RSA Laboratories. A Division of RSA Data Security

# PKCS #15 v1.0: Cryptographic Token Information Format Standard

*RSA Laboratories*

*April 23, 1999*

## Table of Contents




Copyright © 1991-1999 RSA Laboratories, a division of RSA Data Security, Inc., a Security Dynamics Company. License to copy this document is granted provided that it is identified as "RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS)" in all material mentioning or referencing this document.



003-903076-100-000-000

Copyright © 1999 RSA Laboratories.

# 1 Introduction

Many cryptographic tokens such as Integrated Circuit Cards (IC cards or 'smart 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.
- 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

Copyright © 1999 RSA Laboratories.

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.
- Portable permissions for use in off-line situations.

Unfortunately, the use of these tokens for authentication and authorization purposes is 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 tokens - it is already apparent in the use of certificates with World Wide Web browsers, for example - the limited room on many tokens 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).

Copyright © 1999 RSA Laboratories.

- Enable the use of advances in technology without rewriting application-level software (application neutral).
- Maintain consistency with existing, related standards while expanding upon them only where necessary and practical.

As a practical example, the holder of a token containing a digital certificate should be able to present the token to any application running on any host and successfully use the token 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. The format builds on the PKCS #11 specification.

## 2 References and related documents

- ISO/IEC 7816-4:1995 Identification Cards - Integrated Circuit(s) cards with contacts - Part 4: Interindustry commands for interchange.
- ISO/IEC 7816-5:1994 Identification Cards - Integrated Circuit(s) cards with contacts - Part 5: Numbering system and registration procedure for application identifiers.
- ISO/IEC 7816-6:1996 Identification Cards - Integrated Circuit(s) cards with contacts - Part 6: Inter-industry data elements.
- ISO/IEC 7816-8:1999 Identification Cards – Integrated Circuit(s) cards with contacts – Part 8: Security related interindustry commands.
- FCD ISO/IEC 7816-9:1999 Identification Cards – Integrated Circuit(s) cards with contacts – Part 9: Security attributes and additional interindustry commands.
- ISO/IEC 8824-1:1995 Information technology – Abstract Syntax Notation One (ASN.1) - Specification of basic notation.
- ISO/IEC 8824-1:1995/Amd.1:1995 Information technology – Abstract Syntax Notation One (ASN.1) – Specification of basic notation – Amendment 1 – Rules of extensibility.
- ISO/IEC 8824-2:1995 Information technology – Abstract Syntax Notation One (ASN.1) - Information object specification.
- ISO/IEC 8824-2:1995/Amd.1:1995 Information technology – Abstract Syntax Notation One (ASN.1) – Information object specification – Amendment 1 – Rules of extensibility.

Copyright © 1999 RSA Laboratories.

- ISO/IEC 8824-3:1995 Information technology – Abstract Syntax Notation One (ASN.1) - Constraint specification.
- ISO/IEC 8824-4:1995 Information technology – Abstract Syntax Notation One (ASN.1) - Parameterization of ASN.1 specifications.
- ISO/IEC 8825-1:1995 Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).
- ISO/IEC 8825-2:1995 Information technology – ASN.1 encoding rules – Specification of Packed Encoding Rules (PER).
- ISO/IEC 9594-2:1997 Information technology – Open Systems Interconnection – The Directory: Models.
- ISO/IEC 9594-6:1997 Information technology – Open Systems Interconnection – The Directory: Selected attribute types.
- ISO/IEC 9594-8:1997 Information technology - Open Systems Interconnection - The Directory: Authentication framework.
- RSA Laboratories PKCS #1 v2.0: RSA Cryptography Standard.
- RSA Laboratories PKCS #3 v1.4: Diffie-Hellman Key-Agreement Standard.
- RSA Laboratories PKCS #5 v2.0: Password-Based Cryptography Standard.
- RSA Laboratories PKCS #7 v1.5: Cryptographic Message Syntax Standard.
- RSA Laboratories PKCS #8 v1.2: Private Key Information Syntax Standard.
- RSA Laboratories PKCS #11 v2.01: Cryptographic Token Interface Standard.
- RSA Laboratories PKCS #12 v1.0 (DRAFT): Personal Information Exchange Syntax Standard.
- Wireless Application Protocol: Wireless Transport Layer Security Protocol Specification, version 30-Apr-1998 (WTLS).
- D. Atkins, W. Stallings & P. Zimmermann, "*PGP Message Exchange Formats*," IETF RFC 1991, August 1996.
- T. Berners-Lee, R. Fielding, L. Masinter, "*Uniform Resource Identifiers (URI): Generic Syntax,*" IETF RFC 2396, August 1998.


