RSA LABORATORIES

PKCS #15 v1.1: Cryptographic Token Information Syntax Standard (draft)

RSA Laboratories

Draft 2 – March 16, 2000

Editor's note: This is the 2nd draft of PKCS #15 v1.1, which is available for a 30-day public review period. Please send comments and suggestions, both technical and editorial, to pkcs-editor@rsasecurity.com or pkcs-tng@rsasecurity.com.

Table of Contents

1 Introduction

1.1 Background

Cryptographic tokens, such as Integrated Circuit Cards (or IC cards) are intrinsically secure computing platforms ideally suited to providing enhanced security and privacy functionality to applications. They can handle authentication information such as digital certificates and capabilities, authorizations and cryptographic keys. Furthermore, they are capable of providing secure storage and computational facilities for sensitive information such as:

- private keys and key fragments;
- account numbers and stored value;
- passwords and shared secrets; and
- authorizations and permissions.

At the same time, many of these tokens provides an isolated processing facility capable of using this information without exposing it within the host environment where it is at potential risk from hostile code (viruses, Trojan horses, and so on). This becomes critically important for certain operations such as:

- generation of digital signatures, using private keys, for personal identification;
- network authentication based on shared secrets;
- maintenance of electronic representations of value; and
- portable permissions for use in off-line situations.

Unfortunately, the use of these tokens for authentication and authorization purposes has been hampered by the lack of interoperability at several levels. First, the industry lacks standards for storing a common format of digital credentials (keys, certificates, etc.) on them. This has made it difficult to create applications that can work with credentials from a variety of technology providers. Attempts to solve this problem in the application domain invariably increase costs for both development and maintenance. They also create a significant problem for the end-user since credentials are tied to a particular application running against a particular application-programming interface to a particular hardware configuration.

Second, mechanisms to allow multiple applications to effectively share digital credentials have not yet reached maturity. While this problem is not unique to cryptographic cards - it is already apparent in the use of certificates with World Wide Web browsers, for example - the limited room on many cards together with the consumer expectation of universal acceptance will force credential sharing on credential providers. Without agreed-upon standards for credential sharing, acceptance and use of them both by application developers and by consumers will be limited.

To optimize the benefit to both the industry and end-users, it is important that solutions to these issues be developed in a manner that supports a variety of operating environments, application programming interfaces, and a broad base of applications. Only through this approach can the needs of constituencies be supported and the development of credentials-activated applications encouraged, as a cost-effective solution to meeting requirements in a very diverse set of markets.

The objectives of this document are therefore to:

- enable interoperability among components running on various platforms (platform neutral);
- enable applications to take advantage of products and components from multiple manufacturers (vendor neutral);
- enable the use of advances in technology without rewriting application-level software (application neutral); and
- maintain consistency with existing, related standards while expanding upon them only where necessary and practical.

As a practical example, the holder of an IC card containing a digital certificate should be able to present the card to any application running on any host and successfully use the card to present the contained certificate to the application.

As a first step to achieve these objectives, this document specifies a file and directory format for storing security-related information on cryptographic tokens. It has the following characteristics:

- dynamic structure enables implementations on a wide variety of media, including stored value cards;
- allows multiple applications to reside on the card (even multiple EID applications);
- supports storage of any type of objects (keys, certificates and data); and
- support for multiple PINs whenever the token supports it

In general, an attempt has been made to be flexible enough to allow for many different token types, while still preserving the requirements for interoperability. A key factor for this in the case of IC cards is the notion of "Directory Files" (See Section 5.5) which provides a layer of indirection between objects on the card and the actual format of these objects.

This document supersedes PKCS #15 v1.0 [30], but is backward compatible.

1.2 Information access model

The PKCS #15 token information may be read when a token is presented containing this information, and is used by a PKCS #15 interpreter which is part of the software environment, e.g. as shown in the figure below.

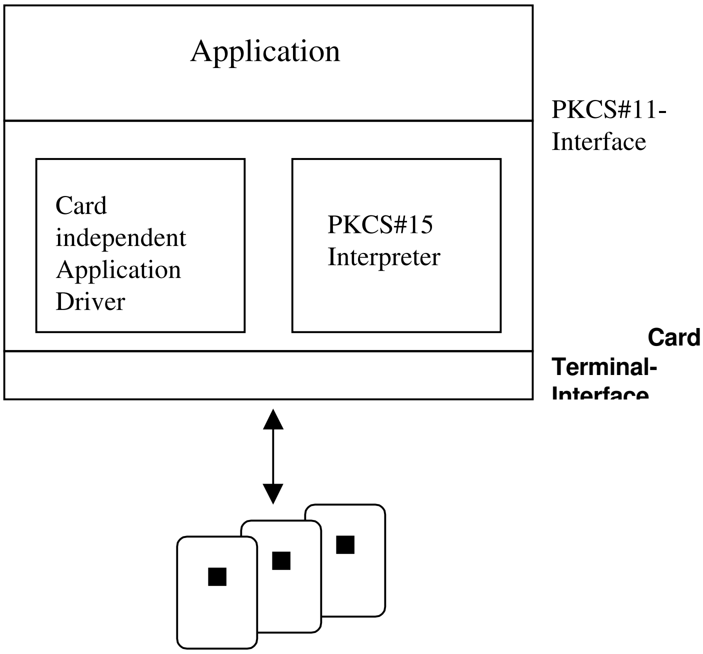


Figure 1 – Embedding of a PKCS #15 interpreter (example)

2 Terms and definitions

For the purposes of this document, the following definitions apply:

application	the data structure, data elements and program modules needed for a specific functionality to be satisfied ([16])
application identifier	data element that identifies an application in a card NOTE – Adapted from [13]
application protocol data unit	message between the card and the interface device, e.g. host computer NOTE – Adapted from [12]
application provider	entity that provides an application NOTE – Adapted from [13]
authentication object directory file	optional elementary file contaning information about authentication objects known to the PKCS #15 application
binary coded decimal	Number representation where a number is expressed as a sequence of decimal digits and then each decimal digit is encoded as a four bit binary number

Example – Decimal 92 would be encoded as the eight bit sequence 1001 0010.

cardholder	person for whom the card was issued
card issuer	organization or entity that issues smart cards and card applications
certificate directory file	optional elementary file containing information about certificate known to the PKCS #15 application
command	message that initiates an action and solicits a response from the card
data object directory file	optional elementary file containing information about data objects known to the PKCS #15 application
dedicated file	file containing file control information, and, optionally, memory available for allocation, and which may be the parent of elementary files and/or other dedicated files NOTE – Adapted from [12]
directory (DIR) file	optional elementary file containing a list of applications supported by the card and optional related data elements NOTE – Adapted from [13]
elementary file	set of data units or records that share the same file identifier, and which cannot be a parent of another file NOTE – Adapted from [12]
file identifier	2-byte binary value used to address a file on a smart card NOTE – Adapted from [12]
function	process accomplished by one or more commands and resultant actions that are used to perform all or part of a transaction
master file	mandatory unique dedicated file representing the root of the structure [12] NOTE – The MF typically has the file identifier $3F00_{16}$.
message	string of bytes transmitted by the interface device

	to the card or vice versa, excluding transmission-oriented characters NOTE – Adapted from [12]
object directory file	elementary file containing information about other directory files in the PKCS #15 application
password	data that may be required by the application to be presented to the card by its user before data or functions can be processed NOTE – Adapted from [12]
path	concatenation of file identifiers without delimitation NOTE 1 – Adapted from [12] NOTE 2 – If the path starts with the MF identifier ($3F00_{16}$), it is an absolute path; otherwise it is a relative path. A relative path shall start with the identifier '$3FFF_{16}$' or with the identifier of the current DF.
personal identification number (PIN)	4 to 8 digit number entered by the cardholder to verify that the cardholder is authorized to use the card
private key directory file	optional elementary file containing information about private keys known to the PKCS #15 application
provider	authority who has or who obtained the right to create the MF or a DF in the card NOTE – Adapted from [13]
public key directory file	optional elementary file containing information about public keys known to the PKCS #15 application
record	string of bytes which can be handled as a whole by the card and referenced by a record number or by a record identifier ([12])
secret key directory file	optional elementary file containing information about secret keys known to the PKCS #15 application
stored value card	card that stores non-bearer information like electronic cash
template	value field of a constructed data object, defined to

	give a logical grouping of data objects ([14])
token	portable device capable of storing persistent data

3 Symbols, abbreviated terms and document conventions

3.1 Symbols

DF(*x*)	Dedicated file *x*
EF(*x*)	Elementary file *x*

3.2 Abbreviated terms

For the purposes of this document, the following abbreviations apply:

AID	application provider identifier
AODF	authentication object directory file
APDU	application protocol data unit
BCD	binary-coded decimal
CDF	certificate directory file
DF	dedicated File
DODF	data object directory file
EF	elementary file
IFD	interface device (e.g. reader)
MF	master file
ODF	object directory file
PIN	personal identification number
PrKDF	private key directory file
PuKDF	public key directory file
RID	registered application provider identifier
SKDF	secret key directory file
URL	uniform resource locator

3.3 Document conventions

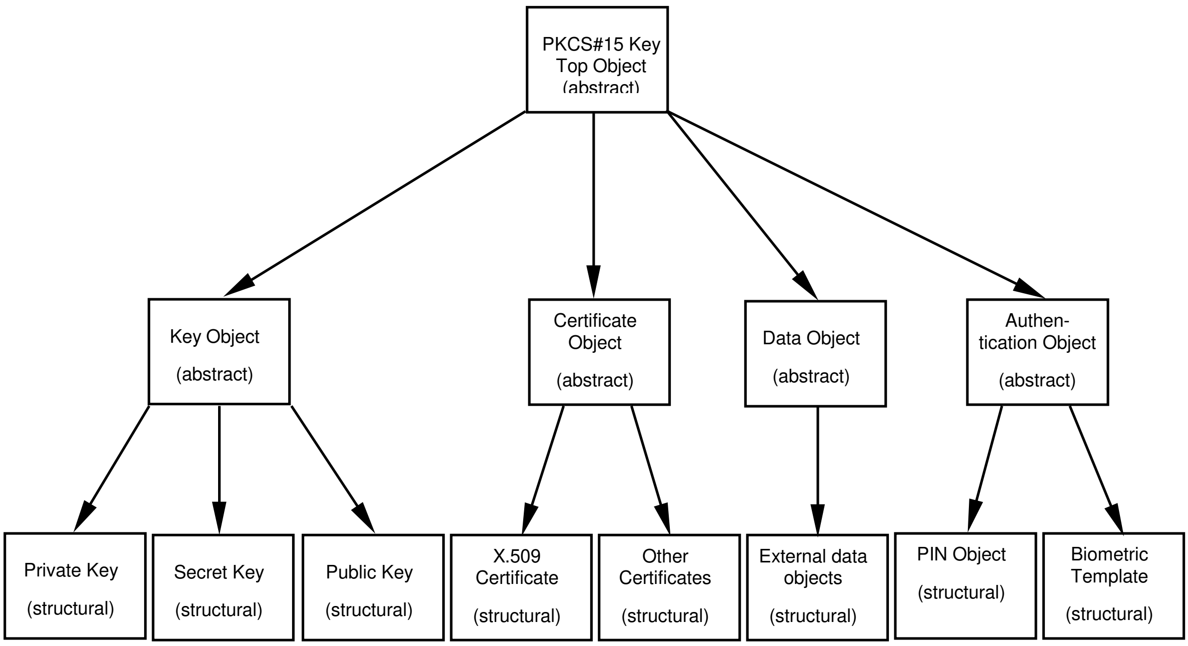
This document presents ASN.1 ([18], [19], [20], and [21]) notation in the **bold Helvetica** typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the **bold Helvetica** typeface. The names of commands, typically referenced when specifying information exchanges between cards and IFDs, are differentiated from normal text by displaying them in the `Courier` typeface.

4 Overview

4.1 Object model

4.1.1 Object classes

This document defines four general classes of objects: Keys, Certificates, Authentication Objects and Data Objects. All these object classes have sub-classes, e.g. Private Keys, Secret Keys and Public Keys, whose instantiations become objects actually stored on cards. The following is a figure of the object hierarchy:



NOTE – instances of abstract object classes does not exist on cards

Figure 2 – PKCS #15 Object hierarchy

4.1.2 Attribute types

All objects have a number of attributes. Objects "inherits" attribute types from their parent classes (in particular, every object inherit attributes from the abstract PKCS #15 "Common" or "Top" object). Attributes are defined in detail in Section 6.

4.1.3 Access methods

Objects can be private, meaning that they are protected against unauthorized access, or public. In the IC card case, access (read, write, etc) to private objects is defined by *Authentication Objects* (which also includes *Authentication Procedures*). Conditional access (from a cardholder's perspective) is achieved with knowledge-based or biometric user information. In other cases, such as when PKCS #15 is implemented in software, private objects may be protected against unauthorized access by cryptographic means.

Public objects are not protected from read-access. Whether they are protected against modifications or not depends on the particular implementation.

5 IC card file format

5.1 Overview

In general, an IC card file format specifies how certain abstract, higher level elements such as keys and certificates are to be represented in terms of more lower level elements such as IC card files and directory structures. The format may also suggest how and under which circumstances these higher level objects may be accessed by external sources and how these access rules are to be implemented in the underlying representation (i.e. the card's operating system). However, since it is anticipated that this document will be used in many types of applications, this latter task has been left to application providers' discretion. Some general suggestions can be found in Appendix A, though, and specific requirements for an Electronic Identity Profile of this specification can be found in Appendix B.

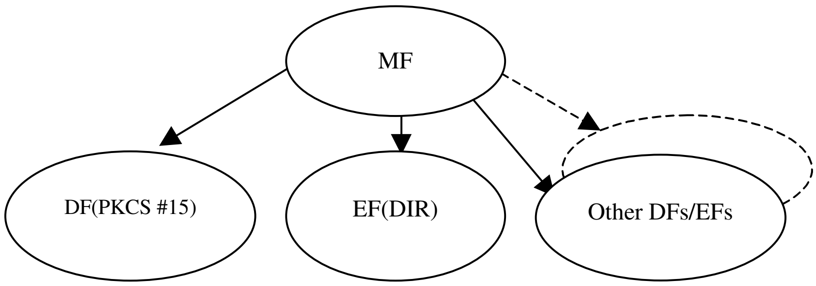
NOTE – The words "format" and "contents" shall be interpreted to mean "The way the information appears to a host side application making use of a predefined set of commands (selected from [12] and possibly [15] and [16]) to access this data." It may well be that a particular card is able to store the information described here in a more compact or efficient way than another card, however the "card-edge" representation of the information shall be the same in both cases. This document is therefore a "card-edge" specification.

5.2 IC card requirements

This section of this document requires that compliant cards have necessary support for ISO/IEC 7816-4, ISO/IEC 7816-5 and ISO/IEC 7816-6 (hierarchic logical file system, direct or indirect application selection, access control mechanisms and read operations). Extended features, especially advanced PIN management functions and higher level security operations may require support for ISO/IEC 7816-8 and/or ISO/IEC 7816-9 (or parts thereof) as well.

5.3 Card file structure

A typical card supporting this specification will have the following layout:



NOTE – For the purpose of this document, EF(DIR) is only needed on IC cards which do not support direct application selection as defined in ISO/IEC 7816-5 or when multiple PKCS #15 applications reside on a single card.

Figure 3 – Typical PKCS #15 Card Layout

The general file structure is shown above. The contents of the PKCS #15 Application Directory is somewhat dependent on the type of IC card and its intended use, but the following file structure is believed to be the most common:

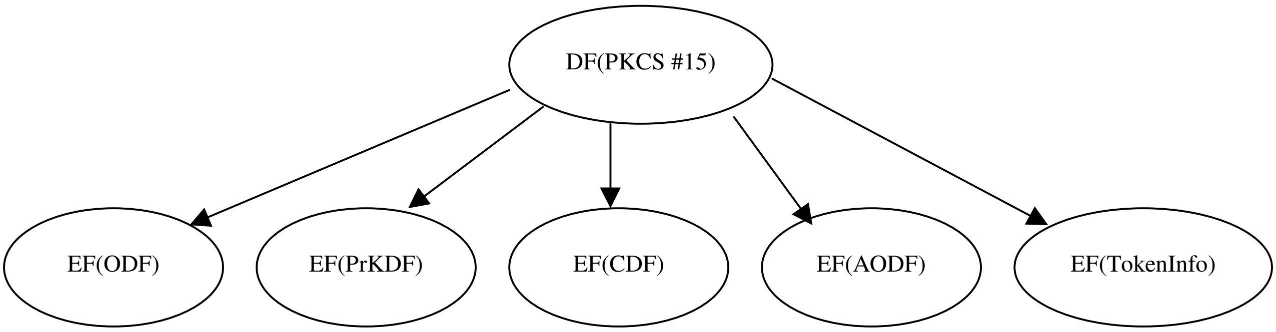


Figure 4 – Contents of DF(PKCS15) (Example).

Other possible topologies are discussed in Annex D. The contents and purpose of each file and directory is described below.

5.4 MF directory contents

5.4.1 EF(DIR)

This optional file shall, if present, contain one or several application templates as defined in ISO/IEC 7816-5. The application template (tag **'61'H**) for a PKCS15 application shall at least contain the following DOs:

– Application Identifier (tag **'4F'H**), value defined in this document

– Path (tag **'51'H**), value supplied by application issuer

Other tags from ISO/IEC 7816-5 may, at the application issuer's discretion, be present as well. In particular, it is recommended that application issuers include both the "Discretionary ASN.1 data objects" data object (tag **'73'H**) and the "Application label" data object (tag **'50'H**). The application label shall contain an UTF-8 encoded label for the application, chosen by the card issuer. The "Discretionary ASN.1 data objects" data object shall, if present, contain a DER-encoded ([22]) value of the ASN.1 type **DDO**:

```
DDO ::= SEQUENCE {
        oid             OBJECT IDENTIFIER,
        odfPath         Path OPTIONAL,
        tokenInfoPath   [0] Path OPTIONAL,
        unusedPath      [1] Path OPTIONAL,
        ... -- For future extensions
        }
```

The **oid** field shall contain an object identifier uniquely identifying the card issuer's implementation. The **odfPath**, **tokenInfoPath** and **unusedPath** fields shall, if present, contain paths to elementary files EF(ODF), EF(TokenInfo) or EF(UnusedSpace) respectively (see Section 5.5). This provides a way for issuers to use non-standard file identifiers for these files without sacrificing interoperability. It also provides card issuers with the opportunity to share TokenInfo files between PKCS #15 applications, when several PKCS #15 applications reside on one card. To summarize, each (logical) record in EF(DIR) must be of the following ASN.1 type, conformant with ISO/IEC 7816-5:

```
DIRRecord ::= [APPLICATION 1] SEQUENCE {
        aid     [APPLICATION 15] OCTET STRING,
        label   [APPLICATION 16] UTF8String OPTIONAL,
        path    [APPLICATION 17] OCTET STRING,
        ddo     [APPLICATION 19] DDO OPTIONAL
        }
```

The use of a DIR files will simplify application selection when several PKCS #15 applications reside on one card. An example of EF(DIR) contents may be found in Annex **Error! Reference source not found.**.

5.5 PKCS #15 application directory contents

5.5.1 EF(ODF)

The mandatory Object Directory File (ODF) is an elementary file, which contains pointers to other EFs (PrKDFs, PuKDFs, SKDFs, CDFs, DODFs and AODFs), each one containing a directory over PKCS #15 objects of a particular class. The ASN.1 syntax for the contents of EF(ODF) is described in Section 6.2.

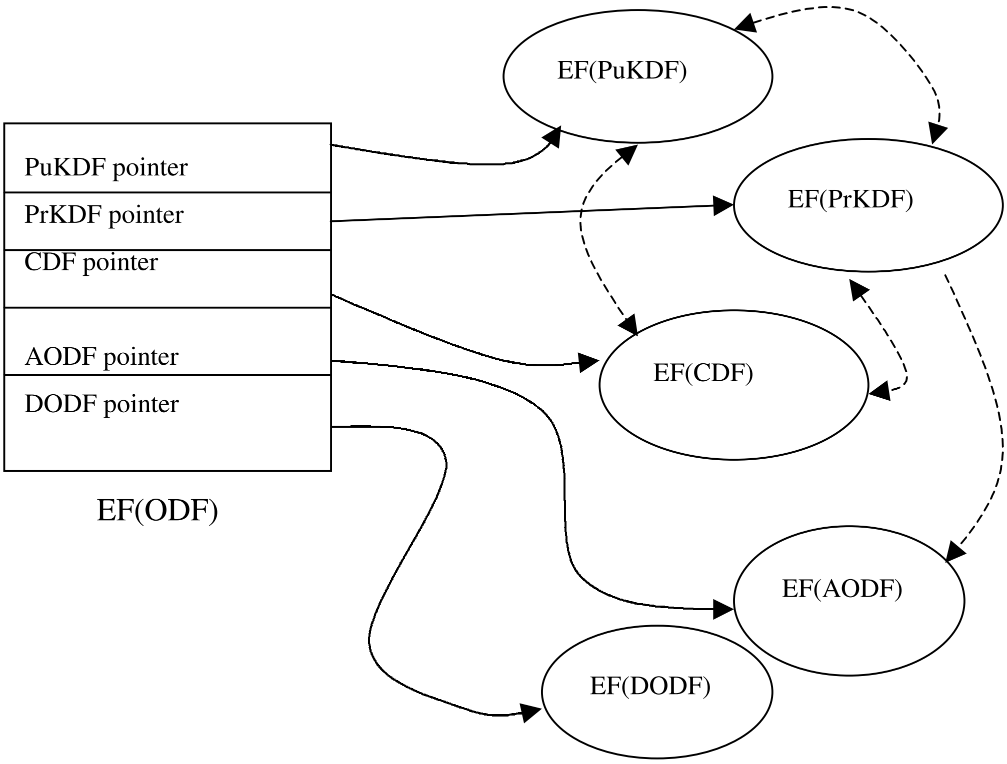


Figure 5 – EF(ODF) points to other EFs. Dashed arrows indicate cross-references

5.5.2 Private Key Directory Files (PrKDFs)

These elementary files can be regarded as directories of private keys known to the PKCS #15 application. They are optional, but at least one PrKDF must be present on an IC card which contains private keys (or references to private keys) known to the PKCS #15 application. They contain general key attributes such as labels, intended usage, identifiers, etc. When applicable, they also contain cross-reference pointers to authentication objects used to protect access to the keys. The rightmost arrow in Figure 4 indicates this. Furthermore, they contain pointers to the keys themselves. There can be any number of PrKDFs in a PKCS #15 DF, but it is anticipated that in the normal case there will be at most one. The keys themselves may reside anywhere on the card. The ASN.1 syntax for the contents of PrKDFs is described in Section 6.3.

