the value of the certificate.

```
CBlob subject () const
```

Returns certificate's subject field.

```
CBlob issuer () const
```

Returns certificate's issuer field.

```
CBlob serialNumber () const
```

Returns certificate's serial number field.

E.3.3.7 CCioDataContainer

CCioDataContainer is an abstract class for object representation of a card data object. Refer to ISO/IEC 7816-15 *`CommonDataContainerObjectAttributes`* for a complete description.

```
CCioDataContainer : public CCioObject
      virtual int choice() const=0
      CBlob applicationName () const
      std::string applicationOID () const
      CBlob id () const
```

```
virtual int choice () const
```

Virtual abstract method to be implemented by subclasses. Allows applications determining which kind of object is returned by **CDataContainerFile::objects()** method.

```
CBlob applicationName () const
```

Returns the name of the application to which the data container belongs.

```
std::string applicationOID () const
```

Returns the OID of the application to which the data container belongs.

```
CBlob id () const
```

Returns object unique identifier.

E.3.3.8 CCioOpaqueDO

CCioOpaqueDO is a class for object representation of an opaque card data object. Refer to ISO/IEC 7816-15 *`OpaqueDOAttributes`* for a complete description.

```
CCioOpaqueDO : public CCioDataContainer
       CCioOpaqueDO()
       CCioOpaqueDO(CBlob const &blob)
       int choice() const
       CCioObjectValue value() const
```

```
CCioOpaqueDO()
```
Default constructor.

```
CCioOpaqueDO(CBlob const &blob)
```
Constructor with object initialization from a buffer compliant with EF file content.

```
int choice () const
```
Base class abstract method implementation. Allows applications determining if a **CCioDataContainer** object is an instance of **CCioOpaqueDO**.

```
CCioObjectValue value() const
```
Returns data container value.

E.3.3.9 CCioKey

CCioKey is an abstract class for object representation of a card key object. Refer to ISO/IEC 7816-15 *`CommonKeyAttributes`* for a complete description.

```
CCioKey : CCioObject
      CBlob id () const
      int usage () const
      bool native ()
      int accessFlags ()
      int accessFlagsDefined ()
      int keyReference () const
      std::string startDate () const
      std::string endDate () const
      std::vector< int > algReference ()
```

`CBlob id () const`

Returns key identifier, common with private key and certificates.

`int usage () const`

Returns possible usages of the key.

`bool native ()`

Returns true if cryptograhic algorithms associated with the key are implemented in the card hardware.

`int accessFlags ()`

Returns key access flags.

`int accessFlagsDefined ()`

Returns true when access flags are specified for the current key, false otherwise.

`int keyReference () const`

Returns card-specific key reference.

`std::string startDate () const`

Returns the date from which the key is valid for use.

`std::string endDate () const`

Returns the date until which the key is valid for use.

`std::vector< int > algReference ()`

Returns the algorithms the key may be used with.

E.3.3.10 CCioPrivateKey

CCioPrivateKey is an abstract class for object representation of a card private key object. Refer to ISO/IEC 7816-15 *CommonPrivateKeyAttributes* for a complete description.

```
CCioPrivateKey : public CCioKey
        virtual int choice() const=0
        CBlob name ()
        std::vector< CCioCredentialIdentifier > keyIdentifiers ()
        CBlob generalName () const
```

```
virtual int choice () const
```

Virtual abstract method to be implemented by subclasses. Allows applications determining which kind of object is returned by **CCioPrivateKeyFile::objects()** method.

```
CBlob name ()
```

Returns the owner of the key, as specified in the corresponding certificate's subject field.

```
std::vector< CCioCredentialIdentifier > keyIdentifiers ()
```

Provides identifiers (or aliases) from which the key can be found or referenced.

```
CBlob generalName () const
```

Returns private key owner name.

E.3.3.11 CCioPrivateDHKey

CCioPrivateDHKey is a class for object representation of a card DH private key object. Refer to ISO/IEC 7816-15 `PrivateDHKeyAttributes` for a complete description.

```
CCioPrivateDHKey : public CCioPrivateKey
      CCioPrivateDHKey()
      CCioPrivateDHKey(CBlob const &blob)
      int choice() const
      CCiaPath value()
      CCioKeyInfo keyInfo ()
```

```
CCioPrivateDHKey()
```

Default constructor.

```
CCioPrivateDHKey(CBlob const &blob)
```

Constructor with object initialization from a buffer compliant with EF file content.

```
int choice () const
```

Base class abstract method implementation. Allows applications determining if a **CCioPrivateKey** object is an instance of **CCioPrivateDHKey**.

```
CCiaPath value()
```

Returns the path to the private Diffie-Hellman key.

```
CCioKeyInfo keyInfo ()
```

Returns private key information.