Copyright © 1999 RSA Laboratories.

- S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels, " IETF RFC 2119, March 1997.
- J. Linn, "*Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures,*" IETF RFC 1421, February 1993.
- D. Solo, R. Housley, W. Ford, T. Polk, "*Internet X.509 Public Key Infrastructure Certificate and CRL Profile,*" IETF RFC 2459, January 1999.
- F. Yergeau, "*UTF-8, a transformation format of ISO 10646,*" IETF RFC 2279, January 1998.
- ANSI X3.4-1968: Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII).
- ANSI X9.42-1999 (DRAFT): Public Key Cryptography for The Financial Service Industry: Agreement of Symmetric Keys on Using Diffie-Hellman and MQV Algorithms.
- ANSI X9.62-1998: Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).


# 3 Definitions

**AID:** Application Identifier. A data element that identifies an application in a card. An application identifier may contain a registered application provider number in which case it is a unique identification for the application. If it contains no application provider number, then this identification may be ambiguous.

**ALW:** Always. Access condition indicating a given function is always accessible.

**ANSI:** American National Standards Institute. An American standards body.

**APDU:** Application protocol data unit. A message between the card and the host computer.

**Application:** The implementation of a well-defined and related set of functions that perform useful work on behalf of the user. It may consist of software and or hardware elements and associated user interfaces.

**Application provider:** An entity that provides an application.

**ASN.1 object:** Abstract Syntax Notation object as defined in ISO/IEC 8824. A formal syntax for describing complex data objects.

Copyright © 1999 RSA Laboratories.

**ATR:** Answer-to-Reset. Stream of data sent from the card to the reader in response to a RESET condition.

**AUT:** Authenticated. Access condition indicating that a function is only available to entities that have been authenticated (typically through a cryptographic protocol involving the successful encryption of a challenge or a CHV, see below).

**BCD:** 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. E.g. decimal 92 would be encoded as the eight bit sequence 1001 0010.

**BER:** Basic Encoding Rules. Rules for encoding an ASN.1 object into a byte sequence.

**Cardholder:** The person or entity presenting a smart card for use.

**Card Issuer:** The organization or entity that owns and provides a smart card product.

**CHV:** CardHolder Verification. Also called the PIN. Typically a 4 to 8 digit number entered by the cardholder to verify that the cardholder is authorized to use the card.

**Command:** A message sent by the terminal to the card that initiates an action and solicits a response from the card.

**Command/response pair:** Set of two messages: a command to the card followed by a response from the card.

**Cryptogram:** Result of a cryptographic operation.

**Data element:** Item of information as seen at the interface between a token and an application for which are defined a name, a description of logical content, a format and a coding. Defined in ISO/IEC 7816-4.

**Data unit:** The smallest set of bits that can be unambiguously referenced. Defined in ISO/IEC 7816-4.

**DER:** Distinguished Encoding Rules for encoding ASN.1 objects in byte-sequences. A special case of BER.

**DF:** Dedicated file. File containing file control information, and, optionally, memory available for allocation. It may be the parent of elementary files and/or other dedicated files. Defined in ISO/IEC 7816-4.

**DIR file:** Directory file. An optional elementary file containing a list of applications supported by the card and optional related data elements. Defined in ISO/IEC 7816-5.

Copyright © 1999 RSA Laboratories.

**DO:** Data Object. Information as seen at the interface between a card and an application. Consists of a tag, a length and a value (i.e., a data element). Defined in ISO/IEC 7816-4.

**EF:** Elementary file. A set of data units or records that share the same identifier. It cannot be a parent of another file. Defined in ISO/IEC 7816-4.