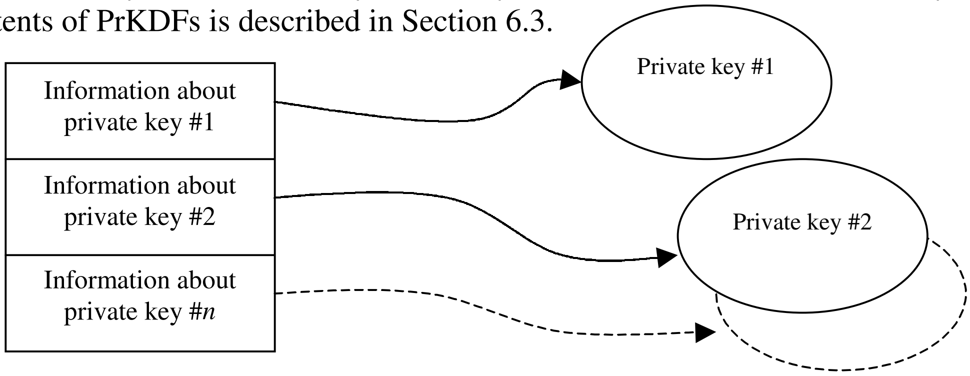


Figure 6 – EF(PrKDF) contains private key attributes and pointers to the keys

5.5.3 Public Key Directory Files (PuKDFs)

These elementary files can be regarded as directories of public keys known to the PKCS #15 application. They are optional, but at least one PuKDF must be present on an IC card which contains public keys (or references to public keys) known to the PKCS #15 application. They contain general key attributes such as labels, intended usage, identifiers, etc. Furthermore, they contain pointers to the keys themselves. When the private key corresponding to a public key also resides on the card, the keys must share the same identifier (this is indicated with a dashed-arrow in Figure 4). There can be any number of PuKDFs in a PKCS #15 DF, but it is anticipated that in the normal case there will be at most one. The keys themselves may reside anywhere on the card. The ASN.1 syntax for the contents of PuKDFs is described in Section 6.4.

NOTE – When a certificate object on the card contains the public key, the public key object and the certificate object shall share the same identifier. This means that in some cases three objects (a private key, a public key and a certificate) will share the same identifier.

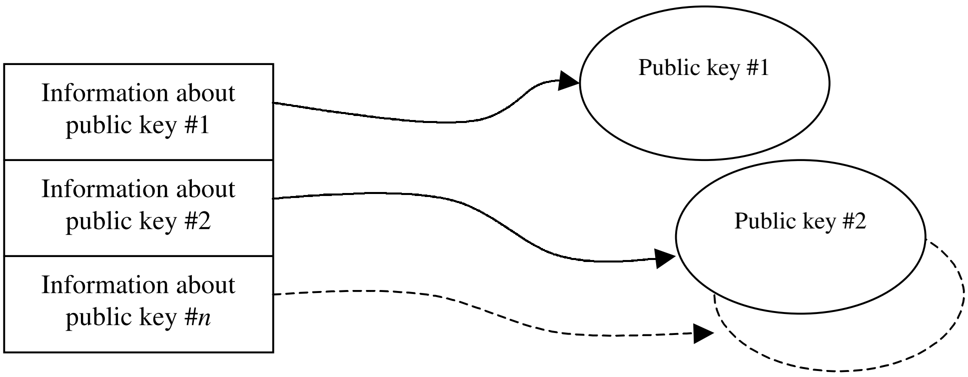


Figure 7 – EF(PuKDF) contains public key attributes and pointers to the keys

5.5.4 Secret Key Directory Files (SKDFs)

These elementary files can be regarded as directories of secret keys known to the PKCS #15 application. They are optional, but at least one SKDF must be present on an IC card which contains secret keys (or references to secret keys) known to the PKCS #15 application. They contain general key attributes such as labels, intended usage, identifiers, etc. When applicable, they also contain cross-reference pointers to authentication objects used to protect access to the keys. Furthermore, they contain pointers to the keys themselves. There can be any number of SKDFs in a PKCS #15 DF, but it is anticipated that in the normal case there will be at most one. The keys themselves may reside anywhere on the card. The ASN.1 syntax for the contents of SKDFs is described in Section 6.5.

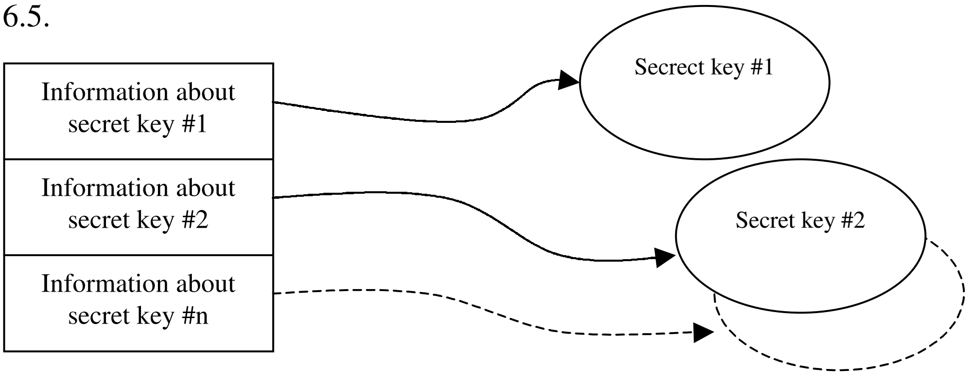


Figure 8 – EF(SKDF) contains secret key attributes and pointers to the keys

5.5.5 Certificate Directory Files (CDFs)

These elementary files can be regarded as directories of certificates known to the PKCS #15 application. They are optional, but at least one CDF must be present on an IC card which contains certificates (or references to certificates) known to the PKCS #15 application. They contain general certificate attributes such as labels, identifiers, etc. When a certificate contains a public key whose private key also resides on the card, the certificate and the private key must share the same identifier (this is indicated with a dashed-arrow in Figure 4). Furthermore, certificate directory files contain pointers to the certificates themselves. There can be any number of CDFs in a PKCS #15 DF, but it is anticipated that in the normal case there will only be one or two (one for trusted certificates and one which the cardholder may update). The certificates themselves may reside anywhere on the card (or even outside the card, see Section 8). The ASN.1 syntax for the contents of CDFs is described in Section 6.6.

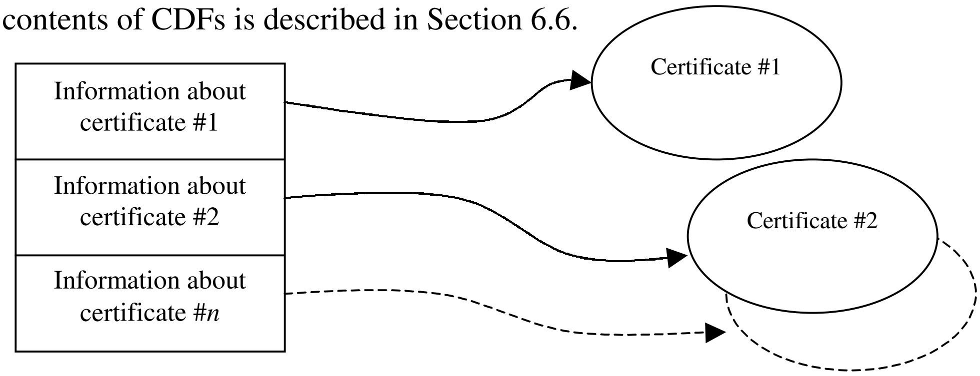


Figure 9 – EF(CDF) contains certificate attributes and pointers to the certificates

5.5.6 Data Object Directory Files (DODFs)

These files can be regarded as directories of data objects (other than keys or certificates) known to the PKCS #15 application. They are optional, but at least one DODF must be present on an IC card which contains such data objects (or references to such data objects) known to the PKCS #15 application. They contain general data object attributes such as identifiers of the application to which the data object belongs, whether it is a private or public object, etc. Furthermore, they contain pointers to the data objects themselves. There can be any number of DODFs in a PKCS #15 DF, but it is anticipated that in the normal case there will be at most one. The data objects themselves may reside anywhere on the card. The ASN.1 syntax for the contents of DODFs is described in Section 6.7.

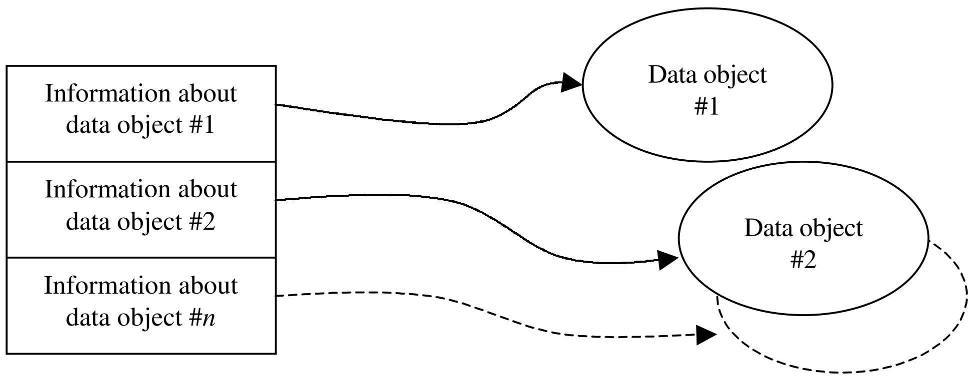


Figure 10 – EF(DODF) contains data object attributes and pointers to the data objects

5.5.7 Authentication Object Directory Files (AODFs)

These elementary files can be regarded as directories of authentication objects (e.g. PINs, passwords, biometric data) known to the PKCS #15 application. They are optional, but at least one AODF must be present on an IC card, which contains authentication objects restricting access to PKCS #15 objects. They contain generic authentication object attributes such as (in the case of PINs) allowed characters, PIN length, PIN padding character, etc. Furthermore, they contain pointers to the authentication objects themselves (e.g. in the case of PINs, pointers to the DF in which the PIN file resides). Authentication objects are used to control access to other objects such as keys. Information about which authentication object that protects a particular key is stored in the key's directory file, e.g. PrKDF (indicated in Figure 4, the rightmost arrow). There can be any number of AODFs in a PKCS #15 DF, but it is anticipated that in most cases there will only be one or two. The authentication objects themselves may reside anywhere on the card. The ASN.1 syntax for the contents of the AODFs is described in Section 6.8.

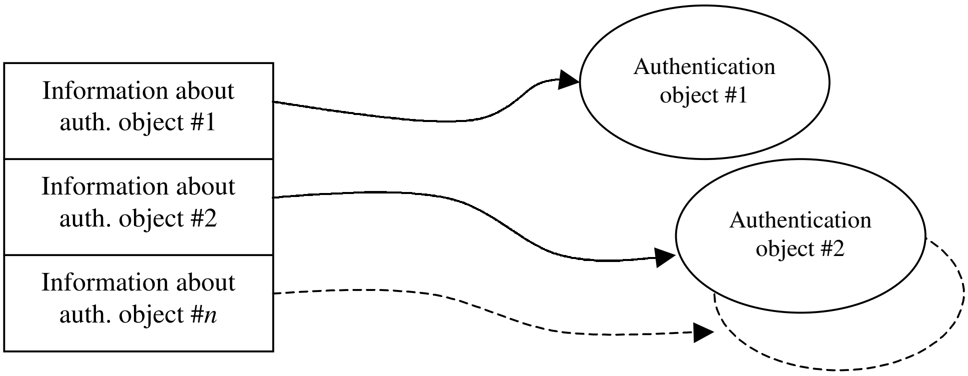


Figure 11 – EF(AODF) contains authentication object attributes and pointers to the authentication objects

5.5.8 EF(TokenInfo)

The mandatory TokenInfo elementary file with transparent structure shall contain generic information about the card as such and it's capabilities, as seen by the PKCS15 application. This information includes the card serial number, supported file types, algorithms implemented on the card, etc. The ASN.1 syntax for the contents of the TokenInfo file is described in detail in Section 6.9.

5.5.9 EF(UnusedSpace)

The optional UnusedSpace elementary file with transparent structure is used to keep track of unused space in already created elementary files. When present, it must initially contain at least one record pointing to an empty space in a file that is possible to update by the cardholder. The use of this file is described in more detail in Section 5.8. The file shall consist of DER-encoded records each with the following ASN.1 syntax:

```
UnusedSpace ::= SEQUENCE {
        path      Path (WITH COMPONENTS {..., index PRESENT, length PRESENT}),
        authId    Identifier OPTIONAL,
        ...,
        accessControlRules SEQUENCE OF AccessControlRule OPTIONAL
}
```

The **path** field points to an area (both **index**, i.e. offset, and **length** shall be present) that is unused and may be used when adding new objects to the card.

The **authID** component, described in more detail in Section 6.1.8, signals that the unused space is in a file modification-protected by a certain authentication object.

The **accessControlRules** component gives further information about access restriction in force for the indicated space. See further Section 6.1.8.

5.5.10 Other elementary files in the PKCS #15 directory

These (optional) files will contain the actual values of objects (such as private keys, public keys, secret keys, certificates and application specific data) referenced from within PrKDFs, SKDFs, PuKDFs, CDFs or DODFs. The ASN.1 format for the contents of these files follows from the ASN.1 descriptions in Section 6.

5.6 File identifiers

The following file identifiers are defined for the PKCS15 files. Note that the RID (see ISO/IEC 7816-5) is A0 00 00 00 63.

Table 1 – File Identifiers

File	**DF**	**File Identifier (relative to nearest DF)**
MF	X	$3F00_{16}$ (ISO/IEC 7816-4)
DIR		$2F00_{16}$ (ISO/IEC 7816-4)
PKCS15	X	Decided by application issuer (AID is RID ‖ "PKCS-15")
ODF		5031_{16} by default (but see also Section 6.4.1)
TokenInfo		5032_{16} by default (but see also Section 6.4.1)
UnusedSpace		5033_{16} by default (but see also Section 6.4.1)
AODFs		Decided by application issuer
PrKDFs		Decided by application issuer
PuKDFs		Decided by application issuer
SKDFs		Decided by application issuer
CDFs		Decided by application issuer
DODFs		Decided by application issuer
Other EFs		Decided by application issuer
- (Reserved)		5034_{16} - 5100_{16} (Reserved for future use)

5.7 The PKCS #15 application

5.7.1 PKCS #15 application selection

PKCS #15 compliant IC cards should support direct application selection as defined in ISO/IEC 7816-4 Section 9 and ISO/IEC 7816-5, Section 6 (the full AID is to be used as parameter for a '`SELECT FILE`' command). If direct application selection is not supported, or several PKCS #15 applications reside on the card, an EF(DIR) file with contents as specified in Section 5.4.1 must be used.

The operating system of the card must keep track of the currently selected application and only allow the commands applicable to that particular application while it is selected.

When several PKCS #15 applications resides on one card, they shall be distinguished by their object identifier in their application template in EF(DIR). It is recommended that the application label (tag **'50'H**) also be present to simplify the man-machine interface (e.g. vendor name in short form). See also Section 5.4.1.

5.7.2 AID for the PKCS #15 application

The Application Identifier (AID) data element consists of 12 bytes and its contents is defined below. The AID is used as the filename for DF(PKCS15) in order to facilitate direct selection of the PKCS #15 application on multi-application cards with only one PKCS #15 application present.

The AID is composed of RID || PIX, where '||' denotes concatenation. RID is the 5 byte globally "Registered Application Provider Identifier" as specified in ISO/IEC 7816-5. The RID shall be set to A0 00 00 00 63 for the purposes of this specification. This RID has been registered with ISO. PIX (Proprietary application Identifier eXtension) should be set to "`PKCS-15`".

The full AID for the current version of this document is thus A0 00 00 00 63 50 4B 43 53 2D 31 35.

5.8 Object management

5.8.1 Adding (Creating) new objects

The UnusedSpace file may be used to find suitable unused space on a card. After free space has been found, and assuming sufficient privileges to a suitable object directory file (e.g. a CDF in the case of a new certificate), the value of the new object is written to the area pointed to from EF(UnusedSpace). After this, the used record in EF(UnusedSpace) shall be updated to point to the first free byte after the newly written object. Finally, a new record is added to the object directory file. If the object directory file (e.g. CDF) is a true linear record file this will be a simple ISO/IEC 7816-4 command ('`APPEND RECORD`'). In the case of a transparent object directory file, an '`UPDATE BINARY`' command is suggested.

If no suitable free space can be found, garbage collection may be necessary, rewriting object directory files as the objects they point to moves around, and updating EF(UnusedSpace) in accordance.

If EF(UnusedSpace) is not being used, the application may have to create a new elementary file and write the value of the new object to this file before updating a suitable object directory file.

In the case of replacing a previous object, space can be conserved in the object directory file by updating the bytes previously used to hold information about that object. The space can be found by searching for a record with a '00' tag in the linear record file case, or a "logical" such record in the transparent file case. Since all records shall consist of DER-encoded values, these "empty" areas will be easy to find ('00' is not a valid ASN.1 tag). This method is also consistent with ISO/IEC 7816-4 annex D.

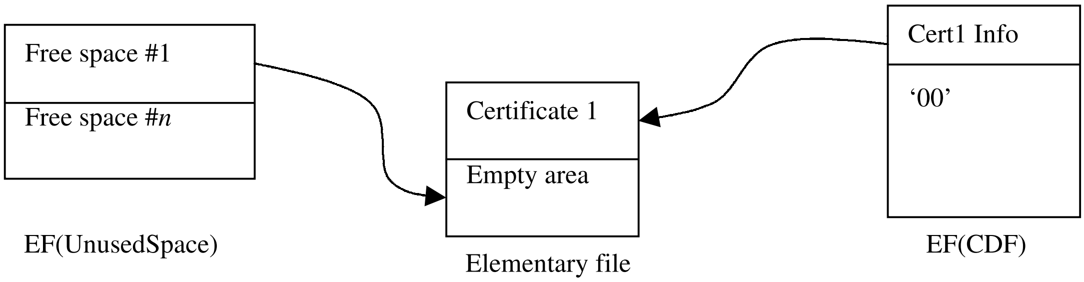


Figure 12 – Before adding a new certificate

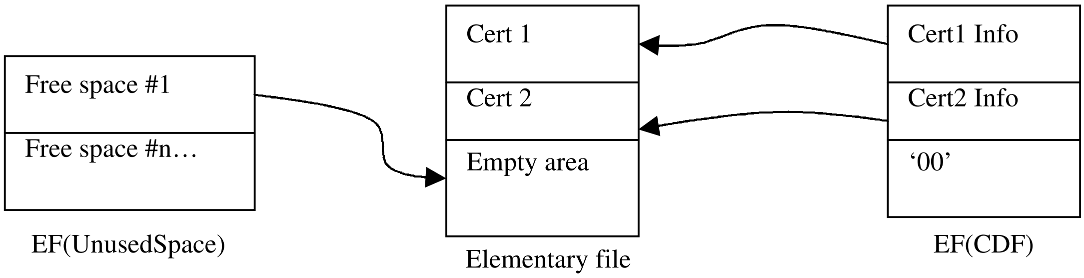


Figure 13 – After adding a new certificate

5.8.2 Removing objects

Once again, sufficient privileges are assumed. In particular, the object in question must be "modifiable" (see Section 6.1.8), and if it is a "private" object (again, see Section 6.1.8), authorization requirements must be met (e.g. a correct PIN must have been presented prior to the operation).

Removing a record from an object directory file is done by the '`WRITE RECORD`' or '`UPDATE RECORD`' command in the linear record file case, and by the '`WRITE BINARY`' or '`UPDATE BINARY`' command in the transparent file case. Records shall be erased be either replacing the outermost tag with a '00' byte or by re-writing the whole file with its new information content. Just overwriting the tag but preserving the length bytes allows for easy traversal of the file later on.

The following two figures shows an example in which a certificate is removed from the PKCS #15 application and EF(UnusedSpace) is used to keep track of unused space.

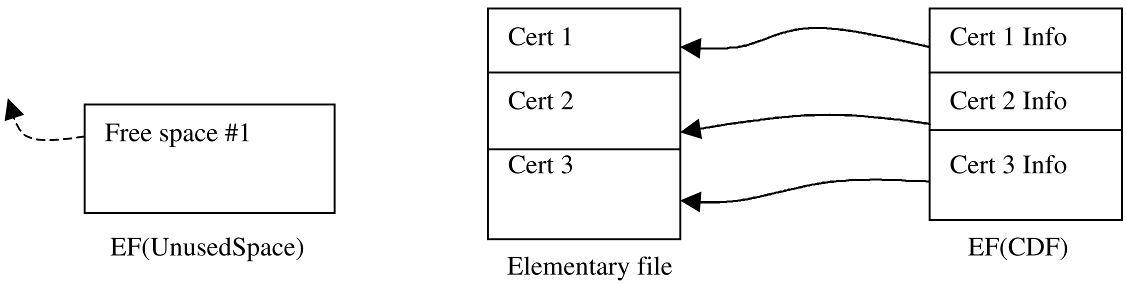


Figure 14 – Before removing certificate 2

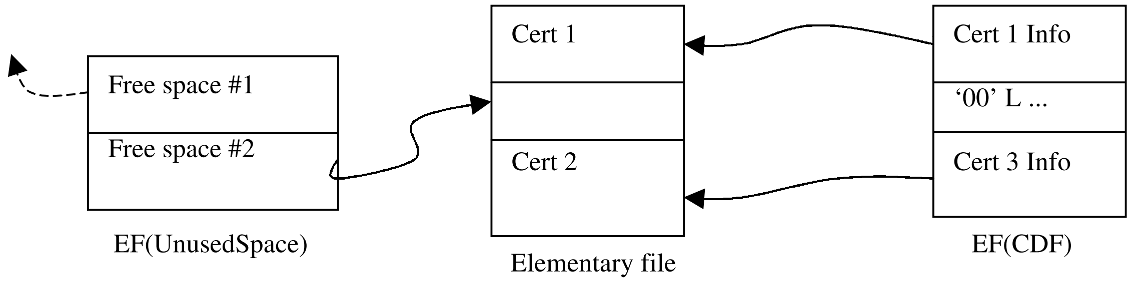


Figure 15 – After removing certificate 2

After having marked the entry in the object directory file as unused ('00'), a new record is added to EF(UnusedSpace), pointing to the area that the object directory file used to point to.