E.3.3.12 CCioPrivateRSAKey

CCioPrivateRSAKey is a class for object representation of a card RSA private key object. Refer to ISO/IEC 7816-15 *PrivateRSAKeyAttributes* for a complete description.

```
CCioPrivateRSAKey : public CCioPrivateKey
      CCioPrivateRSAKey()
      CCioPrivateRSAKey(CBlob const &blob)
      int choice() const
      CCiaPath  value()
      CCioKeyInfo keyInfo ()
      int modulusLength ()
```

```
CCioPrivateRSAKey()
```
Default constructor.

```
CCioPrivateRSAKey(CBlob const &blob)
```
Constructor with object initialization from a buffer compliant with EF file content.

```
int choice () const
```
Base class abstract method implementation. Allows applications determining if a **CCioPrivateKey** object is an instance of **CCioPrivateRSAKey**.

```
CCiaPath  value()
```
Returns the path to the private RSA key.

```
CCioKeyInfo keyInfo ()
```
Returns private key information.

```
int modulusLength ()
```
Returns the key length in bits.

E.3.3.13 CCioPublicKey

CCioPublicKey is an abstract class for object representation of a card public key object. Refer to ISO/IEC 7816-15 *CommonPublicKeyAttributes* for a complete description.

```
CCioPublicKey : public CCioKey
      virtual int choice() const=0
      CBlob name ()
      CCioUsage trustedUsage () const
      std::vector< CCioCredentialIdentifier > keyIdentifiers () const
      CBlob generalName () const
```

`virtual int choice () const`

Virtual abstract method to be implemented by subclasses. Allows applications determining which kind of object is returned by **CCioPublicKeyFile::objects()** method.

`CBlob name ()`

Returns the owner of the key, as specified in the corresponding certificate's subject field.

`CCioUsage trustedUsage () const`

Indicates for which purpose(s) the key is trusted by the cardholder.

`std::vector< CCioCredentialIdentifier > keyIdentifiers () const`

Provides identifiers (or aliases) from which the key can be found or referenced.

`CBlob generalName () const`

Returns public key owner name.

E.3.3.14 CCioPublicDHKey

CCioPublicDHKey is a class for object representation of a card public DH key object. Refer to ISO/IEC 7816-15 *PublicDHKeyAttributes* for a complete description.

```
CCioPublicDHKey : public CCioPublicKey
        CCioPublicDHKey ()
        CCioPublicDHKey (CBlob const &blob)
        int choice() const
        CCioObjectValue  value()
        CCioKeyInfo keyInfo ()
```

```
CCioPublicDHKey ()
```
Default constructor.

```
CCioPublicDHKey (CBlob const &blob)
```
Constructor with object initialization from a buffer compliant with EF file content.

```
int choice () const
```
Base class abstract method implementation. Allows applications determining if a **CCioPublicKey** object is an instance of **CCioPublicDHKey**.

```
CCioObjectValue  value()
```
Returns the path to the public Diffie-Hellman key.

```
CCioKeyInfo keyInfo ()
```
Returns public key information.

E.3.3.15 CCioPublicRSAKey

CCioPublicRSAKey is a class for object representation of a card public RSA key object. Refer to ISO/IEC 7816-15 *PublicRSAKeyAttributes* for a complete description.

```
CCioPublicRSAKey : public CCioPublicKey
      CCioPublicRSAKey ()
      CCioPublicRSAKey (CBlob const &blob)
      int choice() const
      CCioObjectValue  value()
      int modulusLength ()
      CCioKeyInfo keyInfo ()
```

`CCioPublicRSAKey ()`

Default constructor.

`CCioPublicRSAKey (CBlob const &blob)`

Constructor with object initialization from a buffer compliant with EF file content.

`int choice () const`

Base class abstract method implementation. Allows applications determining if a **CCioPublicKey** object is an instance of **CCioPublicRSAKey**.

`CCioObjectValue  value()`

Returns the path to the public RSA key.

`int  modulusLength ()`

Returns the key length in bits.

`CCioKeyInfo keyInfo ()`

Returns public key information.