**File control information (FCI):** Logical, structural, and security attributes of a file as defined in ISO/IEC 7816-4 and in FCD ISO/IEC 7816-9.

**File identifier:** A 2-byte binary value used to address a file on a smart card.

**Function:** A process accomplished by one or more commands and resultant actions that are used to perform all or part of a transaction.

**ICC:** Integrated Circuit Card. Another name for a smart card.

**IEC:** International Electrotechnical Commission.

**Internal Elementary File:** Elementary file for storing data interpreted by the card.

**ISO:** International Organization for Standardization

**Level:** Number of DFs in the path to a file, starting with the path from the master file.

**Memory Card:** A card with a simple memory chip with read and write capacity.

**Message:** String of bytes transmitted by the internal device to the card or vice versa, excluding transmission-control characters.

**MF:** Master file. Mandatory unique dedicated file representing the root of the structure. The MF typically has the file identifier $3F00_{16}$.

**NEV:** An access condition indicating a given function is never accessible.

**Nibble:** Half a byte. The most significant nibble of a byte consists of bits $b_8$ $b_7$ $b_6$ $b_5$ and the least significant of bits $b_4$ $b_3$ $b_2$ $b_1$.

**Parent file:** The MF or DF immediately preceding a given file within the hierarchy.

**Password:** Data that may be required by the application to be presented to the card by its user before data can be processed.

**Path:** Concatenation of file identifiers without delimitation. The Path type is defined in ISO/IEC 7816-4 sub-clause 5.1.2. If the path starts with the MF identifier (0x3F00), it is an absolute path; otherwise it is a relative path. A relative path shall start with the identifier '0x3FFF' or with the identifier of the current DF.

Copyright © 1999 RSA Laboratories.

**PER:** Packed Encoding Rules for encoding ASN.1 objects in byte sequences. A special case of BER.

**PIN:** Personal Identification Number. See CHV.

**PIN Pad:** An arrangement of alphanumeric and command keys to be used for PIN entry.

**Provider:** Authority who has or who obtained the rights to create the MF or a DF in the card.

**Reader:** As used in this specification, refers to a PC peripheral device that supports bi-directional I/O to an ISO/IEC 7816 standard ICC.

**Record:** String of bytes that can be handled as a whole by the card and referenced by a record number or by a record identifier.

**Record Identifier:** Value associated with a record that can be used to reference that record. Several records may have the same record identifier within an EF.

**Record number:** A sequential number assigned to each record that uniquely identifies the record within its EF.

**Response:** A message returned by the ICC to the terminal after the processing of a command message received by the ICC.

**RID:** Registered application provider identifier.

**Stored Value Card:** A smart card that stores non-bearer information like e-cash.

**Template:** Value field of a constructed data object, defined to give a logical grouping of data objects. Defined in ISO/IEC 7816-6.

**Token:** In this specification, a portable device capable of storing persistent data.

**Tokenholder:** Analogous to cardholder.

**Uniform Resource Identifiers:** a compact string of characters for identifying an abstract or physical resource. Described in RFC 2396.

The key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in IETF RFC 2119.

Copyright © 1999 RSA Laboratories.

# 4 Symbols and Abbreviations

**BER** Basic Encoding Rules
**DER** Distinguished Encoding Rules
**DF** Dedicated File
**DF(X)** Dedicated file with name 'X'
**DO** Data Object
**EF** Elementary File
**EF(X)** Elementary file with name 'X'
**FCD** Final Committee Draft
**IDO** Interindustry Data Object
**MF** Master File
**ODF** Object Directory File
**PIN** Personal Identification Number
**TLV** Tag-Length-Value
**URL** Uniform Resource Locator (a class of uniform resource identifiers)

In this document, ASN.1 types, definitions and values are written in **`bold courier`**.

# 5 General Overview

This document defines the PKCS #15 Cryptographic Token Information Format. The format specifies how keys, certificates and application-specific data may be stored on an ISO/IEC 7816 compliant IC card or other media. It has the following characteristics:

- Dynamic structure enables implementations on a wide variety of media, including stored value cards
- When implemented on an IC card, it allows multiple applications to reside on the card (even multiple PKCS #15 applications)
- Supports storage of any PKCS #11 objects (keys, certificates and data)
- 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[1]' (See Section 6.5) which provides a layer of indirection between objects on the token and the actual format of these objects.