5.8.3 Modifying objects

Once again, sufficient privileges as in the previous subsection are assumed. In the linear record file case, the affected object directory file (e.g. EF(CDF), EF(DODF), etc) record is simply updated ('`UPDATE RECORD`'). In the transparent file case, if the encoding of the new information does not require more space than the previous information did, the (logical) record may be updated. Alternatively, the whole file may be re-written, but this may prove to be more costly.

6 Information syntax in ASN.1

NOTE – If nothing else is mentioned, DER-encoding of values is assumed.

6.1 Basic ASN.1 defined types

6.1.1 Identifier

Identifier ::= OCTET STRING (SIZE (0..pkcs15-ub-identifier))

The **Identifier** type is a constrained version of PKCS #11's CKA_ID. It is a card-internal identifier. For cross-reference purposes, two or more objects may have the same **Identifier** value. One example of this is a private key and one or more corresponding certificates.

6.1.2 Reference

Reference ::= INTEGER (0..pkcs15-ub-reference)

This type is used for generic reference purposes.

6.1.3 Label

Label ::= UTF8String (SIZE(0..pkcs15-ub-label))

This type is used for all labels (i.e. user assigned object names).

6.1.4 CredentialIdentifier

CredentialIdentifier {KEY-IDENTIFIER : IdentifierSet} ::= SEQUENCE {

```
        idType  KEY-IDENTIFIER.&id ({IdentifierSet}),
        idValue KEY-IDENTIFIER.&Value ({IdentifierSet}{@idType})
        }

KeyIdentifiers KEY-IDENTIFIER ::= {
        issuerAndSerialNumber           |
        issuerAndSerialNumberHash       |
        subjectKeyIdentifier            |
        subjectKeyHash                  |
        issuerKeyHash                   |
        issuerNameHash                  |
        subjectNameHash,
        ...
        }

KEY-IDENTIFIER ::= CLASS {
        &id  INTEGER UNIQUE,
        &Value
        } WITH SYNTAX {
        SYNTAX &Value IDENTIFIED BY &id
        }
```

The **CredentialIdentifier** type is used to identify a particular private key or certificate. There are currently seven members in the set of identifiers for private keys and certificates, **KeyIdentifiers**:

- **issuerAndSerialNumber**: The value of this type shall be a sequence of the issuer's distinguished name and the serial number of a certificate which contains the public key associated with the private key.
- **issuerAndSerialNumberHash**: As for **issuerAndSerialNumber**, but the value is an **OCTET STRING** which contains a SHA-1 hash value of this information in order to preserve space.
- **subjectKeyIdentifier**: The value of this type must be an **OCTET STRING** with the same value as the corresponding certificate extension in a X.509v3 certificate which contains the public key associated with the private key.
- **subjectPublicKeyHash**: An **OCTET STRING** which contains the SHA-1 hash of the public key associated with the private key. In the RSA case, the modulus of the public key shall be used, and the hash is to be done on the (network-order or big-endian) integer representation of it. The hash-algorithm shall be SHA-1. In the case of Elliptic Curves, it is recommended that the hash be calculated on the x-coordinate of the public key's **ECPoint OCTET STRING**. As an alternative, the hash can also be used as the **CommonKeyAttributes.iD.**

  NOTE – This is different from the hash method used e.g. in IETF RFC 2459 ([31]), but it serves the purpose of being independent of certificate format – alternative certificate formats not DER-encoding the public key has been proposed.
- **issuerKeyHash**: An **OCTET STRING** which contains the SHA-1 hash of the public key used to sign the requested certificate. This value can also, in the case of X.509 v3 certificates, be present in the **authorityKeyIdentifier** extension in the user's certificate, and the **subjectKeyIdentifier** extension in the issuer's certificate.

- **issuerNameHash**: A hash of the issuer's name as it appears in the certificate.
- **subjectNameHash**: A hash of the subject's name as it appears in the certificate.

6.1.5 ReferencedValue and Path

```
    ReferencedValue {Type} ::= CHOICE {
    path       Path,
    url        URL
    } (CONSTRAINED BY {-- 'path' or 'url' shall point to an object of type -- Type})

URL ::= CHOICE {
    url        PrintableString,
    urlWithDigest [3] SEQUENCE {
        url          IA5String,
        digest       Digest
        }
    }

Digest ::= SEQUENCE {
    digestAlgorithm      AlgorithmIdentifier {{DigestAlgorithms}} DEFAULT alg-id-sha1,
    digest               OCTET STRING (SIZE(8..128))
    }

Path ::= SEQUENCE {
    path       OCTET STRING,
    index      INTEGER (0..pkcs15-ub-index) OPTIONAL,
    length     [0] INTEGER (0..pkcs15-ub-index) OPTIONAL
    }( WITH COMPONENTS {..., index PRESENT, length PRESENT}|
    WITH COMPONENTS {..., index ABSENT, length ABSENT})
```

A **ReferencedValue** is a reference to a PKCS15 object value of some kind. This can either be some external reference (captured by the **url** choice) or a reference to a file on the card (the **path** identifier). In the **Path** case, identifiers **index** and **length** may specify a specific location within the file. If the file in question is a linear record file, **index** shall be the record number (in the ISO/IEC 7816-4 definition) and **length** can be set to **0** (if the card's operating system allows an L_e parameter equal to '`0`' in a '`READ RECORD`' command). Lengths of fixed records may be found in the **TokenInfo** file as well (see Section 6.9). In the **url** case, the URL may either be a simple URL or a URL in combination with a cryptographic hash of the object stored at the given location. Assuming that the PKCS #15 token is integrity-protected, the digest will protect the externally protected object as well.

If the file is a transparent file, then **index** can be used to specify an offset within the file, and **length** the length of the segment (**index** would then become parameter P_1 and/or P_2 and **length** the parameter L_e in a '`READ BINARY`' command). By using **index** and **length**, several objects may be stored within the same transparent file.

NOTE – On some IC cards which supports having several keys in one EF, keys are referenced by an identifier when used, but updating the EF requires knowledge of an offset and/or length of the data. In these cases, the **CommonKeyAttributes.keyReference** field shall be used for access to the key, and the presence of the **Path.index** and **Path.length** depends on the card issuer's discretion (they are not needed for card usage purposes, but may be used for modification purposes).

If **path** is two bytes long, it references a file by its file identifier. If **path** is longer than two bytes, it references a file either by an absolute or relative path (i.e. concatenation of file identifiers).

In the **url** case, the given URL must be in accordance with [5].

6.1.6 ObjectValue

```
ObjectValue { Type } ::= CHOICE {
indirect              ReferencedValue {Type},
direct                [0] Type,
indirect-protected    [1] ReferencedValue {EnvelopedData {Type}},
direct-protected      [2] EnvelopedData {Type},
}(CONSTRAINED BY {-- if indirection is being used, then it is expected that the reference
-- points either to a (possibly enveloped) object of type -- Type -- or (key case) to a card-
-- specific key file --})
```

The **ObjectValue** type is intended to catch the choice which can be made between storing a particular PKCS #15 object (key, certificate, etc) "in-line" (but possibly "enveloped", see Section 7) or by indirect reference (i.e. by pointing to another location where the value resides (possibly enveloped)). On tokens supporting the ISO/IEC 7816-4 logical file organization (i.e. EFs and DFs), the indirect alternative shall always be used. In other cases, any of the **CHOICE** alternatives may be used. Tokens not capable of protecting private objects by other means shall use the **indirect-protected** or the **direct-protected** choice.

6.1.7 PathOrObjects

```
PathOrObjects {ObjectType} ::= CHOICE {
path      Path,
objects   [0] SEQUENCE OF ObjectType,
...,
indirect-protected [1] ReferencedValue {EnvelopedData {SEQUENCE OF ObjectType}},
direct-protected [2] EnvelopedData {SEQUENCE OF ObjectType},
}
```

The **PathOrObjects** type is used to reference sequences of objects either residing within the ODF or externally. If the **path** alternative is used, then it is expected that the file pointed to by **path** contain the *value* part of an object of type **SEQUENCE OF ObjectType** (that is, the '**SEQUENCE OF**' tag and length shall not be present in the file). On tokens supporting the ISO/IEC 7816-4 logical file organization (i.e. EFs and DFs), the **path** alternative is strongly recommended. In other cases, any of the **CHOICE** alternatives may be used. The '**indirect-protected**' and '**direct-protected**' choices are intended for tokens not capable of protecting private objects by themselves, see Section 7.

6.1.8 CommonObjectAttributes

NOTE – This type is a container for attributes common to all PKCS #15 objects.

```
CommonObjectAttributes ::= SEQUENCE {
        label              Label OPTIONAL,
        flags              CommonObjectFlags OPTIONAL,
        authId             Identifier OPTIONAL,
        ...,
        userConsent        INTEGER (1..pkcs15-ub-userConsent) OPTIONAL,
        accessControlRules SEQUENCE SIZE (1..MAX) OF AccessControlRule OPTIONAL
        } (CONSTRAINED BY {-- authId should be present in the IC card case if flags.private is set. It
```

```
        -- must equal an authID in one AuthRecord in the AODF -- })

CommonObjectFlags ::= BIT STRING {
        private         (0),
        modifiable      (1)
        }

AccessControlRule ::= SEQUENCE {
        accessMode              AccessMode,
        securityCondition       SecurityCondition,
        … -- For future extensions
        }

AccessMode ::= BIT STRING {
        read     (0),
        update   (1),
        execute  (2)
        }

SecurityCondition ::= CHOICE {
        authId   Identifier,
        not[0]   SecurityCondition,
        and      [1] SEQUENCE SIZE (2..pkcs15-ub-securityConditions) OF SecurityCondition,
        or       [2] SEQUENCE SIZE (2..pkcs15-ub-securityConditions) OF SecurityCondition,
        ... -- For future extensions
        }
```

The **label** is the equivalent of the CKA_LABEL present in PKCS #11 ([29]), and is purely for display purposes (man-machine interface), for example when a user have several certificates for one key pair (e.g. "My bank certificate", "My S/MIME certificate").

The **flags** field indicates whether the particular object is private or not, and whether it is of type read-only or not. As in PKCS #11, a **private** object may only be accessed after proper authentication (e.g. PIN verification). If an object is marked as **modifiable**, it should be possible to update the value of the object. If an object is both **private** and **modifiable**, updating is only allowed after successful authentication, however. Since properties such as **private** and **modifiable** can be deduced by other means on IC cards, e.g. by studying EFs FCI, this field is optional and not necessary when these circumstances applies.

The **authId** field gives, in the case of a private object, a cross-reference back to the authentication object used to protect this object (For a description of authentication objects, see Section 5.5.7).

The **userConsent** field gives, in the case of a private object (or an object for which access conditions has been specified), the number of times an application may access the object without explicit consent from the user (e.g. a value of **3** indicates that a new authentication will be required before the first, the 4^{th}, the 7^{th}, etc. access). A value of **1** means that a new authentication is required before each access.

The **accessControlRules** field gives an alternative, and more fine-grained, way to inform a host-side applications about security conditions for various methods of accessing the object in question. Any Boolean expression in available authentication methods is allowed. When this field and the **authID** field both are present, information in this field takes precedence[1].

6.1.9 CommonKeyAttributes

```
CommonKeyAttributes ::= SEQUENCE {
        iD          Identifier,
        usage       KeyUsageFlags,
        native      BOOLEAN DEFAULT TRUE,
        accessFlags    KeyAccessFlags OPTIONAL,
        keyReference   Reference OPTIONAL,
        startDate      GeneralizedTime OPTIONAL,
        endDate        [0] GeneralizedTime OPTIONAL,
        ... -- For future extensions
        }

KeyUsageFlags ::= BIT STRING {
        encrypt             (0),
        decrypt             (1),
        sign                (2),
        signRecover         (3),
        wrap                (4),
        unwrap              (5),
        verify              (6),
        verifyRecover       (7),
        derive              (8),
        nonRepudiation      (9)
        }

KeyAccessFlags ::= BIT STRING {
        sensitive           (0),
        extractable         (1),
        alwaysSensitive     (2),
        neverExtractable    (3),
        local               (4)
        }
```

The **iD** field must be unique for each key stored in the card, except when a public key object and the corresponding private key object are stored on the card. In this case, the keys must share the same identifier (which may also be shared with a certificate object, see Section 6.1.14).

The **usage** field (**encrypt, decrypt, sign, signRecover, wrap, unwrap, verify, verifyRecover, derive** and **nonRepudiation**) signals the intended usage of the key as defined in PKCS #11. To map between X.509 ([26]) **keyUsage** flags for public keys, PKCS #15 flags for public keys, and PKCS #15 flags for private keys, use the following table:

[1] This can occur for backwards-compatibility reasons.

Table 2 – Mapping between PKCS #15 key usage flags and X.509 keyUsage extension flags

Key usage flags for public keys in X.509 public key certificates	**Corresponding PKCS #15 key usage flags for public keys**	**Corresponding PKCS #15 key usage flags for private keys**
DataEncipherment	Encrypt	Decrypt
DigitalSignature, keyCertSign, cRLSign	Verify	Sign
DigitalSignature, keyCertSign, cRLSign	VerifyRecover	SignRecover
KeyAgreement	Derive	Derive
KeyEncipherment	Wrap	Unwrap
NonRepudiation	NonRepudiation	NonRepudiation
NOTE – Implementations should verify that usage of key usage flags on a card is sound, i.e. that all key usage flags for a particular key pair is consistent with Table 2		

The **native** field identifies whether the card is able to use the key for hardware computations or not (e.g. this field is by default true for all RSA keys stored in special RSA key files on an RSA capable IC card).

The semantics of the **accessFlags** field's **sensitive**, **extractable**, **alwaysSensitive**, **neverExtractable** and **local** identifiers is the same as in PKCS #11. This field is not required to be present in cases where its value can be deduced by other means.

The **keyReference** field is only applicable for IC cards with cryptographic capabilities. If present, it contains a card-specific reference to the key in question (usually a small integer, for further information see ISO/IEC 7816-4 and ISO/IEC 7816-8 [15]).

The **startDate** and **endDate** fields have the same semantics as in PKCS #11.

6.1.10 CommonPrivateKeyAttributes

```
CommonPrivateKeyAttributes ::= SEQUENCE {
        subjectName    Name OPTIONAL,
        keyIdentifiers    [0] SEQUENCE OF CredentialIdentifier {{KeyIdentifiers}} OPTIONAL,
        ... -- For future extensions
        }

KeyIdentifier ::= SEQUENCE {
        idType    KEY-IDENTIFIER.&id ({KeyIdentifiers}),
        idValue   KEY-IDENTIFIER.&Value ({KeyIdentifiers}{@idType})
        }
```

The semantics for the fields of the **CommonPrivateKeyAttributes** type above is as follows:

The **subjectName** field, when present, shall contain the distinguished name of the owner of the private key, as specified in a certificate containing the public key corresponding to this private key.

The **keyIdentifiers** field: When receiving for example an enveloped message together with information about the public key used for encrypting the message's session key, the application needs to deduce which (if any) of the private keys present on the card that should be used for decrypting the session key. In messages based on the PKCS #7 ([28]) format, the **issuerAndSerialNumber** construct may be used, in other schemes other types may be used. This version of this document defines a number of possible ways to identifying a key (see Section 6.1.4).

6.1.11 CommonPublicKeyAttributes

```
CommonPublicKeyAttributes ::= SEQUENCE {
        subjectName    Name OPTIONAL,
        ...,
        trusted        [0] Usage OPTIONAL
        }
```

The semantics for the fields of the **CommonPublicKeyAttributes** type above is as follows:

The **subjectName** field, when present, shall contain the distinguished name of the owner of the public key as it appears in a certificate containing the public key.

The **trusted** field, which has no meaning in IC card cases (use the **trustedKeys** alternative instead; Section 6.2), indicates whether the cardholder trusts the public key in question for the indicated purposes (see Section 6.1.14).

NOTE – The exact semantics of this "trust" is outside the scope of this document.

6.1.12 CommonSecretKeyAttributes

This type contains all attributes common to PKCS #15 secret keys, except for **CommonKeyAttributes** and **PKCSCommonObjectAttributes**.

```
CommonSecretKeyAttributes ::= SEQUENCE {
        keyLen   INTEGER OPTIONAL, -- keylength (in bits)
        ... -- For future extensions
        }
```

The semantics for the fields of the **CommonSecretKeyAttributes** type above is as follows:

The optional **keyLen** field signals the key length used, in those cases where a particular algorithm can have a varying key length.

6.1.13 KeyInfo

This type, which is an optional part of each private and public key type, contains either (IC card case) a reference to a particular entry in the EF(TokenInfo) file, or explicit information about the key in question (parameters and operations supported by the card). The **supportedOperations** field is optional and can be absent on cards, which do not support any operations with the key. Note the distinction between **KeyUsageFlags** and **KeyInfo.paramsAndOps.supportedOperations**: The former indicates the intended *usage* of the key, the latter indicates the operations (if any) the *card* can *perform* with the key.

```
KeyInfo {ParameterType, OperationsType} ::= CHOICE {
        reference            Reference,
```

```
        paramsAndOps        SEQUENCE {
            parameters              ParameterType,
            supportedOperations     OperationsType OPTIONAL
                }
        }
```

6.1.14 CommonCertificateAttributes

```
CommonCertificateAttributes ::= SEQUENCE {
        iD              Identifier,
        authority       BOOLEAN DEFAULT FALSE,
        identifier      CredentialIdentifier {{KeyIdentifiers}} OPTIONAL,
        certHash        [0] OOBCertHash OPTIONAL,
        ...,
        trustedUsage    [1] Usage OPTIONAL,
        identifiers     [2] SEQUENCE OF CredentialIdentifier {{KeyIdentifiers}} OPTIONAL
        }

Usage ::= SEQUENCE {
        keyUsage        KeyUsage OPTIONAL,
        extKeyUsage     SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER OPTIONAL
        }(WITH COMPONENTS {..., keyUsage PRESENT} |
          WITH COMPONENTS {..., extKeyUsage PRESENT})
```

The **iD** field is only present for X.509 certificates in PKCS #11, but has for generality reasons been "promoted" to a common certificate attribute in this document. When a public key in the certificate in question corresponds to a private key also known to the PKCS #15 application, they must share the same value for the **iD** field. This requirement will simplify searches for a private key corresponding to a particular certificate and vice versa.

The **authority** field indicates whether the certificate is for an authority (i.e. CA or AA) or not.

The **identifier** (and, optionally, **identifiers)** field simplifies the search of a particular certificate, when the requester knows (and conveys) some distinguishing information about the requested certificate. This can be used, for example, when a user certificate has to be chosen and sent to a server as part of a user authentication, and the server provides the client with distinguishing information for a particular certificate. Use of the **subjectNameHash** and **issuerNameHash** alternatives may also facilitate fast chain building.

The **certHash** field is useful from a security perspective when the certificate in question is stored external to the card (the **url** choice of **ReferencedValue**), since it enables a user to verify that no one has tampered with the certificate.

NOTE – To find a token-holder certificate for a specific usage, use the **commonKeyAttributes.usage** field, and follow the cross-reference (**commonKeyAttributes.iD**) to an appropriate certificate.

The **trusted** field, which has no meaning in IC card cases (use the **trustedCertificates** alternative instead; Section 6.2), indicates whether the token-holder trusts the certificate in question or not, for the indicated purposes. Object identifiers could be ones defined for the **extKeyUsage** certificate extension, but may also be locally defined.

NOTE – The exact semantics of this "trust" is outside the scope of this document.

6.1.15 CommonDataObjectAttributes

```
CommonDataObjectAttributes ::= SEQUENCE {
        applicationName      Label OPTIONAL,
        applicationOID       OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        } (WITH COMPONENTS {..., applicationName PRESENT}|
        WITH COMPONENTS {..., applicationOID PRESENT})
```

The **applicationName** field is intended to contain the name or the registered object identifier for the application to which the data object in question "belongs". In order to avoid application name collisions, at least the **applicationOID** alternative is recommended. As indicated in ASN.1, at least one of the components has to be present in a value of type **CommonDataObjectAttributes**.

6.1.16 CommonAuthenticationObjectAttributes

```
CommonAuthenticationObjectAttributes ::= SEQUENCE {
        authId Identifier,
        ... –- For future extensions
        }
```

The **authId** must be a unique identifier. It is used for cross-reference purposes from private PKCS #15 objects.

6.1.17 PKCS15Object

This type is a template for all kinds of PKCS #15 objects. It is parameterized with object class attributes, object subclass attributes and object type attributes.

```
PKCS15Object {ClassAttributes, SubClassAttributes, TypeAttributes} ::= SEQUENCE {
        commonObjectAttributes   CommonObjectAttributes,
        classAttributes          ClassAttributes,
        subClassAttributes       [0] SubClassAttributes OPTIONAL,
        typeAttributes           [1] TypeAttributes
        }
```

6.2 PKCS15Objects

```
PKCS15Objects ::= CHOICE {
        privateKeys          [0] PrivateKeys,
        publicKeys           [1] PublicKeys,
        trustedPublicKeys    [2] PublicKeys,
        secretKeys           [3] SecretKeys,
        certificates         [4] Certificates,
        trustedCertificates  [5] Certificates,
        usefulCertificates   [6] Certificates,
        dataObjects          [7] DataObjects,
        authObjects          [8] AuthObjects,
        ... -- For future extensions
        }

PrivateKeys ::=   PathOrObjects {PrivateKeyType}

SecretKeys  ::=   PathOrObjects {SecretKeyType}

PublicKeys  ::=   PathOrObjects {PublicKeyType}

Certificates ::=  PathOrObjects {CertificateType}
```

```
DataObjects ::=    PathOrObjects {DataType}

AuthObjects ::=    PathOrObjects {AuthenticationType}
```

In the IC card case, the intention is that EF(ODF) shall consist of a number of data objects (records) of type **Objects**, representing different object types. Each data object should in the normal case reference a file containing a directory of objects of the particular type. Since the **path** alternative of the **PathOrObject** type is recommended, this will result in a record-oriented ODF, which simplifies updating.