---

[1] Not to be misunderstood with ISO/IEC dedicated files 'DFs'.

Copyright © 1999 RSA Laboratories.

## 5.1 Object Model

### 5.1.1 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 tokens.

The following is a figure of the PKCS #15 object hierarchy:

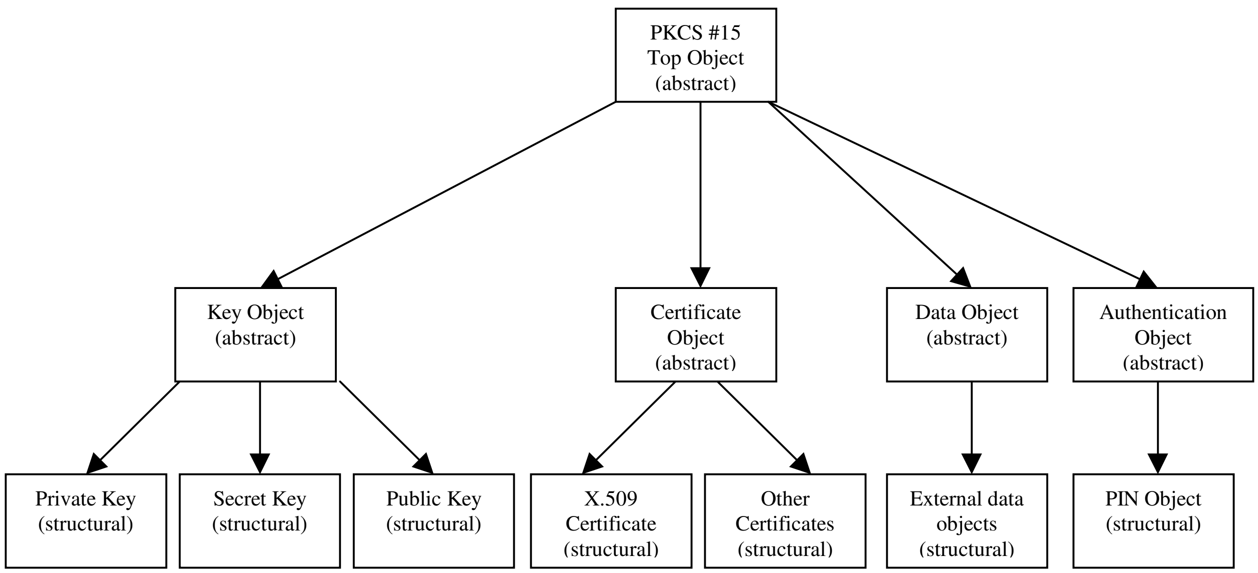


**Figure 1:** PKCS #15 Object hierarchy (instances of abstract object classes does not exist on tokens).

### 5.1.2 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 7.

### 5.1.3 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*. Conditional access (from a cardholder's perspective) is achieved

Copyright © 1999 RSA Laboratories.

with PINs[2]. 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. This version of this document does not distinguish between different ways of accessing objects; this feature may be added in a future version though.

# 6 IC card File Format

This section describes how to implement the PKCS #15 application on IC cards.

## 6.1 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 each application provider's 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 that 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 (ISO/IEC 7816-4 and perhaps the FCD of ISO/IEC 7816-8) 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. PKCS #15 is therefore a "card-edge" specification.

## 6.2 IC card requirements

This section of this document requires that compliant tokens 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).

---

[2] Other authentication methods, such as biometrics and/or challenge-response mechanisms are conceivable but out of scope for this version of this document.

Copyright © 1999 RSA Laboratories.