NOTE – When it is known in advance that it will not be possible for the cardholder to modify e.g. EF(PrKDF) and EF(AODF), an application may store these files with the **direct** option of **PathOrObjects**

The **trustedPublicKeys** choice is intended for implementations on IC cards supporting the ISO/IEC 7816-4 logical file organization (non-IC card implementations should use the **publicKeys** alternative and the **trusted** field in **CommonPublicKeyAttributes**). In these cases, the card issuer might want to include a number of trusted public keys on the card (and make sure that they are not modified or replaced later on by an application). The PuKDF pointed to from this field should therefore be protected from cardholder modifications, as should the public keys pointed to from that PuKDF itself. Trusted public keys are most likely root CA keys that can be used as trust chain origins.

The **certificates** choice shall, in the case of IC cards supporting the ISO/IEC 7816-4 logical file organization, point to certificates issued to the cardholder. They may or may not be possible to modify by the cardholder.

The **trustedCertificates** choice is intended for implementations on IC cards supporting the ISO/IEC 7816-4 logical file organization (non-IC card implementations should use the **certificates** alternative and the **trusted** field in **CommonCertificateAttributes**). As for **trustedPublicKeys**, the card issuer might want to include a number of trusted certificates on the card (and make sure that they are not modified or replaced later on by an application), while still allowing the cardholder to add other certificates issued to himself/herself. The CDF pointed to from this field should therefore be protected from cardholder modifications, as should the certificates pointed to from that CDF itself. It is, however, conceivable that the card issuer can modify the contents of this file (and the files it points to). Trusted certificates are most likely root CA certificates, but does not have to be. Since the intention is that it should be impossible for a cardholder to modify them, they can be regarded as trusted by the cardholder. Root CA certificates included in this structure can therefore be used as trust chain origins.

The **usefulCertificates** choice is also intended for implementations on IC cards supporting the ISO/IEC 7816-4 logical file organization. The intention is that the cardholder may use this entry to store other end-entity or CA certificates that may be useful, e.g. signing certificates for colleagues, in order to simplify certificate path validation.

NOTE – In case of tokens not supporting the ISO/IEC 7816-4 logical file organization, implementers are recommended to use the **certificates** choice for all certificates (and the **publicKeys** choice for all public keys), and use the **trusted** attribute (see Section 6.1.11 and Section 6.1.14) to indicate which objects the tokenholder trusts.

6.3 Private keys

6.3.1 PrivateKeyObjects

This type contains information pertaining to private key objects stored in the card. Since, in the ISO/IEC 7816-4 IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **PrivateKeys** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of private keys, “Private Key Directory Files” (PrKDFs). The contents of an EF(PrKDF) must be the *value* of the DER encoding of a **SEQUENCE OF PrivateKeyType** (i.e. excluding the outermost tag and length bytes). This gives the PrKDFs the same, simple structure as the ODF, namely a number of TLV records.

In the case of cards not supporting the ISO/IEC 7816-4 logical file organization, any of the **CHOICE** alternatives of **PathOrObjects** may be used.

```
PrivateKeyType ::= CHOICE {
        privateRSAKey PrivateKeyObject {PrivateRSAKeyAttributes},
        privateECKey  [0] PrivateKeyObject {PrivateECKeyAttributes},
        privateDHKey  [1] PrivateKeyObject {PrivateDHKeyAttributes},
        privateDSAKey [2] PrivateKeyObject {PrivateDSAKeyAttributes},
        privateKEAKey [3] PrivateKeyObject {PrivateKEAKeyAttributes},
        ... -- For future extensions
        }

PrivateKeyObject {KeyAttributes} ::= PKCS15Object {
        CommonKeyAttributes, CommonPrivateKeyAttributes, KeyAttributes}
```

In other words, in the IC card case, each EF(PrKDF) shall consist of a number of context-tagged elements representing different private keys. Each private key element shall consist of a number of common object attributes (**CommonObjectAttributes**, **CommonKeyAttributes** and **CommonPrivateKeyAttributes**) and, in addition the particular key type’s attributes.

6.3.2 Private RSA key objects

```
PrivateRSAKeyAttributes ::= SEQUENCE {
        value               ObjectValue {RSAPrivateKey},
        modulusLength       INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo             KeyInfo {NULL, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

RSAPrivateKey ::= SEQUENCE {
        modulus             [0] INTEGER OPTIONAL, -- n
        publicExponent      [1] INTEGER OPTIONAL, -- e
        privateExponent     [2] INTEGER OPTIONAL, -- d
        prime1              [3] INTEGER OPTIONAL, -- p
        prime2              [4] INTEGER OPTIONAL, -- q
        exponent1           [5] INTEGER OPTIONAL, -- d mod (p-1)
        exponent2           [6] INTEGER OPTIONAL, -- d mod (q-1)
        coefficient         [7] INTEGER OPTIONAL -- inv(q) mod p
        } (CONSTRAINED BY {-- must be possible to reconstruct modulus and privateExponent from
selected fields --})
```

The semantics of the fields is as follows:

– **PrivateRSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **RSAPrivateKey** or (in the case of a card capable of performing on-chip RSA encryption) some card specific representation of a private RSA key. If there is no need to specify a path to a file, the path value may be set to **''H**, i.e. the empty path. As mentioned, the capability of on-chip private key operations will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative. Note that, besides the case of RSA capable IC cards, although the **RSAPrivateKey** type is very flexible, it is still constrained by the fact that it must be possible to reconstruct the modulus and the private exponent from whatever fields present.

  NOTE – If the private key is "linked" with a certificate, then it might be enough to store the private exponent here, since the modulus can be retrieved from the associated certificate.

– **PrivateRSAKeyAttributes.modulusLength**: On many cards, one must be able to format data to be signed prior to sending the data to the card. In order to be able to format the data in a correct manner the length of the key must be known. The length shall be expressed in bits, e.g. 1024.

– **PrivateRSAKeyAttributes.keyInfo:** Information about parameters that applies to this key (NULL in the case of RSA keys) and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.3.3 Private Elliptic Curve key objects

```
PrivateECKeyAttributes ::= SEQUENCE {
        value       ObjectValue {ECPrivateKey},
        keyInfo     KeyInfo {Parameters, PublicKeyOperations} OPTIONAL,
          ... -- For future extensions
        }

ECPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

– **PrivateECKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **ECPrivateKey** or (in the case of a card capable of performing on-chip EC operations) some card specific representation of a private EC key. If there is no need to specify a path to a file, the path value may be set to **''H**, i.e. the empty path. As mentioned, the capability of on-chip private key encryption will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PrivateECKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.3.4 Private Diffie-Hellman key objects

```
PrivateDHKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DHPrivateKey},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DHPrivateKey ::= INTEGER -- Diffie-Hellman exponent
```

The semantics of these types is as follows:

– **PrivateDHKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **DHPrivateKey** or (in the case of a card capable of performing on-chip Diffie-Hellman operations) some card specific representation of a private Diffie-Hellman key. If there is no need to specify a path to a file, the path value may be set to ''**H**, i.e. the empty path. As mentioned, the capability of on-chip private key operations will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PrivateDHKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.3.5 Private Digital Signature Algorithm key objects

```
PrivateDSAKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DSAPrivateKey},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DSAPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

– **PrivateDSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **DSAPrivateKey** or (in the case of a card capable of performing on-chip DSA operations) some card specific representation of a private DSA key. If there is no need to specify a path to a file, the path value may be set to ''**H**, i.e. the empty path. As mentioned, the capability of on-chip private key operations will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PrivateDSAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.3.6 Private KEA key objects

```
PrivateKEAKeyAttributes ::= SEQUENCE {
        value      ObjectValue {KEAPrivateKey},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

KEAPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

– **PrivateKEAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of the **KEAPrivateKey** type or (in the case of a card capable of performing on-chip KEA operations) some card specific representation of a private KEA key. If there is no need to specify a path to a file, the path value may be set to ‘’**H**, i.e. the empty path. As mentioned, the capability of on-chip private key operations will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PrivateKEAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall “point” to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.4 Public keys

6.4.1 The PublicKeys type

This data structure contains information pertaining to public key objects stored in the card. Since, in the IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **PublicKeys** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of public keys, “Public Key Directory Files” (PuKDFs). The contents of an EF(PuKDF) must be the *value* of the DER encoding of a **SEQUENCE OF PublicKeyType** (i.e. excluding the outermost tag and length bytes). This gives the PuKDFs the same, simple structure as the ODF, namely a number of TLV records.

In the case of cards not supporting the ISO/IEC 7816-4 logical file organization, any of the **CHOICE** alternatives of **PathOrObjects** may be used.

```
PublicKeyType ::= CHOICE {
        publicRSAKey  PublicKeyObject {PublicRSAKeyAttributes},
        publicECKey   [0] PublicKeyObject {PublicECKeyAttributes},
        publicDHKey   [1] PublicKeyObject {PublicDHKeyAttributes},
        publicDSAKey  [2] PublicKeyObject {PublicDSAKeyAttributes},
        publicKEAKey  [3] PublicKeyObject {PublicKEAKeyAttributes},
        … -- For future extensions
        }

PublicKeyObject {KeyAttributes} ::= PKCS15Object {
        CommonKeyAttributes, CommonPublicKeyAttributes, KeyAttributes}
```

In other words, in the IC card case, each EF(PuKDF) shall consist of a number of context-tagged elements representing different public keys. Each element shall consist of a number of common object attributes (**CommonObjectAttributes**, **CommonKeyAttributes** and **CommonPublicKeyAttributes**) and in addition the particular public key type's attributes.

6.4.2 Public RSA key objects

```
PublicRSAKeyAttributes ::= SEQUENCE {
        value               ObjectValue {RSAPublicKeyChoice},
        modulusLength       INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo             KeyInfo {NULL, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

RSAPublicKeyChoice ::= CHOICE {
        raw     RSAPublicKey,
        spki    [1] SubjectPublicKeyInfo, -- See X.509. Must contain a public RSA key
        ...
        }

RSAPublicKey ::= SEQUENCE {
        modulus             INTEGER, -- n
        publicExponent      INTEGER -- e
        }
```

The semantics of the fields is as follows:

- **PublicRSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **RSAPublicKey**, of type **SubjectPublicKeyInfo**, or (in the case of a card capable of performing on-chip RSA public-key encryption) some card specific representation of a public RSA key. As mentioned, this will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PublicRSAKeyAttributes.modulusLength**: On many cards, one must be able to format data to be encrypted prior to sending the data to the card. In order to be able to format the data in a correct manner the length of the key must be known. The length shall be expressed in bits, e.g. 1024.

- **PublicRSAKeyAttributes.keyInfo:** Information about parameters that applies to this key (**NULL** in the case of RSA keys) and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.4.3 Public Elliptic Curve key objects

```
PublicECKeyAttributes ::= SEQUENCE {
        value   ObjectValue {ECPublicKeyChoice},
        keyInfo KeyInfo {Parameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

ECPublicKeyChoice ::= CHOICE {
        raw     ECPoint,
        spki    SubjectPublicKeyInfo, -- See X.509. Must contain a public EC key
        ...
        }
```

The semantics of these types is as follows:

– **PublicECKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **ECPublicKey**, of type **SubjectPublicKeyInfo**, or (in the case of a card capable of performing on-chip EC public-key operations) some card specific representation of a public EC key. As mentioned, this will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PublicECKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.4.4 Public Diffie-Hellman key objects

```
PublicDHKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DHPublicKeyChoice},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DHPublicKeyChoice ::= CHOICE {
        raw        DiffieHellmanPublicNumber,
        spki       SubjectPublicKeyInfo, -- See X.509. Must contain a public D-H key
        ...
        }
```

The semantics of these types is as follows:

– **PublicDHKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **DHPublicKey**, of type **SubjectPublicKeyInfo**, or (in the case of a card capable of performing on-chip Diffie-Hellman public-key operations) some card specific representation of a public Diffie-Hellman key. As mentioned, this will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

– **PublicDHKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.4.5 Public Digital Signature Algorithm objects

```
PublicDSAKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DSAPublicKeyChoice},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DSAPublicKeyChoice ::= CHOICE {
        raw        INTEGER,
```

```
    spki      SubjectPublicKeyInfo, -- See X.509. Must contain a public DSA key.
    ...
    }
```

The semantics of these types is as follows:

- **PublicDSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **DSAPublicKey**, of type **SubjectPublicKeyInfo**, or (in the case of a card capable of performing on-chip DSA public-key operations) some card specific representation of a public DSA key. As mentioned, this will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.
- **PublicDSAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.4.6 Public KEA key objects

```
PublicKEAKeyAttributes ::= SEQUENCE {
    value     ObjectValue {KEAPublicKeyChoice},
    keyInfo   KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
    ... -- For future extensions
    }

KEAPublicKeyChoice ::= CHOICE {
    raw       INTEGER,
    spki      SubjectPublicKeyInfo, -- See X.509. Must contain a public KEA key
    ...
    }
```

The semantics of these types is as follows:

- **PublicKEAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **KEAPublicKey**, of type **SubjectPublicKeyInfo**, or (in the case of a card capable of performing on-chip KEA public-key operations) some card specific representation of a public KEA key. As mentioned, this will be indicated in the **CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.
- **PublicKEAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the card can carry out with this key. In the IC card case, the **reference** alternative of a **KeyInfo** must be used, and the reference shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

6.5 Secret keys

6.5.1 The SecretKeys type

This data structure contains information pertaining to secret keys stored in the card. Since, in the IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **SecretKeys** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of secret keys, "Secret Key Directory Files" (SKDFs). The contents of an EF(SKDF) must be the *value* of the DER encoding of a **SEQUENCE OF SecretKeyType** (i.e. excluding the outermost tag and length bytes). This gives the SKDFs the same, simple structure as the ODF, namely a number of TLV records.

In the case of cards not supporting the ISO/IEC 7816-4 logical file organization, any of the **CHOICE** alternatives of **PathOrObjects** may be used.

```
SecretKeyType ::= CHOICE {
        genericSecretKey    SecretKeyObject {GenericSecretKeyAttributes},
        rc2key              [0] SecretKeyObject {GenericSecretKeyAttributes},
        rc4key              [1] SecretKeyObject {GenericSecretKeyAttributes},
        desKey              [2] SecretKeyObject {GenericSecretKeyAttributes},
        des2Key             [3] SecretKeyObject {GenericSecretKeyAttributes},
        des3Key             [4] SecretKeyObject {GenericSecretKeyAttributes},
        castKey             [5] SecretKeyObject {GenericSecretKeyAttributes},
        cast3Key            [6] SecretKeyObject {GenericSecretKeyAttributes},
        cast128Key          [7] SecretKeyObject {GenericSecretKeyAttributes},
        rc5Key              [8] SecretKeyObject {GenericSecretKeyAttributes},
        ideaKey             [9] SecretKeyObject {GenericSecretKeyAttributes},
        skipjackKey         [10] SecretKeyObject {GenericSecretKeyAttributes},
        batonKey            [11] SecretKeyObject {GenericSecretKeyAttributes},
        juniperKey          [12] SecretKeyObject {GenericSecretKeyAttributes},
        rc6Key              [13] SecretKeyObject {GenericSecretKeyAttributes},
        otherKey            [14] OtherKey,
        ... -- For future extensions
        }

SecretKeyObject {KeyAttributes} ::= PKCS15Object {
         CommonKeyAttributes, CommonSecretKeyAttributes, KeyAttributes}

OtherKey ::= SEQUENCE {
        keyType  OBJECT IDENTIFIER,
        keyAttr  SecretKeyObject {GenericSecretKeyAttributes}
        }
```

In other words, in the IC card case, each EF(SKDF) shall consist of a number of context-tagged elements representing different secret keys. Each element shall consist of a number of common object attributes (**CommonObjectAttributes, CommonKeyAttributes** and **CommonSecretKeyAttributes**) and in addition the particular secret key type's attributes. All key types defined in this version correspond to key types defined in PKCS #11, and they all contain the same attributes, **GenericSecretKeyAttributes**, defined below.

6.5.2 Generic secret key objects

These objects represent generic keys, available for use in various algorithms, or for derivation of other secret keys.

```
GenericSecretKeyAttributes ::= SEQUENCE {
```

```
value       ObjectValue { OCTET STRING },
... -- For future extensions
}
```

The semantics of the field is as follows:

– **GenericSecretKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing an **OCTET STRING**. In other cases, the application issuer is free to choose any alternative offered by the **ObjectValue** type.

6.5.3 Tagged key objects

These key objects represent keys of various types. In the case of cards capable of performing cryptographic computations with keys of certain types, the key representation is card specific (indicated by the **CommonKeyAttributes.native** field). Otherwise, the key shall be stored as an **OCTET STRING**, as indicated above.

6.5.4 The PKCS15OtherKey type

This choice is intended to be a "catch-all" case, a placeholder for keys for which the algorithm is not already represented by a defined tag. The **OtherKey** type shall contain an object identifier identifying the type of the key and the usual secret key attributes.

6.6 Certificates

6.6.1 The Certificates type

This data structure contains information pertaining to certificate objects stored in the card. Since, in the IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **Certificates** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of certificates, "Certificate Directory Files" (CDFs). The contents of an EF(CDF) must be the *value* of the DER encoding of a **SEQUENCE OF CertificateType** (i.e. excluding the outermost tag and length bytes). This gives the CDFs the same, simple structure as the ODF, namely a number of TLV records.

In the case of cards not supporting the ISO/IEC 7816-4 logical file organization, any of the **CHOICE** alternatives of **PathOrObjects** may be used.

```
CertificateType ::= CHOICE {
        x509Certificate               CertificateObject { X509CertificateAttributes},
        x509AttributeCertificate      [0] CertificateObject {X509AttributeCertificateAttributes},
        spkiCertificate               [1] CertificateObject {SPKICertificateAttributes},
        pgpCertificate                [2] CertificateObject {PGPCertificateAttributes},
        wtlsCertificate               [3] CertificateObject {WTLSCertificateAttributes},
        x9-68Certificate              [4] CertificateObject {X9-68CertificateAttributes},
        ...,
        cvCertificate                 [5] CertificateObject {CVCertificateAttributes}
        }

CertificateObject {CertAttributes} ::= PKCS15Object {
        CommonCertificateAttributes, NULL, CertAttributes}
```

In other words, in the IC card case, each EF(CDF) shall consist of a number of context-tagged elements representing different certificate objects. Each element shall consist of a number of common object attributes (**CommonObjectAttributes** and **CommonCertificateAttributes**) and in addition the particular certificate type's attributes.

6.6.2 X.509 certificate objects

```
X509CertificateAttributes ::= SEQUENCE {
        value           ObjectValue { Certificate },
        subject         Name OPTIONAL,
        issuer          [0] Name OPTIONAL,
        serialNumber    CertificateSerialNumber OPTIONAL,
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **X509CertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** either identifying a file containing a DER encoded certificate at the given location, or a url pointing to some location where the certificate in question can be found. In other cases, the application issuer is free to choose any alternative.
- **X509CertificateAttributes.subject, X509CertificateAttributes.issuer** and **X509CertificateAttributes.serialNumber**: The semantics of these fields is the same as for the corresponding fields in PKCS #11. The reason for making them optional is to provide some space-efficiency, since they already are present in the certificate itself.

6.6.3 X.509 attribute certificate objects

```
X509AttributeCertificateAttributes ::= SEQUENCE {
        value       ObjectValue { AttributeCertificate },
        issuer      GeneralNames OPTIONAL,
        serialNumber    CertificateSerialNumber OPTIONAL,
        attrTypes       [0] SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **X509AttributeCertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing a DER encoded attribute certificate at the given location, or a url pointing to some location where the attribute certificate in question can be found. In other cases, the application issuer is free to choose any alternative.
- **X509AttributeCertificateAttributes.issuer** and **X509AttributeCertificateAttributes.serialNumber:** The values of these fields should be exactly the same as for the corresponding fields in the attribute certificate itself. They may be stored explicitly for easier lookup.
- **X509AttributeCertificateAttributes.attrTypes:** This optional field shall, when present, contain a list of object identifiers for the attributes that are present in this attribute certificate. This offers an opportunity for applications to search for a particular attribute certificate without downloading and parsing the certificate itself.

6.6.4 SPKI (Simple Public Key Infrastructure) certificate objects

NOTE – SPKI Certificates are defined in IETF RFC 2693 ([7]).

```
SPKICertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

– **SPKICertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing a SPKI certificate at the given location, or a url pointing to some location where the certificate can be found. In other cases, the application issuer is free to choose any alternative.

6.6.5 PGP (Pretty Good Privacy) certificate objects

NOTE – PGP Certificates are defined in IETF RFC 2440 ([6]).

```
PGPCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

– **PGPCertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing a PGP certificate at the given location, or a url pointing to some location where the certificate can be found. In other cases, the application issuer is free to choose any alternative.

6.6.6 WTLS certificate objects

NOTE – WTLS Certificates are defined in the "Wireless Transport Layer Security Protocol" specification ([32]).

```
WTLSCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

– **WTLSCertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing a WTLS encoded certificate at the given location, or a url pointing to some location where the certificate in question can be found. In other cases, the application issuer is free to choose any alternative.

6.6.7 ANSI X9.68 lightweight certificate objects

NOTE – X9.68 certificates are currently being defined by ANSI X9F1.

```
X9-68CertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

− **X9-68CertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing a DER or PER ([23]) encoded ANSI X9.68 certificate at the given location, or a url pointing to some location where the certificate in question can be found. In other cases, the application issuer is free to choose any alternative.

6.6.8 Card Verifiable Certificate objects

NOTE – Card Verifiable Certificates are defined in ISO/IEC 7816-8. Thanks to their simple structure, they can be interpreted by e.g. an IC card. Their main use is in public-key based card authentication methods.

```
CVCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type},
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

− **CVCertificateAttributes.value**: The value shall, in the IC card case, be a **ReferencedValue** identifying either a file containing an ISO/IEC 7816-8 Card Verifiable Certificate at the given location, or a URL pointing to some location where the certificate in question can be found. In other cases, the application issuer is free to choose any alternative.

6.7 Data objects

6.7.1 The DataObjects type

This data structure contains information pertaining to data objects stored in the card. Since, in the IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **DataObjects** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of data objects, "Data Object Directory Files" (DODFs). The contents of an EF(DODF) must be the *value* of the DER encoding of a **SEQUENCE OF DataType** (i.e. excluding the outermost tag and length bytes). This gives the DODFs the same, simple structure as the ODF, namely a number of TLV records.

In the case of cards not supporting the ISO/IEC 7816-4 logical file organization, any of the **CHOICE** alternatives of **PathOrObjects** may be used.

```
DataType ::= CHOICE {
        opaqueDO      DataObject {Opaque},
        externalIDO   [0] DataObject {ExternalIDO},
        oidDO         [1] DataObject {OidDO},
        ... -- For future extensions
        }

DataObject {DataObjectAttributes} ::= PKCS15Object {
        CommonDataObjectAttributes, NULL, DataObjectAttributes}
```

In other words, in the IC card case, DODFs shall consist of a number of context-tagged elements representing different data objects. Each element shall consist of a number of common object attributes (**CommonObjectAttributes** and **CommonDataObjectAttributes**) and in addition the particular data object type's attributes.

6.7.2 Opaque data objects

Opaque data objects are the least specified data objects. PKCS #15 makes no interpretation of these objects at all; it is completely left to applications accessing these objects.

```
Opaque ::= ObjectValue {PKCS15-OPAQUE.&Type}
```

6.7.3 External data objects

As an alternative, the DODF may contain information about one or several externally defined inter-industry data objects. These objects must follow a compatible tag allocation scheme as defined in Section 4.4 of ISO/IEC 7816-6.

```
ExternalIDO ::= ObjectValue {PKCS15-OPAQUE.&Type}
        (CONSTRAINED BY {-- All data objects must be defined in accordance with ISO/IEC 7816-6 --})
```

In the IC card case, each **externalIDO** entry in EF(DODF) will therefore point to a file which must conform to ISO/IEC 7816-6. By using these data objects, applications enhance interoperability.

6.7.4 Data objects identified by OBJECT IDENTIFIERS

This type provides a way to store, search and retrieve data objects with assigned object identifiers. An example of this type of information is any ASN.1 **ATTRIBUTE**.

```
OidDO ::= SEQUENCE {
        id      OBJECT IDENTIFIER,
        value   ObjectValue {PKCS15-OPAQUE.&Type}
        }
```

6.8 Authentication objects

6.8.1 The AuthenticationObjects type

This data structure, only relevant to cards capable of authenticating card-holders, contains information about how the card-holder authentication shall be carried out. Since, in the IC card case, the **path** alternative of the **PathOrObjects** type is to be chosen, **AuthenticationObjects** entries (records) in EF(ODF) points to elementary files that can be regarded as directories of authentication objects, "Authentication Object Directory Files" (AODFs). The contents of an EF(AODF) must be the *value* of the DER encoding of a **SEQUENCE OF AuthenticationType** (i.e. excluding the outermost tag and length bytes). This gives the AODFs the same, simple structure as the ODF, namely a number of TLV records.

```
AuthenticationType ::= CHOICE {
        pin                 AuthenticationObject { PinAttributes },
        ...,
        biometricTemplate   [0] AuthenticationObject{ BiometricAttributes},
        authKey             [1] AuthenticationObject {AuthKeyAttributes},
        external            [2] AuthenticationObject {ExternalAuthObjectAttributes}
```

```
    }
AuthenticationObject {AuthObjectAttributes} ::= PKCS15Object {
    CommonAuthenticationObjectAttributes, NULL, AuthObjectAttributes}
```

In other words, in the IC card case, each EF(AODF) shall consist of a number of context-tagged elements representing different authentication objects. Each element shall consist of a number of common object attributes (**CommonObjectAttributes** and **CommonAuthenticationObjectAttributes**) and in addition the particular authentication object type's attributes. Each authentication object must have a distinct **CommonAuthenticationObjectAttributes.authID**, enabling unambiguous authentication object lookup for private objects.

6.8.2 Pin objects

```
PinAttributes ::= SEQUENCE {
    pinFlags        PinFlags,
    pinType         PinType,
    minLength       INTEGER (pkcs15-lb-minPinLength..pkcs15-ub-minPinLength),
    storedLength    INTEGER (0..pkcs15-ub-storedPinLength),
    maxLength       INTEGER OPTIONAL,
    pinReference    [0] Reference DEFAULT 0,
    padChar         OCTET STRING (SIZE(1)) OPTIONAL,
    lastPinChange   GeneralizedTime OPTIONAL,
    path            Path OPTIONAL,
    ... -- For future extensions
    }

PinFlags ::= BIT STRING {
    case-sensitive              (0),
    local                       (1),
    change-disabled             (2),
    unblock-disabled            (3),
    initialized                 (4),
    needs-padding               (5),
    unblockingPin               (6),
    soPin                       (7),
    disable-allowed             (8),
    integrity-protected         (9),
    confidentiality-protected   (10),
    exchangeRefData             (11)
    } (CONSTRAINED BY { -- 'unblockingPin' and 'soPIN' cannot both be set -- })

PinType ::= ENUMERATED {bcd, ascii-numeric, utf8, ..., half-nibble-bcd, iso9564-1}
```

The semantics of these fields is as follows:

– **PinAttributes.pinFlags**: This field signals whether the PIN:

  – is **case-sensitive**, meaning that a user-given PIN shall *not* be converted to all-uppercase before presented to the card (see below);

  – is **local**, meaning that the PIN is local to the application to which it belongs;

  NOTE – A pin, which is not "local," is considered "global". A local PIN may only be used to protect data within a given application. For a local PIN the lifetime of verification is not guaranteed and it may have to be re-verified on each use. In contrast to this, a successful verification of a global PIN means that the verification remains in effect until the card has been removed or reset, or until a new verification of the same PIN fails. An application, which has verified a global PIN, can assume that the PIN remains valid, even if other applications verify their own, local PINs, select other DFs, etc.

  - is **change-disabled**, meaning that it is not possible to change the PIN;
  - is **unblock-disabled**, meaning that it is not possible to unblock the PIN;
  - is **initialized**, meaning that the PIN has been initialized;
  - **needs-padding**, meaning that, depending on the length of the given PIN and the stored length, the PIN may need to be padded before being presented to the card;
  - is an **unblockingPin** (ISO/IEC 7816-8: *resetting code*), meaning that this PIN may be used for unblocking purposes, i.e. to reset the retry counter of the related authentication object to its initial value;
  - is a **soPin**, meaning that the PIN is a Security Officer PIN (in the PKCS #11 sense);

    NOTE – Since PINs are PKCS #15 objects they may be protected by other authentication objects. This gives a way to specify the PIN that can be used to unblock another PIN - let the **authID** of a PIN point to an unblocking PIN

  - is **disable-allowed**, meaning that the PIN might be disabled;
  - shall be presented to the card with secure messaging (**integrity-protected**);
  - shall be presented to the card encrypted (**confidentiality-protected**);
  - can be changed by just presenting new reference data to the token or if both old and new reference data needs to be presented. If the bit is set, both old and new reference data shall be presented; otherwise only new reference data needs to be presented (**exchangeRefData**).

- **PinAttributes.pinType**: This field determines the type of PIN:
  - **bcd** (Binary Coded Decimal, each nibble of a byte shall contain one digit of the PIN);
  - **ascii-numeric** (Each byte of the PIN contain an ASCII [1] encoded digit); or
  - **utf8** (Each character is encoded in accordance with UTF8 [33]).
  - **half-nibble-bcd** (lower nibble of a byte shall contain one digit of the PIN, upper nibble shall contain **'F'H.**
  - **iso9564-1** (Encoding in accordance with ISO 9564-1 ([24]).
- **PinAttributes.minLength**: Minimum length (in characters) of new PINs (if allowed to change).

– **PinAttributes.storedLength**: Stored length on card (in bytes). Used to deduce the number of padding characters needed. Value can be set to **0** and disregarded if **pinFlags** indicate that padding is not needed (i.e. no padding characters are sent to the card).

– **PinAttributes.maxLength**: On some cards, PINs are not padded, and there is therefore a need to know the maximum PIN length (in characters) allowed.

– **PinAttributes.pinReference**: This field is a card-specific reference to the PIN in question. It is anticipated that it can be used as a 'P2' parameter in the ISO/IEC 7816-4 'VERIFY' command, when applicable. If not present, it defaults to the value **0**.

– **PinAttributes.padChar**: Padding character to use (usually 'FF_{16}' or '00_{16}'). Not needed if **pinFlags** indicates that padding is not needed for this card. If the **PinAttributes.pinType** is of type **bcd**, then **padChar** should consist of two nibbles of the same value, any nibble could be used as the "padding nibble". E.g., '55_{16}' is allowed, meaning padding with '0101_2', but '34_{16}' is illegal.

– **PinAttributes.lastPinChange**: This field is intended to be used in applications that requires knowledge of the date the PIN last was changed (e.g. to enforce PIN expiration policies). When the PIN is not set (or never has been changed) the value shall be (using the value-notation defined in ISO/IEC 8824-1) '**000000000000Z**'. As another example, a PIN changed on January 6, 1999 at 1934 (7 34 PM) UTC would have a **lastPinChange** value of '**19990106193400Z**'.

– **PinAttributes.path**: Path to the DF in which the PIN resides. The path shall be selected by a host application before doing a PIN operation, in order to enable a suitable authentication context for the PIN operation. If not present, a card-holder verification must always be possible to perform without a prior 'SELECT' operation.

6.8.2.1 Transforming a supplied PIN

The steps taken by a host-side application to transform a user-supplied PIN to something presented to the card shall be as follows:

1. Convert the PIN in accordance with the PIN type:

   a. If the PIN is a **utf8** PIN, transform it to UTF8: *x = UTF8(PIN).* Then, if the **case-sensitive** bit is off, convert x to uppercase: *x = NLSUPPERCASE(x)* (*NLSUPPERCASE* = locale dependent uppercase)

   b. If the PIN is a **bcd** PIN, verify that each character is a digit and encode the characters as BCD (see Section 2) digits: *x = BCD(PIN)*

   c. If the PIN is an **ascii-numeric** or **iso9564-1** PIN, verify that each character is a digit in the current code-page and –if needed– encode the characters as ASCII ([2]) digits: *x = ASCII(PIN)*

   d. If the PIN is a **half-nibble-bcd** PIN, verify that each character is a digit and encode the characters as BCD in the lower half of each byte, setting each upper nibble to 'F_{16}': *x = Half-BCD(PIN)*

2. If indicated in the **pinFlags** field, pad x to the right with the padding character, *padChar*, to stored length *storedLength*: $x = PAD(x, padChar, storedLength)$.
3. If the **pinFlags.integrity-protected** or **pinFlags.confidentiality-protected** bits are set, apply the appropriate algorithms and keys to the converted and formatted PIN.
4. Present the PIN to the card.

EXAMPLE – (ascii-) Numeric PIN '1234_{10}', stored length 8 bytes, and padding character 'FF_{16}' gives that the value presented to the card will be '$31323334FFFFFFFF_{16}$'

6.8.3 Biometric reference data objects

This type, only relevant to cards capable of authenticating card-holders by comparing a stored biometric template with a provided biometric reading, contains information about the stored biometric template.

```
BiometricAttributes ::= SEQUENCE {
        bioFlags        BiometricFlags,
        templateId      OBJECT IDENTIFIER,
        bioType         BiometricType,
        bioReference    Reference DEFAULT 0,
        lastChange      GeneralizedTime OPTIONAL,
        path            Path OPTIONAL,
        ... -- For future extensions
        }

BiometricFlags ::= BIT STRING {
        local                       (1),
        change-disabled             (2),
        unblock-disabled            (3),
        initialized                 (4),
        disable-allowed             (8),
        integrity-protected         (9),
        confidentiality-protected   (10)
        } -- Note: bits 0, 5, 6, and 7 are reserved for future use

BiometricType ::= CHOICE {
        fingerPrint             FingerPrint,
         irisScan               [0] IrisScan,
        -- Possible extensions:
        -- voiceScan            VoiceScan,
        -- faceScan             FaceScan,
        -- retinaScan           Retinascan,
        -- handGeometry         HandGeometry,
        -- writeDynamics        WriteDynamics,
        -- keyStrokeDynamics        KeyStrokeDynamics,
        -- lipDynamics          LipDynamics,
        ... -- For future extensions
        }

FingerPrint ::= SEQUENCE {
        hand    ENUMERATED {left, right},
        finger  ENUMERATED {thumb, pointerFinger, middleFinger, ringFinger, littleFinger},
        …
        }

IrisScan ::= SEQUENCE {
        eye     ENUMERATED {left, right},
```

```
    ...
    }
```

The semantics of these fields is as follows:

- **BiometricAttributes.bioFlags**: Same as for **PINAttributes.pinFlags**, but replace "PIN" with "biometrical reference data."
- **BiometricAttributes.templateId**: This field identifies the data structure that has to be sent to the card. It is anticipated that data structures will be registered with some international organization, by using **OBJECT IDENTIFIER**s.
- **BiometricAttributes.bioType**: This field determines the type of biometrical information stored in the card, e.g. the right pointer finger.
- **BiometricAttributes.bioReference,** **BiometricAttributes.lastChange**, and **BiometricAttributes.path**: As for corresponding fields in **PINAttributes**, but replace "PIN" with "biometrical reference data."

6.8.4 Authentication objects for external authentication

```
ExternalAuthObjectAttributes ::= CHOICE {
    authKeyAttributes     AuthKeyAttributes,
    certBasedAttributes  [0] CertBasedAuthenticationAttributes,
    ... -- For future extensions
    }

AuthKeyAttributes ::= SEQUENCE {
    derivedKey      BOOLEAN DEFAULT TRUE,
    authKeyId       Identifier,
    ... -- For future extensions
    }

CertBasedAuthenticationAttributes ::= SEQUENCE {
    cha         OCTET STRING,
    ...
    }
```

The semantics of these fields is as follows:

- **AuthKeyAttributes.derivedKey**: This field specifies whether the authentication key stored in the token is a derived key (i.e. an individual key), a group key, or a master key, used for deriving individual keys.
- **AuthKeyAttributes.authKeyId**: This field specifies the identifier (**CommonKeyAttribute.iD**) of the authentication key as described in an EF(SKDF).
- **CertBasedAuthenticationAttributes.cha**: This field specifies the certificate holder authorization as presented in a card-verifiable certificate (see ISO/IEC 7816-8 and –9). If a card-verifiable certificate containing this value is verified, and the authentication procedure with the corresponding key pair has been successfully completed, then the **cha** is set as valid, and access to private objects protected within this certificate-holder's authorization granted.

6.9 The cryptographic token information file, EF(TokenInfo)

This file contains general information about the PKCS #15 application and the token it resides on. It's data structure is defined as follows:

```
TokenInfo ::= SEQUENCE {
        version             INTEGER {v1(0)} (v1,...),
        serialNumber        OCTET STRING,
        manufacturerID      Label OPTIONAL,
        label               [0] Label OPTIONAL,
        tokenflags          TokenFlags,
        seInfo              SEQUENCE OF SecurityEnvironmentInfo OPTIONAL,
        recordInfo          [1] RecordInfo OPTIONAL,
        supportedAlgorithms[2] SEQUENCE OF AlgorithmInfo OPTIONAL,
        ...,
        issuerId            [3] Label OPTIONAL,
        holderId            [4] Label OPTIONAL,
        lastUpdate          GeneralizedTime OPTIONAL,
        preferredLanguage   PrintableString OPTIONAL – In accordance with IETF RFC 1766
        } (CONSTRAINED BY { -- Each AlgorithmInfo.reference value must be unique --})

TokenFlags ::= BIT STRING {
        readonly        (0),
        loginRequired   (1),
        prnGeneration   (2),
        eidCompliant    (3)
        }

SecurityEnvironmentInfo ::= SEQUENCE {
        se      INTEGER (0..127),
        owner   OBJECT IDENTIFIER,
        ... -- For future extensions
        }

RecordInfo ::= SEQUENCE {
        oDFRecordLength     [0] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        prKDFRecordLength   [1] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        puKDFRecordLength   [2] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        sKDFRecordLength    [3] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        cDFRecordLength     [4] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        dODFRecordLength    [5] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        aODFRecordLength    [6] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL
        }

AlgorithmInfo ::= SEQUENCE {
        reference           Reference,
        algorithm           PKCS15-ALGORITHM.&id({AlgorithmSet}),
        parameters          PKCS15-ALGORITHM.&Parameters({AlgorithmSet}{@algorithm}),
        supportedOperationsPKCS15-ALGORITHM.&Operations({AlgorithmSet}{@algorithm}),,
        algId               PKCS15-ALGORITHM.&objectIdentifier ({AlgorithmSet}{@algorithm}),
        algRef              Reference
        }
```

The interpretation of these fields shall be as follows:

– **TokenInfo.version:** This field contains the number of the particular version of this specification the application is based upon. For this version of this document, the value of **version** shall be 0 (**v1**).

- **TokenInfo.serialNumber:** This field shall contain the token's unique serial number, for IC card issued in accordance with ISO/IEC 7812-1 ([11]) and coded in accordance with ISO/IEC 8583 ([17]).
- **TokenInfo.manufacturerID**: This optional field shall, when present, contain identifying information about the token manufacturer (e.g. the card manufacturer), UTF8-encoded.
- **TokenInfo.label**: This optional field shall, when present, contain identifying information about the application.
- **TokenInfo.tokenflags**: This field contains information about the token *per se*. Flags include: If the token is read-only, if login (i.e. authentication) is required before accessing any data, if the token supports pseudo-random number generation and whether it conforms to the electronic identification profile of this specification, specified in Annex C.
- **TokenInfo.seInfo**: This optional field is intended to convey information about pre-set security environments on the token, and the owner of these environments. The definition of these environments is currently out of scope for this document, see further [15].
- **TokenInfo.recordInfo**: This optional field has two purposes:
  - to indicate whether the elementary files ODF, PrKDF, PuKDF, SKDF, CDF, DODF and AODF are linear record files or transparent files (if the field is present, they shall be linear record files, otherwise they shall be transparent files); and
  - if they are linear record files, whether they are of fixed-length or not (if they are of fixed length, corresponding values in **RecordInfo** are present and not equal to zero and indicates the record length. If some files are linear record files but not of fixed length, then corresponding values in **RecordInfo** can either be absent or set to zero.
- **TokenInfo.supportedAlgorithms**: The intent of this optional field is to indicate cryptographic algorithms, associated parameters, operations and algorithm input formats (e.g. PKCS #1 for digital signature input) supported by the card. The **reference** field of **AlgorithmInfo** is a unique reference that is used for cross-reference purposes from PrKDFs and PuKDFs. Values of the **supportedOperations** field (**compute-checksum**, **compute-signature**, **verify-checksum**, **verify-signature**, **encipher**, **decipher**, **hash** and **derive-key**) identifies operations the *card* can perform with a particular algorithm. The **algId** field indicates the object identifier for the algorithm in question. The **algRef** field indicates the identifier used by the card for denoting this algorithm (and. which occurs at the token interface as a parameter of, e.g., an `EXTERNAL AUTHENTICATE` command).
- **TokenInfo.issuerId**: This optional field shall, when present, contain identifying information about the token issuer (e.g. the card issuer).
- **TokenInfo.holderId**: This optional field shall, when present, contain identifying information about the token holder (e.g. the cardholder).

- **TokenInfo.lastUpdate**: This optional field shall, when present, contain the date of the last update of files in the PKCS #15 application. The presence of this field, together with the **TokenInfo.serialNumber** field, will enable host-side applications to quickly find out whether they have to read EF(ODF), EF(CDF), etc., or if they can used cached copies (if available).
- **TokenInfo.preferredLanguage:** The preferred language of the token holder, encoded in accordance with [1].

7 Software (Virtual card) format

7.1 Introduction

This section describes considerations to be made when implementing PKCS#15 on tokens not capable of protecting the integrity and confidentiality of PKCS#15 objects themselves. The typical case is when PKCS#15 is being implemented in software (so-called "virtual smart cards"). The format described in this section is an application of PKCS #7 v1.5 and IETF RFC 2630 ([9]).

Both private and public objects needs to be protected from unauthorized access. The solution chosen for PKCS#15 is a combination of integrity-protection and encryption. Private objects are to be encrypted, and after combining (encrypted) private and public objects in one data structure, the whole structure is optionally authenticated. Content encryption keys shall be session-keys (one-time use) and the session keys themselves shall be encrypted by long-term key-encryption keys and stored within the structure. Message authentication keys are also session-keys, encrypted with long-term key-encryption keys. Key-encryption keys may be derived from user passwords.

7.2 Useful types

7.2.1 The EnvelopedData type

This type is equivalent to the **EnvelopedData** type defined in PKCS #7 v1.5 and IETF RFC 2630, but has been parameterized here for better type checking.

```
EnvelopedData {Type} ::= SEQUENCE {
version                  INTEGER {v0(0), v1(1), v2(2), v3(3), v4(4)}(v0|v1|v2,...),
originatorInfo           [0] OriginatorInfo OPTIONAL,
recipientInfos           RecipientInfos,
encryptedContentInfo     EncryptedContentInfo{Type},
unprotectedAttrs         [1] SET SIZE (1..MAX) OF Attribute
}
```

version is the syntax version number. If any of the **RecipientInfo** structures included has a version other than **0** (see RFC 2630), then the version shall be **v2**, otherwise version shall be **v0**.

OriginatorInfo, **recipientInfos**, and **unprotectedAttrs** are as in IETF RFC 2630.

EncryptedContentInfo is described below.

7.2.2 The EncryptedContentInfo type

This type is equivalent to the **EncryptedContentInfo** type defined in PKCS #7 v1.5 and IETF RFC 2630, but has been parameterized here for better type checking.

```
EncryptedContentInfo {Type} ::= SEQUENCE {
contentType                    OBJECT IDENTIFIER,
contentEncryptionAlgorithm     AlgorithmIdentifier {{ContentEncryptionAlgorithms}},
encryptedContent               [0] OCTET STRING
}(CONSTRAINED BY {-- 'encryptedContent' shall be the result of encrypting DER-encoded
-- value of type – Type})
```

contentType shall always be set to **id-data**, since the type will always be known from the context in which the **EnvelopedData** occurs.

EncryptedContent shall be the result of encrypting a DER-encoded value of type **Type**.