### 6.3 Card File Structure

A card supporting this specification will have the following layout:

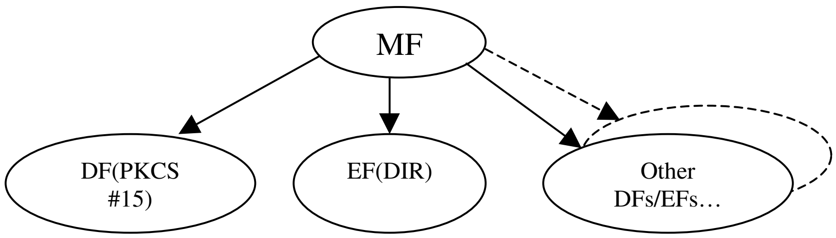


**Figure 2:** Typical PKCS #15 Card Layout[3].

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:

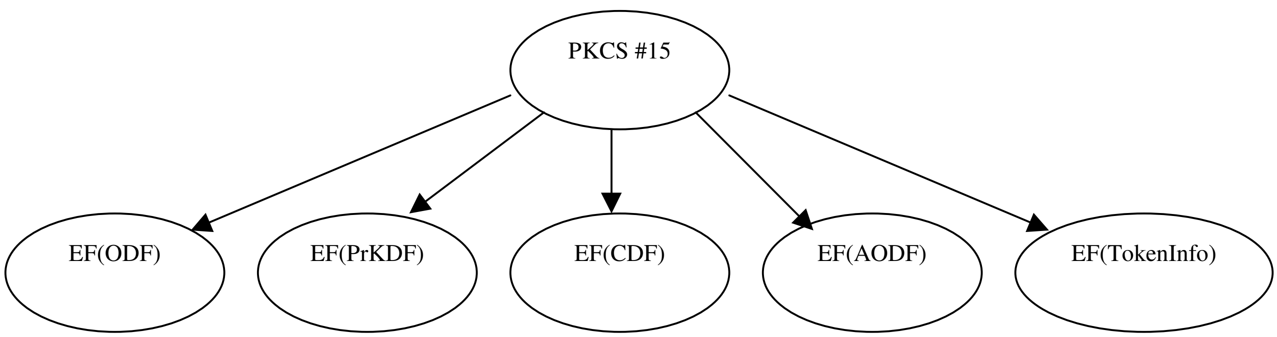


**Figure 3:** Contents of DF(PKCS15) (Example).

The contents and purpose of each file and directory is described below.

### 6.4 MF directory contents

This section describes some EFs of the IC card's master directory, MF. Currently, only one EF in the MF might be affected by the PKCS #15 application, EF(DIR).

#### 6.4.1 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 '0x61') for a PKCS15 application shall at least contain the following DOs:

---

[3] 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.

Copyright © 1999 RSA Laboratories.

- Application Identifier (tag '0x4F'), value defined in this document
- Path (tag '0x51'), 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 '0x73') and the 'Application label' data object (tag '0x50'). 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 value of the ASN.1 type **`PKCS15DDO`**:

```
PKCS15DDO ::= SEQUENCE {
    oid           OBJECT IDENTIFIER,
    odfPath       PKCS15Path OPTIONAL,
    tokenInfoPath [0] PKCS15Path OPTIONAL,
    unusedPath    [1] PKCS15Path 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 6.5). 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 record in EF(DIR) must be a value of the following ASN.1 type, conformant with ISO/IEC 7816-5:

```
PKCS15DIRRecord ::= [APPLICATION 1] SEQUENCE {
    aid   [APPLICATION 15] OCTET STRING,
    label [APPLICATION 16] UTF8String OPTIONAL,
    path  [APPLICATION 17] OCTET STRING,
    ddo   [APPLICATION 19] PKCS15DDO 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 Appendix C.

### 6.5 PKCS #15 Application Directory Contents

This section describes the EFs of the PKCS #15 application directory, DF(PKCS15).

#### 6.5.1 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

Copyright © 1999 RSA Laboratories.

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 7.2.

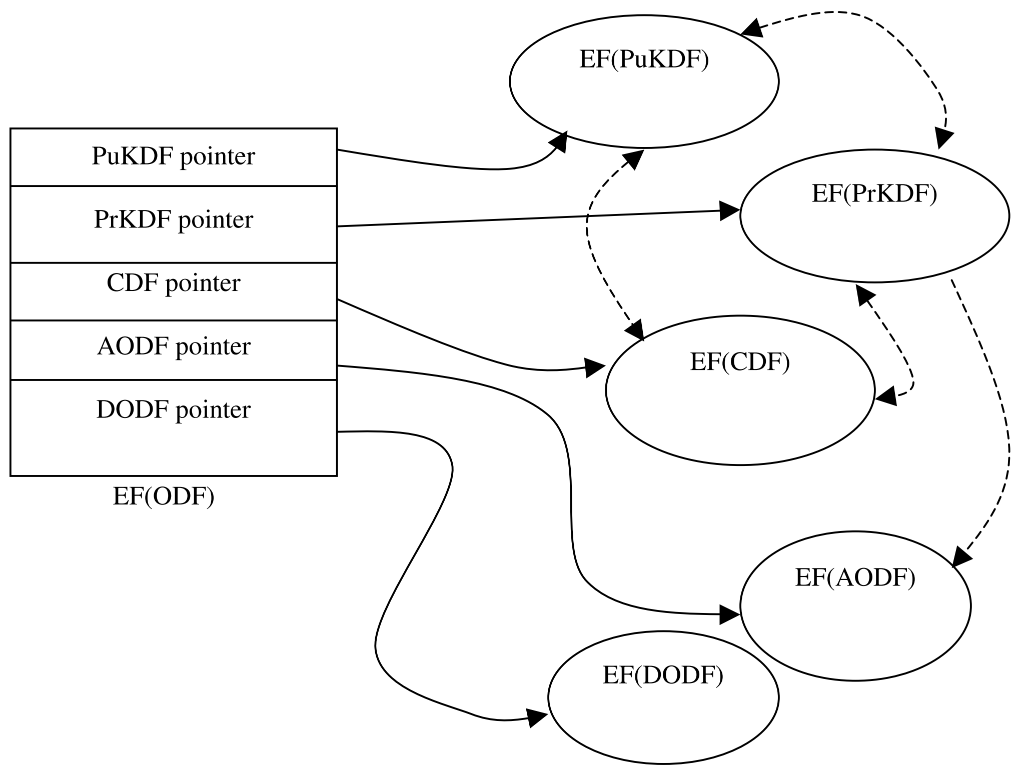


**Figure 4:** EF(ODF) points to other EFs. Dashed arrows indicate cross-references.

### 6.5.2 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 7.3.

Copyright © 1999 RSA Laboratories.

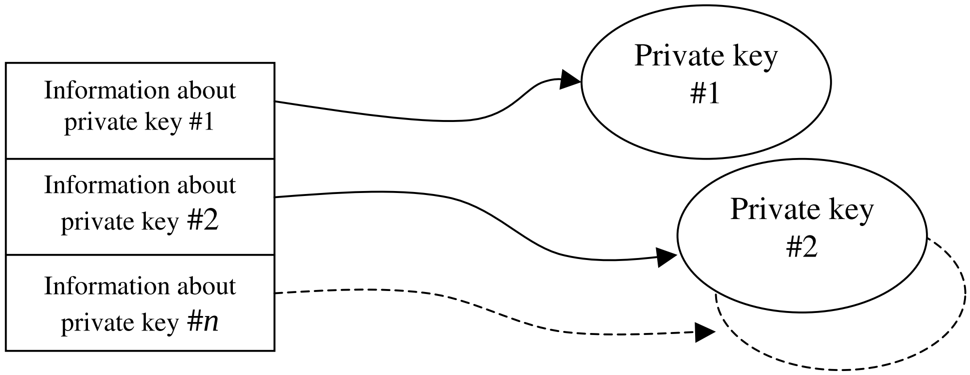


**Figure 5:** EF(PrKDF) contains private key attributes and pointers to the keys

### 6.5.3 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[4] (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 7.4.