7.3 The PKCS15Token type

```
PKCS15Token ::= SEQUENCE {
    version              INTEGER {v1(0)} (v1,...),
    preferredLanguage    PrintableString OPTIONAL, – In accordance with IETF RFC 1766
    keyManagementInfo    [0] KeyManagementInfo OPTIONAL,
    pkcs15Objects        SEQUENCE OF PKCS15Objects
    }

KeyManagementInfo ::= SEQUENCE OF SEQUENCE {
    keyId                Identifier,
    keyInfo              CHOICE {
        recipientInfo    RecipientIInfo,
        passwordInfo     [0] PasswordInfo
        }
    } (CONSTRAINED BY {-- Each keyID must be unique --})

PasswordInfo ::= SEQUENCE {
    hint     Label OPTIONAL,
    algId    AlgorithmIdentifier {{KeyDerivationAlgorithms}},
    ...
    } (CONSTRAINED BY {--keyID shall point to a KEKRecipientInfo--})
```

The intent is to collect all **PKCS15Objects** (whether "enveloped" or not) in a structure of type **PKCS15Token**. Enveloped objects may, in their **recipientInfos** field, use the **KEKRecipientInfo** choice (see IETF RFC 2630) to refer back to a key in the **KeyManagementInfo** "table." In the table, each entry has a unique **keyId**. The **passwordInfo** choice contains an optional **hint**, e.g. ("My bank key password"), and an algorithm identifier for a key derivation algorithm. Given the user's password, the key derivation algorithm is applied and the resulting key is used to decrypt the encrypted content-encryption key in the object's **KEKRecipientInfo**. This allows for protecting several objects with different content-encryption keys but the same password. The **recipientInfo** choice allows for recursive referencing, e.g. several **recipientInfos** may "point" to a **passwordInfo** value, but also for alternative, centralized, key-encryption methods. The **PKCS15Token.preferredLanguage** field is to be interpreted as in Section 6.9.

When using PKCS #15 soft-tokens in conjunction with CMS ([9]), the **contentType** for a plain **PKCS15Token** shall be **{pkcs15-ct 1}**.

If a **PKCS15Token** is stored in an environment in which its integrity cannot be guaranteed, it should be integrity-protected by wrapping it in an **AuthenticatedData** or **SignedData** structure (see [9]). The **AuthenticatedData.recipientInfos** (or **SignedData.recipientInfos**) field may or may not refer to a **keyId** in the wrapped **PKCS15Token.keyManagementInfo** table. If it does not, then it is assumed that a relying partner will use some out-of-band knowledge to find the corresponding key-encryption key.

NOTE – If stored within a directory, the **pKCS15Token** attribute, defined in PKCS #9 v2.0, may be used to represent the resulting type. PKCS #9 v2.0 also defines an object class, **pkcsEntity**, capable of carrying this attribute.

7.4 Permitted algorithms

7.4.1 Key derivation algorithms

Allowed key derivation functions (algorithms used to derive a key-encryption key from a user password) are defined in PKS#5 v2.0 ([27]). The set of allowed algorithms is thus:

```
KeyDerivationAlgorithms ALGORITHM-IDENTIFIER ::= {
        PBKDF2Algorithms,
        ... -- For future extensions
        }
```

7.4.2 Other algorithms

IETF RFC 2630 defines algorithms for encryption of session keys ("Symmetric Key-Encryption Algorithms"), for authenticity of information ("Message Authentication Code Algorithms"), for cryptographic hashing of data ("Digest Algorithms"), and for encryption of data ("Content Encryption Algorithms"). These algorithms are adapted for the same use in this specification. In addition, algorithms defined in PKCS #5 v2.0 for encryption of data are included in the **ContentEncryptionAlgorithms** set. The set of allowed algorithms is thus:

```
KeyEncryptionAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY id-alg-CMS3DESwrap} |
        {INTEGER IDENTIFIED BY id-alg-CMSRC2wrap},
        ... -- For future extensions
        }

ContentEncryptionAlgorithms ALGORITHM-IDENTIFIER ::= {
        SupportingAlgorithms EXCEPT {NULL IDENTIFIED BY id-hmacWithSHA1},
        -- SupportingAlgorithms are defined in PKCS #5 v2
        ... -- For future extensions
        }

MACAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY hMAC-SHA1},
        ... – For future extensions
        }

DigestAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY sha-1},
        ... -- For future extensions
        }
```

A. ASN.1 module

This section includes all ASN.1 type, value and information object class definitions contained in this document, in the form of the ASN.1 module **PKCS-15**.

```
        PKCS-15 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-15(15) modules(1) pkcs-
15(1)}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

IMPORTS

informationFramework, authenticationFramework, certificateExtensions
        FROM UsefulDefinitions {joint-iso-itu-t(2) ds(5) module(1)
        usefulDefinitions(0) 3}

Name, Attribute
        FROM InformationFramework informationFramework

Certificate, AttributeCertificate, CertificateSerialNumber, SubjectPublicKeyInfo
        FROM AuthenticationFramework authenticationFramework

GeneralNames, KeyUsage
        FROM CertificateExtensions certificateExtensions

RecipientInfos, RecipientInfo, OriginatorInfo, sha-1, id-alg-CMS3DESwrap, id-alg-CMSRC2wrap,
        hMAC-SHA1, des-ede3-cbc
        FROM CryptographicMessageSyntax {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
        pkcs-9(9) smime(16) modules(0) cms(1)}

AlgorithmIdentifier, SupportingAlgorithms, PBKDF2Algorithms, ALGORITHM-IDENTIFIER,
        id-hmacWithSHA1
        FROM PKCS-5 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-5(5) modules(16)
        pkcs-5(1)}

ECPoint, Parameters
        FROM ANSI-X9-62 {iso(1) member-body(2) us(840) ansi-x962(10045) module(4) 1}

DiffieHellmanPublicNumber, DomainParameters
        FROM ANSI-X9-42 {iso(1) member-body(2) us(840) ansi-x942(10046) module(5) 1}

OOBCertHash
        FROM PKIXCMP {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5)
        pkix(7) id-mod(0) id-mod-cmp(9)};

-- Constants
        pkcs15-ub-identifier            INTEGER ::= 255
        pkcs15-ub-reference             INTEGER ::= 255
        pkcs15-ub-index                 INTEGER ::= 65535
        pkcs15-ub-label                 INTEGER ::= pkcs15-ub-identifier
        pkcs15-lb-minPinLength          INTEGER ::= 4
        pkcs15-ub-minPinLength          INTEGER ::= 8
        pkcs15-ub-storedPinLength       INTEGER ::= 64
        pkcs15-ub-recordLength          INTEGER ::= 16383
        pkcs15-ub-userConsent           INTEGER ::= 15
        pkcs15-ub-securityConditions    INTEGER ::= 255
        pkcs15-ub-seInfo                INTEGER ::= 255

-- Object Identifiers

pkcs15      OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
```

```
                                        pkcs-15(15)}
      pkcs15-mo     OBJECT IDENTIFIER ::= {pkcs15 1} -- Modules branch
      pkcs15-at     OBJECT IDENTIFIER ::= {pkcs15 2} -- Attribute branch
      pkcs15-ct     OBJECT IDENTIFIER ::= {pkcs15 3} -- Content type branch

-- Content Types
      pkcs15-ct-PKCS15Token OBJECT IDENTIFIER ::= {pkcs15-ct 1}

-- Basic types

Identifier ::= OCTET STRING (SIZE (0..pkcs15-ub-identifier))

Reference ::= INTEGER (0..pkcs15-ub-reference)

Label ::= UTF8String (SIZE(0..pkcs15-ub-label))

-- Credential Identifiers

KEY-IDENTIFIER ::= CLASS {
      &id  INTEGER UNIQUE,
      &Value
      } WITH SYNTAX {
      SYNTAX &Value IDENTIFIED BY &id
      }

CredentialIdentifier {KEY-IDENTIFIER : IdentifierSet} ::= SEQUENCE {
      idType  KEY-IDENTIFIER.&id ({IdentifierSet}),
      idValue KEY-IDENTIFIER.&Value ({IdentifierSet}{@idType})
      }

KeyIdentifiers KEY-IDENTIFIER ::= {
      issuerAndSerialNumber           |
      issuerAndSerialNumberHash       |
      subjectKeyIdentifier            |
      subjectKeyHash                  |
      issuerKeyHash                   |
      issuerNameHash                  |
      subjectNameHash,
      ...
      }

issuerAndSerialNumber KEY-IDENTIFIER::=
      {SYNTAX PKCS15-OPAQUE.&Type IDENTIFIED BY 1}
      -- As defined in RFC 2630

subjectKeyIdentifier KEY-IDENTIFIER ::=
      {SYNTAX OCTET STRING IDENTIFIED BY 2}
      -- From x509v3 certificate extension

issuerAndSerialNumberHash KEY-IDENTIFIER ::=
      {SYNTAX OCTET STRING IDENTIFIED BY 3}
      -- Assumes SHA-1 hash of DER encoding of IssuerAndSerialNumber

subjectKeyHash KEY-IDENTIFIER ::=
      {SYNTAX OCTET STRING IDENTIFIED BY 4}

issuerKeyHash KEY-IDENTIFIER ::=
      {SYNTAX OCTET STRING IDENTIFIED BY 5}

issuerNameHash KEY-IDENTIFIER ::=
      {SYNTAX OCTET STRING IDENTIFIED BY 6}
      -- SHA-1 hash of DER-encoded issuer name

subjectNameHash KEY-IDENTIFIER ::=
```

```
        {SYNTAX OCTET STRING IDENTIFIED BY 7}
        -- SHA-1 hash of DER-encoded subject name
        ReferencedValue {Type} ::= CHOICE {
        path        Path,
        url         URL
        } (CONSTRAINED BY {-- 'path' or 'url' shall point to an object of type -- Type})

URL ::= CHOICE {
        url         PrintableString,
        urlWithDigest [3] SEQUENCE {
            url             IA5String,
            digest          Digest
            }
        }
        alg-id-sha1 AlgorithmIdentifier {{DigestAlgorithms}} ::= {
        algorithm       sha-1,
        parameters      SHA1Parameters : NULL}

SHA1Parameters ::= NULL

Digest ::= SEQUENCE {
        digestAlg       AlgorithmIdentifier {{DigestAlgorithms}} DEFAULT alg-id-sha1,
        digest          OCTET STRING (SIZE(8..128))
        }

Path ::= SEQUENCE {
        path        OCTET STRING,
        index       INTEGER (0..pkcs15-ub-index) OPTIONAL,
        length      [0] INTEGER (0..pkcs15-ub-index) OPTIONAL
        }( WITH COMPONENTS {..., index PRESENT, length PRESENT}|
        WITH COMPONENTS {..., index ABSENT, length ABSENT})

ObjectValue { Type } ::= CHOICE {
        indirect                ReferencedValue {Type},
        direct                  [0] Type,
        indirect-protected      [1] ReferencedValue {EnvelopedData {Type}},
        direct-protected        [2] EnvelopedData {Type}
        }(CONSTRAINED BY {-- if indirection is being used, then it is expected that the reference
        -- points either to a (possibly enveloped) object of type -- Type -- or (key case) to a card-
        -- specific key file --})

PathOrObjects {ObjectType} ::= CHOICE {
        path        Path,
        objects     [0] SEQUENCE OF ObjectType,
        ...,
        indirect-protected [1] ReferencedValue {EnvelopedData {SEQUENCE OF ObjectType}},
        direct-protected [2] EnvelopedData {SEQUENCE OF ObjectType}
        }

CommonObjectAttributes ::= SEQUENCE {
        label           Label OPTIONAL,
        flags           CommonObjectFlags OPTIONAL,
        authId          Identifier OPTIONAL,
        ...,
        userConsent     INTEGER (1..pkcs15-ub-userConsent) OPTIONAL,
        accessControlRules  SEQUENCE SIZE (1..MAX) OF AccessControlRule OPTIONAL
        } (CONSTRAINED BY {-- authId should be present in the IC card case if flags.private is set.
        -- It must equal an authID in one AuthRecord in the AODF -- })

CommonObjectFlags ::= BIT STRING {
        private         (0),
```

```
        modifiable      (1)
        }

AccessControlRule ::= SEQUENCE {
        accessMode              AccessMode,
        securityCondition       SecurityCondition,
        ... -- For future extensions
        }

AccessMode ::= BIT STRING {
        read    (0),
        update  (1),
        execute (2)
        }

SecurityCondition ::= CHOICE {
        authId  Identifier,
        not[0] SecurityCondition,
        and     [1] SEQUENCE SIZE (2..pkcs15-ub-securityConditions) OF SecurityCondition,
        or  [2] SEQUENCE SIZE (2..pkcs15-ub-securityConditions) OF SecurityCondition,
        ... -- For future extensions
        }

CommonKeyAttributes ::= SEQUENCE {
        iD              dentifier,
        usage           KeyUsageFlags,
        native          BOOLEAN DEFAULT TRUE,
        accessFlags     KeyAccessFlags OPTIONAL,
        keyReference    Reference OPTIONAL,
        startDate       GeneralizedTime OPTIONAL,
        endDate         [0] GeneralizedTime OPTIONAL,
        ... -- For future extensions
        }

KeyUsageFlags ::= BIT STRING {
        encrypt                 (0),
        decrypt                 (1),
        sign                    (2),
        signRecover             (3),
        wrap                    (4),
        unwrap                  (5),
        verify                  (6),
        verifyRecover           (7),
        derive                  (8),
        nonRepudiation          (9)
        }

KeyAccessFlags ::= BIT STRING {
        sensitive               (0),
        extractable             (1),
        alwaysSensitive         (2),
        neverExtractable        (3),
        local                   (4)
        }

CommonPrivateKeyAttributes ::= SEQUENCE {
        subjectName     Name OPTIONAL,
        keyIdentifiers  [0] SEQUENCE OF CredentialIdentifier {{KeyIdentifiers}} OPTIONAL,
        ... -- For future extensions
        }
```

```
KeyIdentifier ::= SEQUENCE {
        idType    KEY-IDENTIFIER.&id ({KeyIdentifiers}),
        idValue   KEY-IDENTIFIER.&Value ({KeyIdentifiers}{@idType})
        }

CommonPublicKeyAttributes ::= SEQUENCE {
        subjectName    Name OPTIONAL,
        ...,
        trusted        [0] Usage OPTIONAL
        }

CommonSecretKeyAttributes ::= SEQUENCE {
        keyLen    INTEGER OPTIONAL, -- keylength (in bits)
        ... -- For future extensions
        }

KeyInfo {ParameterType, OperationsType} ::= CHOICE {
        reference           Reference,
        paramsAndOps        SEQUENCE {
            parameters              ParameterType,
            supportedOperations     OperationsType OPTIONAL
            }
        }

CommonCertificateAttributes ::= SEQUENCE {
        iD              Identifier,
        authority       BOOLEAN DEFAULT FALSE,
        identifier      CredentialIdentifier {{KeyIdentifiers}} OPTIONAL,
        certHash        [0] OOBCertHash OPTIONAL,
        ...,
        trustedUsage    [1] Usage OPTIONAL,
        identifiers     [2] SEQUENCE OF CredentialIdentifier{{KeyIdentifiers}} OPTIONAL
        }

Usage ::= SEQUENCE {
        keyUsage        KeyUsage OPTIONAL,
        extKeyUsage     SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER OPTIONAL
        }(WITH COMPONENTS {..., keyUsage PRESENT} |
          WITH COMPONENTS {..., extKeyUsage PRESENT})

CommonDataObjectAttributes ::= SEQUENCE {
        applicationName     Label OPTIONAL,
        applicationOID      OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        } (WITH COMPONENTS {..., applicationName PRESENT}|
           WITH COMPONENTS {..., applicationOID PRESENT})

CommonAuthenticationObjectAttributes ::= SEQUENCE {
        authId Identifier,
        ... -- For future extensions
        }

PKCS15Object {ClassAttributes, SubClassAttributes, TypeAttributes} ::= SEQUENCE {
        commonObjectAttributes  CommonObjectAttributes,
        classAttributes         ClassAttributes,
        subClassAttributes      [0] SubClassAttributes OPTIONAL,
        typeAttributes          [1] TypeAttributes
        }

PKCS15Objects ::= CHOICE {
        privateKeys         [0] PrivateKeys,
        publicKeys          [1] PublicKeys,
```

```
        trustedPublicKeys     [2] PublicKeys,
        secretKeys            [3] SecretKeys,
        certificates          [4] Certificates,
        trustedCertificates   [5] Certificates,
        usefulCertificates    [6] Certificates,
        dataObjects           [7] DataObjects,
        authObjects           [8] AuthObjects,
        ... -- For future extensions
        }

PrivateKeys   ::= PathOrObjects {PrivateKeyType}

SecretKeys    ::= PathOrObjects {SecretKeyType}

PublicKeys    ::= PathOrObjects {PublicKeyType}

Certificates  ::= PathOrObjects {CertificateType}

DataObjects   ::= PathOrObjects {DataType}

AuthObjects   ::= PathOrObjects {AuthenticationType}

PrivateKeyType ::= CHOICE {
        privateRSAKey PrivateKeyObject {PrivateRSAKeyAttributes},
        privateECKey  [0] PrivateKeyObject {PrivateECKeyAttributes},
        privateDHKey  [1] PrivateKeyObject {PrivateDHKeyAttributes},
        privateDSAKey [2] PrivateKeyObject {PrivateDSAKeyAttributes},
        privateKEAKey [3] PrivateKeyObject {PrivateKEAKeyAttributes},
        ... -- For future extensions
        }

PrivateKeyObject {KeyAttributes} ::= PKCS15Object {
        CommonKeyAttributes, CommonPrivateKeyAttributes, KeyAttributes}

PrivateRSAKeyAttributes ::= SEQUENCE {
        value                 ObjectValue {RSAPrivateKey},
        modulusLength         INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo               KeyInfo {NULL, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

RSAPrivateKey ::= SEQUENCE {
        modulus               [0] INTEGER OPTIONAL, -- n
        publicExponent        [1] INTEGER OPTIONAL, -- e
        privateExponent       [2] INTEGER OPTIONAL, -- d
        prime1                [3] INTEGER OPTIONAL, -- p
        prime2                [4] INTEGER OPTIONAL, -- q
        exponent1             [5] INTEGER OPTIONAL, -- d mod (p-1)
        exponent2             [6] INTEGER OPTIONAL, -- d mod (q-1)
        coefficient           [7] INTEGER OPTIONAL -- inv(q) mod p
        } (CONSTRAINED BY {-- must be possible to reconstruct modulus and privateExponent from
        -- selected fields --})

PrivateECKeyAttributes ::= SEQUENCE {
        value     ObjectValue {ECPrivateKey},
        keyInfo   KeyInfo {Parameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

ECPrivateKey ::= INTEGER

PrivateDHKeyAttributes ::= SEQUENCE {
        value     ObjectValue {DHPrivateKey},
```

```
        keyInfo   KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DHPrivateKey ::= INTEGER -- Diffie-Hellman exponent

PrivateDSAKeyAttributes ::= SEQUENCE {
        value     ObjectValue {DSAPrivateKey},
        keyInfo   KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DSAPrivateKey ::= INTEGER

PrivateKEAKeyAttributes ::= SEQUENCE {
        value     ObjectValue {KEAPrivateKey},
        keyInfo   KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

KEAPrivateKey ::= INTEGER

PublicKeyType ::= CHOICE {
        publicRSAKey  PublicKeyObject {PublicRSAKeyAttributes},
        publicECKey   [0] PublicKeyObject {PublicECKeyAttributes},
        publicDHKey   [1] PublicKeyObject {PublicDHKeyAttributes},
        publicDSAKey  [2] PublicKeyObject {PublicDSAKeyAttributes},
        publicKEAKey  [3] PublicKeyObject {PublicKEAKeyAttributes},
        ... -- For future extensions
        }

PublicKeyObject {KeyAttributes} ::= PKCS15Object {
        CommonKeyAttributes, CommonPublicKeyAttributes, KeyAttributes}

PublicRSAKeyAttributes ::= SEQUENCE {
        value              ObjectValue {RSAPublicKeyChoice},
        modulusLength      INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo            KeyInfo {NULL, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

RSAPublicKeyChoice ::= CHOICE {
        raw       RSAPublicKey,
        spki      [1] SubjectPublicKeyInfo, -- See X.509. Must contain a public RSA key
        ...
        }

RSAPublicKey ::= SEQUENCE {
        modulus            INTEGER, -- n
        publicExponent     INTEGER -- e
        }

PublicECKeyAttributes ::= SEQUENCE {
        value     ObjectValue {ECPublicKeyChoice},
        keyInfo   KeyInfo {Parameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

ECPublicKeyChoice ::= CHOICE {
        raw       ECPoint,
        spkid SubjectPublicKeyInfo, -- See X.509. Must contain a public EC key
        ...
        }
```

```
PublicDHKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DHPublicKeyChoice},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DHPublicKeyChoice ::= CHOICE {
        raw         DiffieHellmanPublicNumber,
        spki SubjectPublicKeyInfo, -- See X.509. Must contain a public D-H key
        ...
        }

PublicDSAKeyAttributes ::= SEQUENCE {
        value      ObjectValue {DSAPublicKeyChoice},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

DSAPublicKeyChoice ::= CHOICE {
        raw         INTEGER,
        spki SubjectPublicKeyInfo, -- See X.509. Must contain a public DSA key.
        ...
        }

PublicKEAKeyAttributes ::= SEQUENCE {
        value      ObjectValue {KEAPublicKeyChoice},
        keyInfo    KeyInfo {DomainParameters, PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

KEAPublicKeyChoice ::= CHOICE {
        raw         INTEGER,
        spki SubjectPublicKeyInfo, -- See X.509. Must contain a public KEA key
        ...
        }

SecretKeyType ::= CHOICE {
        genericSecretKey SecretKeyObject {GenericSecretKeyAttributes},
        rc2key          [0] SecretKeyObject {GenericSecretKeyAttributes},
        rc4key          [1] SecretKeyObject {GenericSecretKeyAttributes},
        desKey          [2] SecretKeyObject {GenericSecretKeyAttributes},
        des2Key         [3] SecretKeyObject {GenericSecretKeyAttributes},
        des3Key         [4] SecretKeyObject {GenericSecretKeyAttributes},
        castKey         [5] SecretKeyObject {GenericSecretKeyAttributes},
        cast3Key        [6] SecretKeyObject {GenericSecretKeyAttributes},
        cast128Key      [7] SecretKeyObject {GenericSecretKeyAttributes},
        rc5Key          [8] SecretKeyObject {GenericSecretKeyAttributes},
        ideaKey         [9] SecretKeyObject {GenericSecretKeyAttributes},
        skipjackKey     [10] SecretKeyObject {GenericSecretKeyAttributes},
        batonKey        [11] SecretKeyObject {GenericSecretKeyAttributes},
        juniperKey      [12] SecretKeyObject {GenericSecretKeyAttributes},
        rc6Key          [13] SecretKeyObject {GenericSecretKeyAttributes},
        otherKey        [14] OtherKey,
        ... -- For future extensions
        }

SecretKeyObject {KeyAttributes} ::= PKCS15Object {
         CommonKeyAttributes, CommonSecretKeyAttributes, KeyAttributes}

OtherKey ::= SEQUENCE {
        keyType  OBJECT IDENTIFIER,
```

```
        keyAttr   SecretKeyObject {GenericSecretKeyAttributes}
        }

GenericSecretKeyAttributes ::= SEQUENCE {
        value     ObjectValue { OCTET STRING },
        ... -- For future extensions
        }

CertificateType ::= CHOICE {
        x509Certificate            CertificateObject { X509CertificateAttributes},
        x509AttributeCertificate   [0] CertificateObject {X509AttributeCertificateAttributes},
        spkiCertificate            [1] CertificateObject {SPKICertificateAttributes},
        pgpCertificate             [2] CertificateObject {PGPCertificateAttributes},
        wtlsCertificate            [3] CertificateObject {WTLSCertificateAttributes},
        x9-68Certificate           [4] CertificateObject {X9-68CertificateAttributes},
        ...,
        cvCertificate              [5] CertificateObject {CVCertificateAttributes}
        }

CertificateObject {CertAttributes} ::= PKCS15Object {
        CommonCertificateAttributes, NULL, CertAttributes}

X509CertificateAttributes ::= SEQUENCE {
        value          ObjectValue { Certificate },
        subject        Name OPTIONAL,
        issuer         [0] Name OPTIONAL,
        serialNumber   CertificateSerialNumber OPTIONAL,
        ... -- For future extensions
        }

X509AttributeCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { AttributeCertificate },
        issuer    GeneralNames OPTIONAL,
        serialNumber CertificateSerialNumber OPTIONAL,
        attrTypes      [0] SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        }

SPKICertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

PGPCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

WTLSCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

X9-68CertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

CVCertificateAttributes ::= SEQUENCE {
        value     ObjectValue { PKCS15-OPAQUE.&Type},
        ... -- For future extensions
        }
```

```
DataType ::= CHOICE {
        opaqueDO        DataObject {Opaque},
        externalIDO     [0] DataObject {ExternalIDO},
        oidDO           [1] DataObject {OidDO},
        ... -- For future extensions
        }

DataObject {DataObjectAttributes} ::= PKCS15Object {
        CommonDataObjectAttributes, NULL, DataObjectAttributes}

Opaque ::= ObjectValue {PKCS15-OPAQUE.&Type}

ExternalIDO ::= ObjectValue {PKCS15-OPAQUE.&Type}
        (CONSTRAINED BY {-- All data objects must be defined in
        -- accordance with ISO/IEC 7816-6 --})

OidDO ::= SEQUENCE {
        id          OBJECT IDENTIFIER,
        value       ObjectValue {PKCS15-OPAQUE.&Type}
        }

AuthenticationType ::= CHOICE {
        pin                     AuthenticationObject { PinAttributes },
        ...,
        biometricTemplate       [0] AuthenticationObject {BiometricAttributes},
        authKey                 [1] AuthenticationObject {AuthKeyAttributes},
        external                [2] AuthenticationObject {ExternalAuthObjectAttributes}
        }

AuthenticationObject {AuthObjectAttributes} ::= PKCS15Object {
        CommonAuthenticationObjectAttributes, NULL, AuthObjectAttributes}

PinAttributes ::= SEQUENCE {
        pinFlags        PinFlags,
        pinType         PinType,
        minLength       INTEGER (pkcs15-lb-minPinLength..pkcs15-ub-minPinLength),
        storedLength    INTEGER (0..pkcs15-ub-storedPinLength),
        maxLength       INTEGER OPTIONAL,
        pinReference    [0] Reference DEFAULT 0,
        padChar         OCTET STRING (SIZE(1)) OPTIONAL,
        lastPinChange GeneralizedTime OPTIONAL,
        path            Path OPTIONAL,
        ... -- For future extensions
        }

PinFlags ::= BIT STRING {
        case-sensitive                  (0),
        local                           (1),
        change-disabled                 (2),
        unblock-disabled                (3),
        initialized                     (4),
        needs-padding                   (5),
        unblockingPin                   (6),
        soPin                           (7),
        disable-allowed                 (8),
        integrity-protected             (9),
        confidentiality-protected       (10),
        exchangeRefData                 (11)
        } (CONSTRAINED BY { -- 'unblockingPin' and 'soPIN' cannot both be set -- })

PinType ::= ENUMERATED {bcd, ascii-numeric, utf8, ..., half-nibble-bcd, iso9564-1}
```

```
BiometricAttributes ::= SEQUENCE {
        bioFlags        BiometricFlags,
        templateId      OBJECT IDENTIFIER,
        bioType         BiometricType,
        bioReference    Reference DEFAULT 0,
        lastChange      GeneralizedTime OPTIONAL,
        path            Path OPTIONAL,
        ... -- For future extensions
        }

BiometricFlags ::= BIT STRING {
        local                   (1),
        change-disabled         (2),
        unblock-disabled        (3),
        initialized             (4),
        disable-allowed         (8),
        integrity-protected     (9),
        confidentiality-protected   (10)
        } -- Note: bits 0, 5, 6, and 7 are reserved for future use

BiometricType ::= CHOICE {
        fingerPrint     FingerPrint,
        irisScan        [0] IrisScan,
        -- Possible extensions:
        -- voiceScan    VoiceScan,
        -- faceScan     FaceScan,
        -- retinaScan   Retinascan,
        -- handGeometry         HandGeometry,
        -- writeDynamics        WriteDynamics,
        -- keyStrokeDynamicsKeyStrokeDynamics,
        -- lipDynamics  LipDynamics,
        ... -- For future extensions
        }

FingerPrint ::= SEQUENCE {
        hand    ENUMERATED {left, right},
        finger  ENUMERATED {thumb, pointerFinger, middleFinger, ringFinger, littleFinger},
        ...
        }

IrisScan ::= SEQUENCE {
        eye     ENUMERATED {left, right},
        ...
        }

ExternalAuthObjectAttributes ::= CHOICE {
        authKeyAttributes       AuthKeyAttributes,
        certBasedAttributes  [0] CertBasedAuthenticationAttributes,
        ... -- For future extensions
        }

AuthKeyAttributes ::= SEQUENCE {
        derivedKey      BOOLEAN DEFAULT TRUE,
        authKeyId       Identifier,
        ... -- For future extensions
        }

CertBasedAuthenticationAttributes ::= SEQUENCE {
        cha     OCTET STRING,
        ...
        }
```

```
TokenInfo ::= SEQUENCE {
        version             INTEGER {v1(0)} (v1,...),
        serialNumber        OCTET STRING,
        manufacturerID      Label OPTIONAL,
        label               [0] Label OPTIONAL,
        tokenflags          TokenFlags,
        seInfo              SEQUENCE OF SecurityEnvironmentInfo OPTIONAL,
        recordInfo          [1] RecordInfo OPTIONAL,
        supportedAlgorithms[2] SEQUENCE OF AlgorithmInfo OPTIONAL,
        ...,
        issuerId            [3] Label OPTIONAL,
        holderId            [4] Label OPTIONAL,
        lastUpdate          GeneralizedTime OPTIONAL,
        preferredLanguage   PrintableString OPTIONAL -- In accordance with IETF RFC 1766
        } (CONSTRAINED BY { -- Each AlgorithmInfo.reference value must be unique --})

TokenFlags ::= BIT STRING {
        readonly        (0),
        loginRequired   (1),
        prnGeneration   (2),
        eidCompliant    (3)
        }

SecurityEnvironmentInfo ::= SEQUENCE {
        se      INTEGER (0..pkcs15-ub-seInfo),
        owner   OBJECT IDENTIFIER,
        ... -- For future extensions
        }

RecordInfo ::= SEQUENCE {
        oDFRecordLength     [0] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        prKDFRecordLength   [1] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        puKDFRecordLength   [2] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        sKDFRecordLength    [3] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        cDFRecordLength     [4] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        dODFRecordLength    [5] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL,
        aODFRecordLength    [6] INTEGER (0..pkcs15-ub-recordLength) OPTIONAL
        }

AlgorithmInfo ::= SEQUENCE {
        reference           Reference,
        algorithm           PKCS15-ALGORITHM.&id({AlgorithmSet}),
        parameters          PKCS15-ALGORITHM.&Parameters({AlgorithmSet}{@algorithm}),
        supportedOperationsPKCS15-ALGORITHM.&Operations({AlgorithmSet}{@algorithm}),
        algId               PKCS15-ALGORITHM.&objectIdentifier({AlgorithmSet}{@algorithm})
                            OPTIONAL,
        algRef              Reference OPTIONAL
        }

PKCS15-ALGORITHM ::= CLASS {
         &id INTEGER UNIQUE,
         &Parameters,
         &Operations Operations,
        &objectIdentifier OBJECT IDENTIFIER OPTIONAL
} WITH SYNTAX {
          PARAMETERS &Parameters OPERATIONS &Operations ID &id [OID &objectIdentifier]}

PKCS15-OPAQUE ::= TYPE-IDENTIFIER

PublicKeyOperations ::= Operations
```

```
Operations ::= BIT STRING {
          compute-checksum  (0), -- H/W computation of checksum
          compute-signature (1), -- H/W computation of signature
          verify-checksum   (2), -- H/W verification of checksum
          verify-signature  (3), -- H/W verification of signature
          encipher          (4), -- H/W encryption of data
          decipher          (5), -- H/W decryption of data
          hash              (6), -- H/W hashing
          generate-key      (7)  -- H/W key generation
          }

pkcs15-alg-null  PKCS15-ALGORITHM ::= {
           PARAMETERS NULL OPERATIONS {{generate-key}} ID -1}

AlgorithmSet PKCS15-ALGORITHM ::= {
           pkcs15-alg-null,
           ... -- For future (or local) extensions
           }

-- Soft token related types and objects

EnvelopedData {Type} ::= SEQUENCE {
          version                  INTEGER{v0(0),v1(1),v2(2),v3(3),v4(4)}(v0|v1|v2,...),
          originatorInfo           [0] OriginatorInfo OPTIONAL,
          recipientInfos           RecipientInfos,
          encryptedContentInfo     EncryptedContentInfo{Type},
          unprotectedAttrs         [1] SET SIZE (1..MAX) OF Attribute
          }

EncryptedContentInfo {Type} ::= SEQUENCE {
          contentType                   OBJECT IDENTIFIER,
          contentEncryptionAlgorithm    AlgorithmIdentifier{{ContentEncryptionAlgorithms}},
          encryptedContent              [0] OCTET STRING
          }(CONSTRAINED BY {-- 'encryptedContent' shall be the result of encrypting DER-encoded
          -- value of type -- Type})

PKCS15Token ::= SEQUENCE {
          version             INTEGER {v1(0)} (v1,...),
          preferredLanguage   PrintableString OPTIONAL, -- In accordance with IETF RFC 1766
          keyManagementInfo   [0] KeyManagementInfo OPTIONAL,
          pkcs15Objects       SEQUENCE OF PKCS15Objects
          }

KeyManagementInfo ::= SEQUENCE OF SEQUENCE {
          keyId          Identifier,
          keyInfo        CHOICE {
              recipientInfo     RecipientInfo,
              passwordInfo      [0] PasswordInfo
              }
          } (CONSTRAINED BY {-- Each keyID must be unique --})

PasswordInfo ::= SEQUENCE {
          hint      Label OPTIONAL,
          algId     AlgorithmIdentifier {{KeyDerivationAlgorithms}},
          ...
          } (CONSTRAINED BY {--keyID shall point to a KEKRecipientInfo--})

KeyDerivationAlgorithms ALGORITHM-IDENTIFIER ::= {
          PBKDF2Algorithms,
          ... -- For future extensions
          }
```

```
KeyEncryptionAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY id-alg-CMS3DESwrap} |
        {INTEGER IDENTIFIED BY id-alg-CMSRC2wrap},
        ... -- For future extensions
        }

ContentEncryptionAlgorithms ALGORITHM-IDENTIFIER ::= {
        SupportingAlgorithms EXCEPT {NULL IDENTIFIED BY id-hmacWithSHA1},
        -- SupportingAlgorithms are defined in PKCS #5 v2
        ... -- For future extensions
        }

MACAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY hMAC-SHA1},
        ... -- For future extensions
        }

DigestAlgorithms ALGORITHM-IDENTIFIER ::= {
        {NULL IDENTIFIED BY sha-1},
        ... -- For future extensions
        }

-- Misc

DDO ::= SEQUENCE {
        oid             OBJECT IDENTIFIER,
        odfPath         Path OPTIONAL,
        tokenInfoPath   [0] Path OPTIONAL,
        unusedPath      [1] Path OPTIONAL,
        ... -- For future extensions
        }

DIRRecord ::=   [APPLICATION 1] SEQUENCE {
        aid     [APPLICATION 15] OCTET STRING,
        label   [APPLICATION 16] UTF8String OPTIONAL,
        path    [APPLICATION 17] OCTET STRING,
        ddo     [APPLICATION 19] DDO OPTIONAL
        }

UnusedSpace ::= SEQUENCE {
        path    Path (WITH COMPONENTS {..., index PRESENT, length PRESENT}),
        authId  Identifier OPTIONAL,
        ...,
        accessControlRules SEQUENCE OF AccessControlRule OPTIONAL
        }

END
```

B. File access conditions

B.1 Scope

This appendix is only applicable to IC card implementations.

B.2 Background

Since this document is intended to be independent of particular IC card brands and models, we define "generic" IC card access methods which should be straightforward to map to actual IC card operating system-native commands (assuming the card is an ISO/IEC 7816-4 compliant IC card).

B.3 Read-Only and Read-Write cards

Access conditions for files in the PKCS15 application can be set up differently depending on if the application is to be read-only or read-write. A read-only card might be desired for high-security purposes, for example when it has been issued using a secure issuing process, and it is to be certain that it can not be manipulated afterwards.

The following is a table of different possible access methods, which is a superset of the **Operations** type. These are generic methods which should be possible to map to all different IC card types (sometimes the mapping might turn out to be a "No-Op", because the card does not support any similar operation). The exact access methods, and their meaning, varies for each IC card type. In the table, a '*' indicates that the access method is only relevant for files containing keys. These methods are abbreviated to '`CRYPT`' in Table 5.

Table 3 – File access methods

File type	Access method	Meaning
DF	`Create`	Allows new files, both EFs and DFs to be created in the DF.
	`Delete`	Allows files in the DF to be deleted. Relevant only for cards which support deletion.
EF	`Read`	It is allowed to read the file's contents.
	`Update`	It is allowed to update the file's contents.
	`Append`	It is allowed to append information to the file (usually only applicable to linear record files).
	`Compute checksum`	* The contents of the file can be used when computing a checksum
	`Compute Signature`	* The contents of the file can be used when computing a signature
	`Verify checksum`	* The contents of the file can be used when verifying a checksum
	`Verify signature`	* The contents of the file can be used when verifying a signature
	`Encipher`	* The contents of the file can be used in an enciphering operation
	`Decipher`	* The contents of the file can be used in a deciphering operation

NOTE – It is the directory's access methods, and not the files', which decide if files in the directory are allowed to be created or deleted.

Each access method can have the following conditions. These are also generic and should be possible to implement on all card types.

Table 4 – Possible access conditions

Type	Meaning
NEV	The operation is never allowed, not even after cardholder verification.
ALW	The operation is always allowed, without cardholder verification.
CHV	The operation is allowed after a successful card holder verification.
SYS	The operation is allowed after a system key presentation, typically available only to the card issuer (The Security Officer case), e.g. 'EXTERNAL AUTHENTICATE'

The following access conditions are recommended for files related to the PKCS #15 application:

Table 5 -Recommended file access conditions

File	DF	R/O card	R/W card
MF	X	Create: SYS Delete: NEV	Create: SYS Delete: SYS
DIR		Read: ALW Update: SYS Append: SYS	Read: ALW Update: SYS Append: SYS
PIN files		Read: NEV Update: NEV Append: NEV	Read: NEV Update: CHV Append: NEV
PKCS15	X	Create: SYS Delete: NEV	Create: CHV | SYS Delete: CHV | SYS
TokenInfo		Read: ALW Update: NEV Append: NEV	Read: ALW Update: CHV | SYS | NEV Append: NEV
ODF		Read: ALW Update: NEV Append: SYS | NEV	Read: ALW Update: SYS | NEV Append: SYS | NEV
AODFs		Read: ALW Update: NEV Append: NEV	Read: ALW Update: CHV | SYS | NEV Append: CHV | SYS | NEV
PrKDFs, PuKDFs, SKDFs, CDFs and DODFs		Read: ALW | CHV Update: NEV Append: SYS | NEV	Read: ALW | CHV Update: CHV Append: CHV
Trusted CDFs Trusted PuKDFs		Read: ALW | CHV Update: NEV Append: SYS | NEV	Read: ALW | CHV Update: SYS | NEV Append: SYS | NEV

Key files		Read: NEV Update: NEV Append: NEV Crypt: CHV	Read: NEV Update: CHV | SYS | NEV Append: CHV | SYS | NEV Crypt: CHV
Other EFs		Read: ALW | CHV Update: NEV Append: SYS | NEV	Read: ALW | CHV Update: CHV Append: CHV
NOTE 1 – "Key files" means "Files containing private or secret keys and the card supports crypto-related commands for these files" NOTE 2 – A "|" in the table stands for "or", i.e. the card issuer may choose any Boolean expression of available options. E.g. UPDATE of an EF(ODF) on a R/W card may be permitted only after correct cardholder verification ('CHV') AND an external authentication ('SYS').			

The difference between a read-only and a read-write (R-W) card is basically as follows. For an R-W card, new files can be created (to allow addition of new objects) and some EFs (e. g. CDFs only containing references to public objects) can be updated (to allow adding info about new objects) after correct cardholder verification. It is also possible to replace files on an R-W card.

It is recommended that all cards be personalized with the read-write access control settings, unless they are issued for an environment with high security requirements.

C. An electronic identification profile of PKCS #15

C.1 Scope

This section describes a profile of PKCS #15 suitable for electronic identification (EID) purposes and requirements for it. Implementations may claim compliance with this profile. The profile includes requirements both for cards and for host-side applications making use of EID cards.

C.2 PKCS #15 objects

– Private Keys: A PKCS #15 card issued for EID purposes should contain at least two private keys, of which one should be used for digital signature purposes only. At least one of the other keys should have the value **decrypt** set in its key usage flags. Authentication objects or encryption must protect all private keys. On cards supporting on-chip digital signature operations, it is recommended that the signature-only key be protected from modifications. It must be protected from read-access. Usage of the signature-only key should furthermore require cardholder verification with an authentication object used only for this key. The key length must be sufficient for intended purposes (e.g. 1024 bits or more in the RSA case and 160 bits or more in the EC case, assuming all other parameters has been chosen in a secure manner).

  Allowed private key types for this profile are:

  – RSA keys

  - Elliptic Curve keys (This profile places no restrictions on the domain parameters other than the ones mentioned above)
  - DSA keys

  Host-side applications claiming full conformance to this profile must recognize all these key types and be able to use them. Cards must contain keys of at least one of these types.

- Secret Keys: No requirements. Objects of this type may or may not be present on the card, depending on the application issuer's discretion. There is no requirement for host-side applications to handle these keys.
- Public Keys: No requirements. Objects of this type may or may not be present on the card, depending on the application issuer's discretion. There is no requirement for host-side applications to handle these keys.
- Certificates: For each private key at least one corresponding certificate should be stored in the card. The certificates must be of type **X509Certificate**. If an application issuer stores CA certificates on a card which supports the ISO/IEC 7816-4 logical file organization, and which has suitable file access mechanisms, then it is recommended that they are stored in a protected file. This file shall be pointed to by a CDF file which is only modifiable by the card issuer (or not modifiable at all). This implies usage of the **trustedCertificates** choice in the **Objects** type. User certificates for which private keys exist on the card should be profiled in accordance with IETF RFC 2459. Host-side applications are required to recognize and be able to use the **X509Certificate** type.
- Data objects: No requirements. Objects of this type may or may not be present on the card, depending on the application issuer's discretion.
- Authentication objects: As follows from the description above, in the case of an IC card capable of protecting files with authentication objects, at least one authentication object must be present on the card, protecting private objects. As stated above, a separate authentication object should be used for the signature-only key, if such a key exist. Any use of the signature-only private key shall require a new user authentication, if technically possible. In the case of PIN codes, any positive verification of one PIN code shall not enable the use of security services associated with another PIN code. Consecutive and incorrect verifications of a certain user PIN code shall block all security services associated with that PIN code. It is left to the application issuers to decide the number of consecutive incorrect verifications that triggers a blocking of the card.

  NOTE – Future versions of this profile may also include support for biometric authentication methods

  PINs must be at least 4 characters (BCD, UTF8 or ASCII) long.

  When a PIN is blocked through after consecutive incorrect PIN verifications, the PIN may only be unblocked through a special unblocking procedure, defined by the application issuer.

C.3 Other files

Use of an EF(UnusedSpace) is recommended if the cardholder is allowed to update the contents of the PKCS #15 application.

C.4 Constraints on ASN.1 types

Unless otherwise mentioned, conforming applications are required to recognize and parse all **OPTIONAL** fields. The following constraints applies for cards and applications claiming conformance to this EID profile:

NOTE – "recognize" means "being able to proceed also when the field is present, but not necessarily being able to interpret the field's contents."

- **CommonObjectAttributes.label** must be present for all certificate objects.
- **CommonKeyAttributes.startDate** must not be present.
- **CommonKeyAttributes.endDate** must not be present.
- **CommonPrivateKeyAttributes.subjectName** must not be present.
- **CommonPrivateKeyAttributes.keyIdentifiers** must be recognized by host-side applications but need not be interpreted.
- **CommonCertificateAttributes.requestID** must be recognized by host-side applications but need not be interpreted.
- **X509CertificateAttributes.subject** must not be present.
- **X509CertificateAttributes.issuer** must not be present.
- **X509CertificateAttributes.serialNumber** must not be present.
- **PinAttributes.lastPinChange** must be recognized by host-side applications but need not be interpreted.