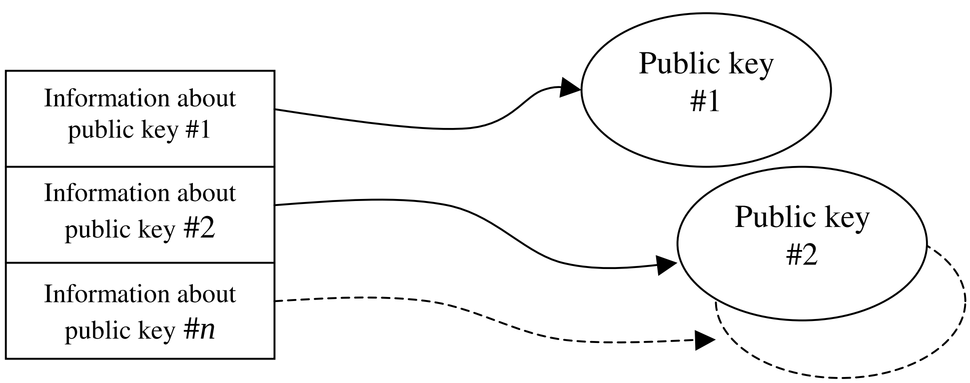


**Figure 6:** EF(PuKDF) contains public key attributes and pointers to the keys

[4] The same is true when a certificate object on the token contains the public key; in this case 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.

Copyright © 1999 RSA Laboratories.

### 6.5.4 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 7.5.

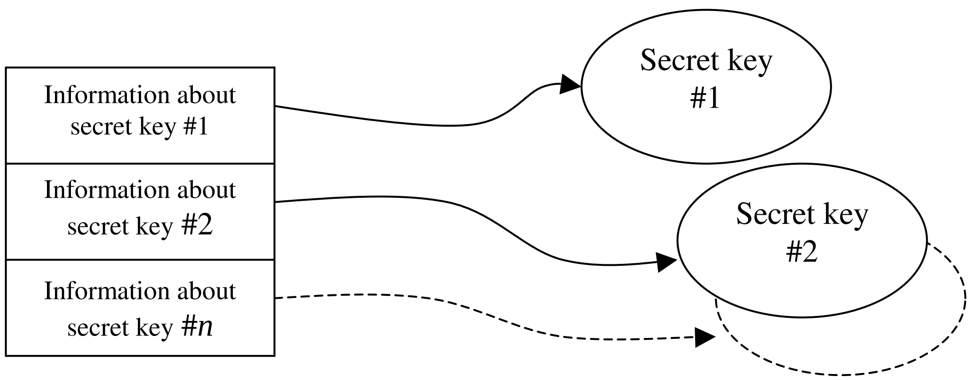


**Figure 7:** EF(SKDF) contains secret key attributes and pointers to the keys

### 6.5.5 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 7.6.

Copyright © 1999 RSA Laboratories.