C.5 File relationships in the IC card case

The purpose of the following figure is to show the relationship between certain files (EF(ODF), EF(PrKDF), EF(AODF) and EF(CDF)) in the DF(PKCS15) directory.

NOTE – It is possible for **Path** pointers in EF(ODF) to point to locations inside the EF(ODF) itself. For example, if a card issuer intends to "lock" EF(ODF), EF(PrKDF) and EF(AODF), they can all be stored within the same (physical) EF, EF(ODF). The advantage of this is that fewer '`SELECT`' and '`READ`' operations need to be done in order to read the contents of these files. There should be no need for host side applications to be modified due to this fact, however, since ordinary path pointers should be used anyway.

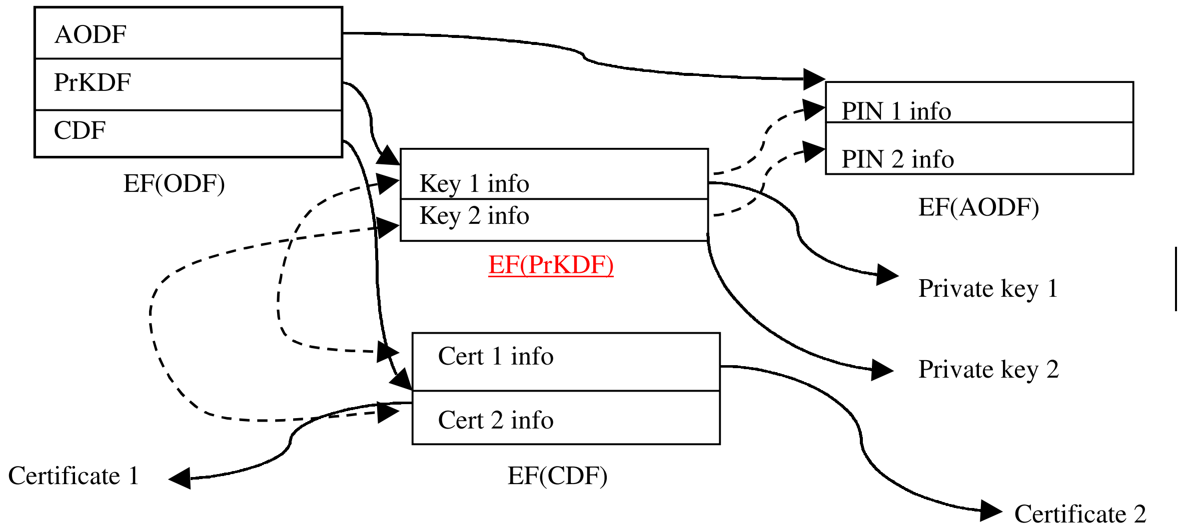


Figure 16 – IC card file relationships in DF(PKCS15). Dashed arrows indicate cross-references.

C.6 Access control rules

Private keys must be private objects, and should be marked as **sensitive**. Files, which contain private keys, should be protected against removal and/or overwriting. Using the definitions in Appendix A, the following access conditions shall be set for files in the PKCS #15 application directory (as in Appendix A, a "|" in the table stands for "or", i.e. a card issuer is free to make any choice, including Boolean expressions of available options):

Table 6 – File access conditions for the EID profile of PKCS #15

File	Access Conditions, R-O card	Access Conditions. R-W card
MF	`Create: SYS` `Delete: NEV`	`Create: SYS` `Delete: SYS`

EF(DIR)	`Read: ALW`
	`Update: SYS`
	`Append: SYS`
		`Read: ALW`
		`Update: SYS`
		`Append: SYS`

PIN files	`Read: NEV`
	`Update: NEV`
	`Append: NEV`
		`Read: NEV`
		`Update: CHV`
		`Append: NEV`

DF(PKCS15)	`Create: SYS` `Delete: NEV`	`Create: CHV │ SYS` `Delete: SYS`

EF(TokenInfo)	`Read: ALW`
	`Update: NEV`
	`Append: NEV`
		`Read: ALW`
		`Update: CHV │ SYS │ NEV`
		`Append: NEV`

EF(ODF)	`Read: ALW`
	`Update: NEV`
	`Append: NEV`
		`Read: ALW`
		`Update: SYS`
		`Append: SYS`

AODFs	`Read: ALW`
	`Update: NEV`
	`Append: NEV`
		`Read: ALW`
		`Update: NEV`
		`Append: CHV | SYS`

PrKDFs, PuKDFs, SKDFs, CDFs and DODFs	`Read: ALW | CHV`
	`Update: NEV`
	`Append: SYS | NEV`
		`Read: ALW | CHV`
		`Update: CHV | SYS | NEV`
		`Append: CHV | SYS`

Trusted CDFs	`Read: ALW | CHV`
	`Update: NEV`
	`Append: SYS | NEV`
		`Read: ALW | CHV`
		`Update: SYS | NEV`
		`Append: SYS | NEV`

Key files (see footnote for Table 5)	`Read: NEV`
	`Update: NEV`
	`Append: NEV`
	`Crypt: CHV`
		`Read: NEV`
		`Update: CHV | SYS | NEV`
		`Append: CHV | SYS | NEV`
		`Crypt: CHV`

Other EFs in the PKCS15 directory	`Read: ALW | CHV`
	`Update: NEV`
	`Append: SYS | NEV`
	`Crypt: CHV` (when applic.)
		`Read: ALW | CHV`
		`Update: CHV | SYS | NEV`
		`Append: CHV | SYS | NEV`
		`Crypt: CHV` (when applic.)

NOTE – If an application issuer wants to protect an object directory file with an authentication object, then by default the first authentication object in EF(AODF) shall be used. Obviously, EF(ODF) and EF(AODF) cannot be protected in this manner.

D. Example PKCS #15 topologies

The following figures show possible implementations of PKCS #15 on multi-application tokens.

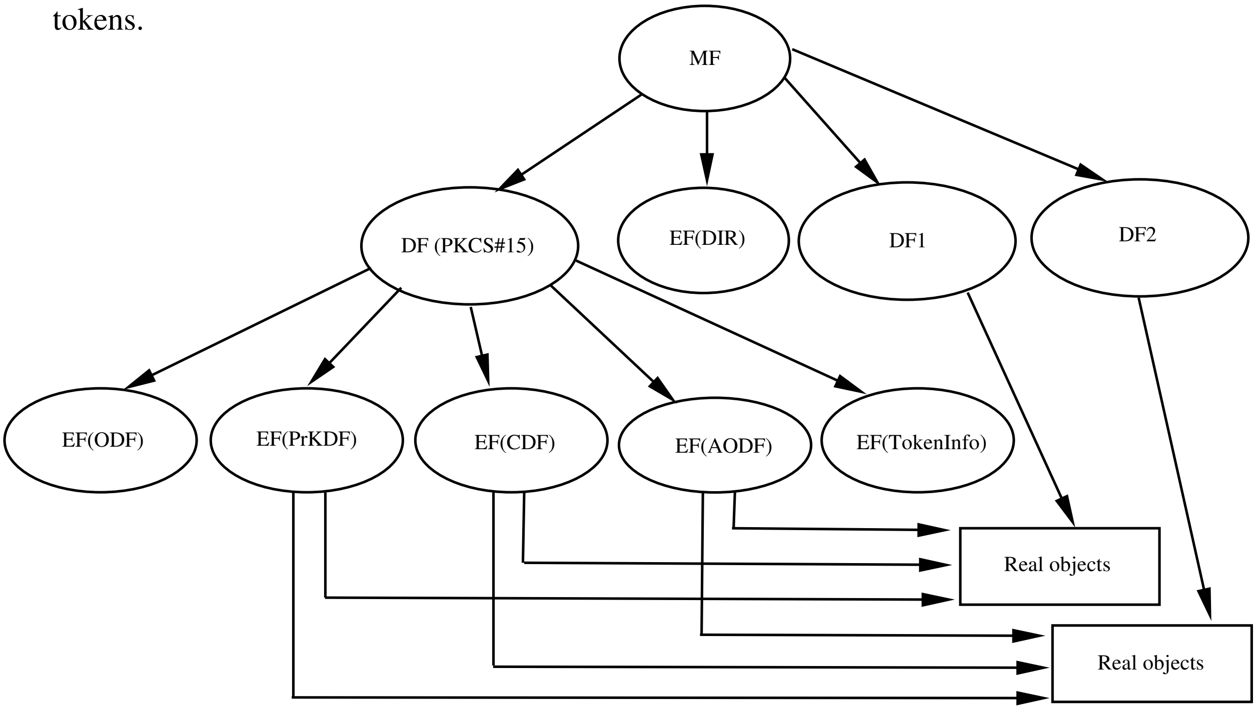


Figure 17 – Example with three applications. "Real" objects (i.e. keys, certificates) are stored outside the PKCS #15 application

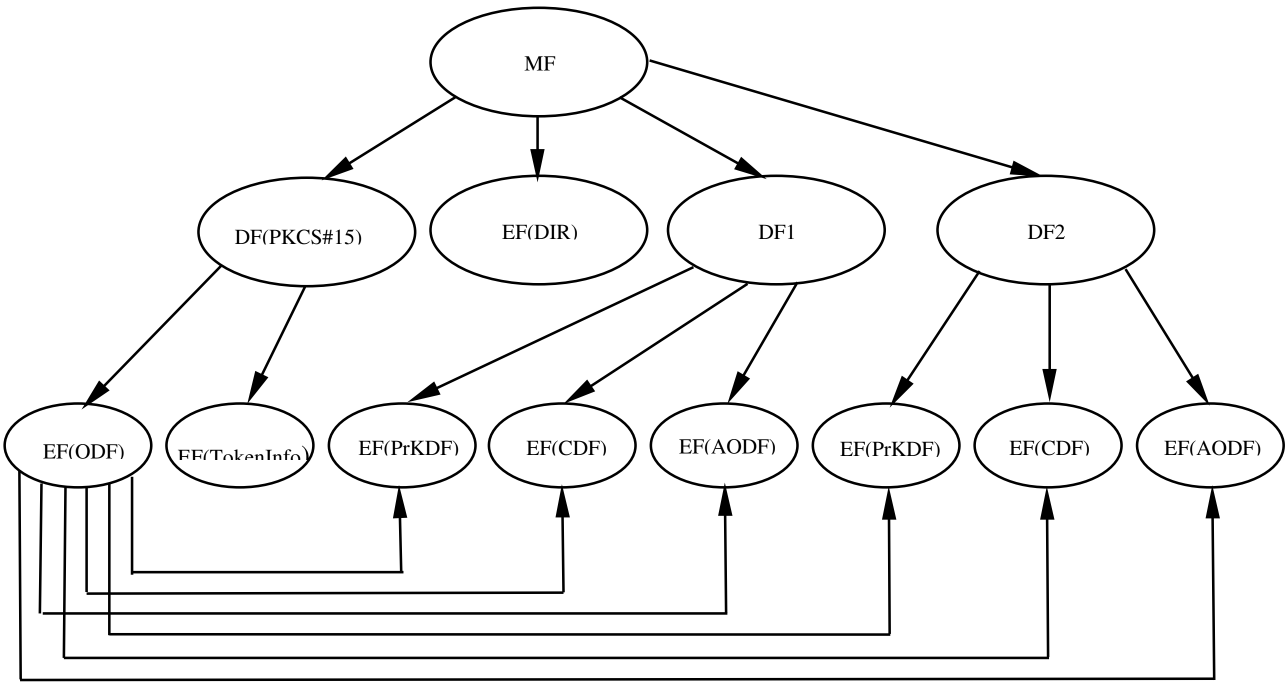


Figure 18 – Example with three applications. Only EF(ODF) and EF(TokenInfo) in DF(PKCS #15)

E. Using PKCS #15 software tokens

E.1 Constructing a PKCS#15 token in software

E.1.1 Scope

This section describes how to generate a password-protected value of type **PKCS15Token**, for use, e.g. as a "virtual smart card." It also applies when PKCS#15 tokens are stored on other untrusted media, such as stored-value cards without PIN protection.

E.1.2 Constructing password-protected values of type 'Enveloped Data'

EnvelopedData{} structures is created for password-protected private values as follows (refer to RFC 2630 for identifier and type definitions):

- **EnvelopedData.version** must have the value '**2**'
- **EnvelopedData.recipientInfos** should contain at least one **RecipientInfo** of type **KEKRecipientInfo**. with the following restrictions:
  - **KEKRecipientInfo.version** must be '**4**'
  - **KEKRecipientInfo.kekid.keyIdentifier** shall be set to the corresponding **keyId** in **PKCS15Token.keyManagementInfo** (see below)
  - **KEKRecipientInfo.kekid.date** need not be present
  - **KEKRecipientInfo.kekid.other** need not be present
  - **KEKRecipientInfo.keyEncryptionAlgorithm** must be chosen from the set of **KeyEncryptionAlgorithms** defined above.
  - **KEKRecipientInfo.encryptedKey** shall contain the encrypted content-encryption key
- **EnvelopedData.encryptedContentInfo.contentType** shall have the value **id-data** (see, e.g. PKCS #7).
- **EnvelopedData.encryptedContentInfo.contentEncryptionAlgorithm** must be chosen from the set of **ContentEncryptionAlgorithms** defined in Section 7.4
- **EnvelopedData.encryptedContentInfo.encryptedContent** must contain the result of encrypting a DER-encoding of the value in question (a **ObjectValue** or a **SEQUENCE OF PKCS15Objects**) with the content-encryption key encrypted in **KEKRecipientInfo.encryptedKey** and the algorithm indicated in **EnvelopedData.encryptedContentInfo.contentEncryptionAlgorithm**.
- The content-encryption key shall be encrypted with a key-encryption key derived from the user's authentication password (see next section) by an algorithm from the **KeyDerivationAlgorithms** set.

E.1.3 Integrity-protection

After all private values have been wrapped in **EnvelopedData** structures, all **EnvelopedData{}** structures all sequences of public objects is collected in a value of type **PKCS15Token**[2]. The **keyManagementInfo** field shall contain a **SEQUENCE OF KeyManagementInfo** values, each one with a unique **keyID**, corresponding to a **keyIdentifier** in some **EnvelopedData** structure. For **PasswordInfo**s, the **KeyDerivationAlgorithm** is indicated together with an optional hint about the password. Finally a value of type **AuthenticatedData** is optionally constructed as follows:

- **AuthenticatedData.version** shall have the value '**0**'
- If the **KEKRecipientInfo** alternative of **RecipientInfos** is chosen, the following restrictions apply:
  - **KEKRecipientInfo.version** must be '**4**'
  - **KEKRecipientInfo.kekid.keyIdentifier** should be set to a **keyID** present in the enclosed **keyManagementInfo**, but may also be set to something else. If a **keyID** already present in the enclosed **keyManagementInfo** is used, then the key-encryption shall be derived using information from the **keyManagementInfo**.
  - **KEKRecipientInfo.keyEncryptionAlgorithm** must be chosen from the set of **KeyEncryptionAlgorithms** defined in Section 7.4.
- **AuthenticatedData.macAlgorithm** must be chosen from the set of **MACAlgorithms** defined in Section 7.4.
- **AuthenticatedData.digestAlgorithm** must be present and contain a value from the set of **DigestAlgorithms** defined above
- **AuthenticatedData.encapContentInfo.contentType** shall have the value **{pkcs15-ct 1}**
- **AuthenticatedData.encapContentInfo.content** shall contain a DER-encoding of the **PKCS15Token** structure. The encoding is wrapped inside an **OCTET STRING**, as specified in RFC 2630.
- **AuthenticatedData.authenticatedAttributes** shall be present, and contain both the **id-contentType** attribute and the **id-messageDigest** attribute, as specified in RFC 2630.
- **AuthenticatedData.mac** shall contain the message authentication code.
- **AuthenticatedData.unauthenticatedAttributes** need not be present.

NOTE – For interoperability reasons, it could be wise to restrict users to use just one or two passwords for all **EnvelopedData{}** structures (and any optional, enclosing **AuthenticatedData** structures).

---

[2] Note – The choice of preferred practice in terms of the number of EnvelopedData structures is out of scope for this document. Among factors influencing this choice are: varying access permissions; access time; modularity; etc.

F. Notes to implementors

F.1 Diffie-Hellman parameter mismatch

Implementers should note that the order of parameters **g** and **q** in the **DomainParameters** type defined in [3] have changed in later drafts; it is now consistent with, e.g., [10].

F.2 Explicit tagging of parameterized types

When a parameterized type, such as the **PKCS15Object**, contains a context tag, that tag will be encoded as an **EXPLICIT** tag regardless of the tagging policy of the module in which the parameterized type occurs. See further the section on tagging in [18].

G. Intellectual property considerations

RSA Security makes no patent claims on the general constructions described in this document, although specific underlying techniques may be covered.

License to copy this document is granted provided that it is identified as "RSA Security Inc. Public-Key Cryptography Standards (PKCS)" in all material mentioning or referencing this document.

RSA Security makes no representations regarding intellectual property claims by other parties. Such determination is the responsibility of the user.

H. Revision history

Version 1.0

Version 1.0 was published in April, 1999.

Version 1.1

This is the second draft of PKCS #15 version 1.1, and contains corrections and updates to the first draft. Major differences between this draft and PKCS #15 version 1.0 includes:

- Added support for other authentication methods (biometrics, external, cha)
- Added support for more access control information
- Added support for software tokens (virtual smart cards)
- Added support for card-verifiable certificates

I. References

[1] H. Alvestrand, "*Tags for the Identification of Languages*," IETF RFC 1766, March 1995

[2] ANSI X3.4:1968, *Information Systems – Coded Character Sets – 7-Bit American National Standard Code for Information Interchange*

[3] ANSI X9.42-1999 (DRAFT): *Public Key Cryptography for The Financial Service Industry: Agreement of Symmetric Keys on Using Diffie-Hellman and MQV Algorithms*

[4] ANSI X9.62-1998: *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*

[5] T. Berners-Lee, R. Fielding, L. Masinter, "*Uniform Resource Identifiers (URI): Generic Syntax,*" IETF RFC 2396, August 1998

[6] J. Callas, L. Donnerhacke, H. Finney, R. Thayer, "*OpenPGP Message*," IETF RFC 2440, November 1998

[7] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen, "*SPKI Certificate Theory*," IETF RFC 2693, September 1999

[8] DIN NI-17.4, "*Specification of chipcard interface with digital signature application/function acc. to SigG and SigV*", December 1998

[9] R. Housley, "*Cryptographic Message Syntax*," IETF RFC 2630, June 1999

[10] R. Housley, W. Ford, W. Polk, D. Solo, "*Internet X.509 Public Key Infrastructure – Certificate and CRL Profile*," IETF RFC 2459, January 1999

[11] ISO/IEC 7812-1:1998, Information Technology – Identification Cards – Identification of Issuers – Part1: Numbering System

[12] ISO/IEC 7816-4:1995, *Information Technology – Identification Cards – Integrated Circuit(s) cards with contacts – Part 4: Interindustry commands for interchange*

[13] ISO/IEC 7816-5:1994, *Information Technology – Identification Cards – Integrated Circuit(s) cards with contacts – Part 5: Numbering system and registration procedure for application identifiers*

[14] ISO/IEC 7816-6:1996, *Information Technology – Identification Cards – Integrated Circuit(s) cards with contacts – Part 6: Inter-industry data elements*

[15] ISO/IEC 7816-8:1999, *Information Technology – Identification Cards – Integrated Circuit(s) cards with contacts – Part 8: Security related interindustry commands*

[16] DIS ISO/IEC 7816-9:2000, *Information Technology – Identification Cards – Integrated Circuit(s) cards with contacts – Part 9: Security attributes and additional interindustry command*

[17] ISO/IEC 8583-2:1998, *Financial transaction card originated messages – Interchange message specifications – Part 2: Application and registration procedures for Institution Identification Codes (IIC)*

[18] ISO/IEC 8824-1:1998 | ITU-T Recommendation X.680 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

[19] ISO/IEC 8824-2:1998 | ITU-T Recommendation X.681 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*

[20] ISO/IEC 8824-3:1998 | ITU-T Recommendation X.682 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification*

[21] ISO/IEC 8824-4:1998 | ITU-T Recommendation X.683 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*

[22] ISO/IEC 8825-1:1998 | ITU-T Recommendation X.690 (1997), *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

[23] ISO/IEC 8825-2:1998 | ITU-T Recommendation X.691 (1997), *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*

[24] ISO/IEC 9564-1:1991, *Banking – Personal Identification Number management and security – Part 1: PIN protection principles and techniques*

[25] ISO/IEC 9594-2:1995 | ITU-T Recommendation X.501 (1993), *Information technology – Open Systems Interconnection – The Directory: Models*

[26] ISO/IEC 9594-8:1997 | ITU-T Recommendation X.509 (1997), *Information technology – Open Systems Interconnection – The Directory: Authentication framework*

[27] RSA Laboratories, *PKCS #5 v2.0: Password-based Cryptography Standard*

[28] RSA Laboratories, *PKCS #7 v1.5: Cryptographic Message Syntax Standard*

[29] RSA Laboratories, *PKCS #11 v2.01: Cryptographic Card Interface Standard*

[30] RSA Laboratories, *PKCS #15 v1.0: Cryptographic Token Information Format Standard*

[31] D. Solo, R. Housley, W. Ford, T. Polk, "*Internet X.509 Public Key Infrastructure Certificate and CRL Profile*," IETF RFC 2459, January 1999

[32] WAP Forum, *Wireless Application Protocol – Wireless Transport Layer Security Protocol Specification*, version 8-Nov-1999

[33] F. Yergeau, "*UTF-8, a transformation format of ISO 10646*," IETF RFC 2279, January 1998

J. About PKCS

The *Public-Key Cryptography Standards* are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography. First published in 1991 as a result of meetings with a small group of early adopters of public-key technology, the PKCS documents have become widely referenced and implemented. Contributions from the PKCS series have become part of many formal and *de facto* standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL.

Further development of PKCS occurs through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. For more information, contact:

PKCS Editor
RSA Laboratories
20 Crosby Drive
Bedford, MA 01730 USA
pkcs-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/pkcs