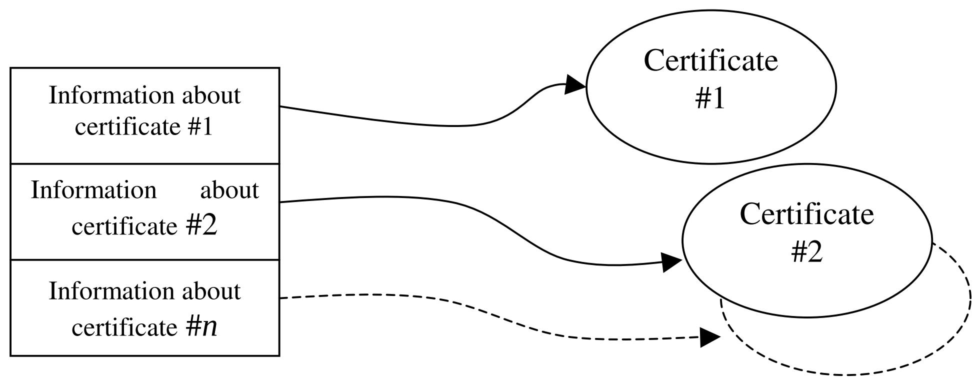


**Figure 8:** EF(CDF) contains certificate attributes and pointers to the certificates

### 6.5.6 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 7.7.

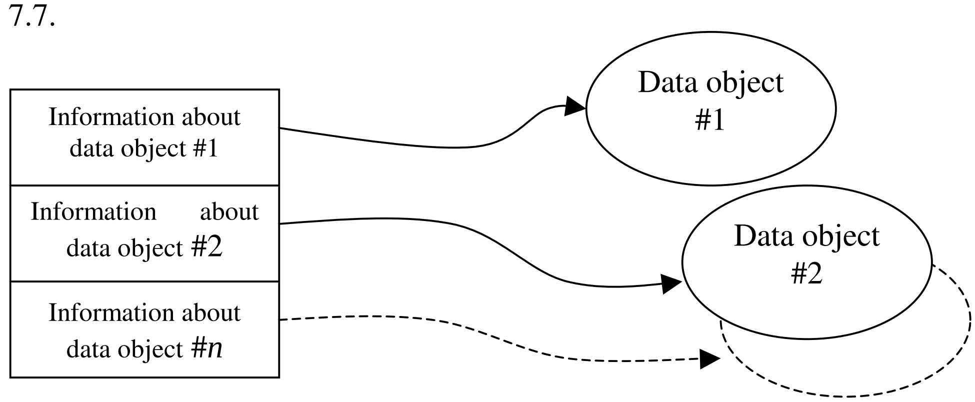


**Figure 9:** EF(DODF) contains data object attributes and pointers to the data objects.

### 6.5.7 Authentication Object Directory Files (AODFs)

These elementary files can be regarded as directories of authentication objects (e.g. PINs) 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

Copyright © 1999 RSA Laboratories.

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 the normal case there will only be one. The authentication objects themselves may reside anywhere on the card. The ASN.1 syntax for the contents of the AODFs is described in Section 7.8.

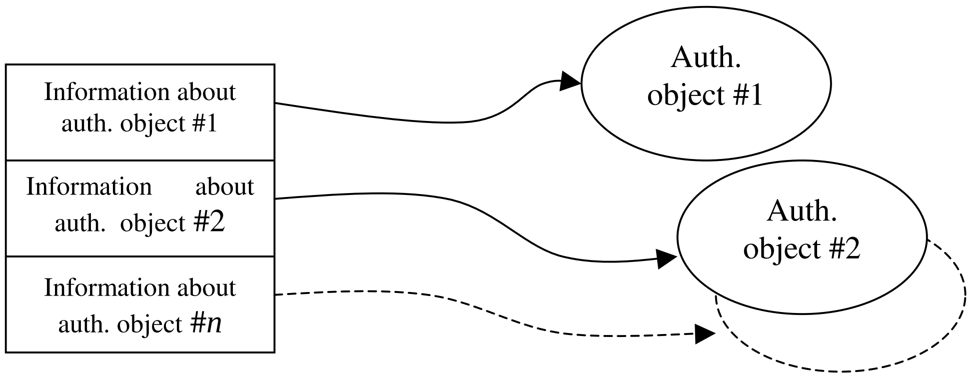


**Figure 10:** EF(AODF) contains authentication object attributes and pointers to the authentication objects

### 6.5.8 EF(TokenInfo)

The mandatory TokenInfo elementary file shall contain generic information about the token as such and it's capabilities, as seen by the PKCS15 application. This information includes the token serial number, supported file types, algorithms implemented on the token, etc. The ASN.1 syntax for the contents of the TokenInfo file is described in detail in Section 7.9.

### 6.5.9 EF(UnusedSpace)

The optional UnusedSpace elementary file 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 6.8. The file shall consist of DER-encoded records each with the following ASN.1 syntax:

```
PKCS15UnusedSpace ::= SEQUENCE {
    path   PKCS15Path (WITH COMPONENTS
               {..., index PRESENT, length PRESENT}),
    authId PKCS15Identifier 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.

Copyright © 1999 RSA Laboratories.

The **`authID`** component, described in more detail in Section 7.1.7, signals that the unused space is in a file protected by a certain authentication object.

### 6.5.10 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 7.

## 6.6 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.

| **File** | **DF** | **File Identifier (relative to nearest DF)** |
|---|---|---|
| MF | X | 0x3F00 (ISO/IEC 7816-4) |
| DIR | | 0x2F00 (ISO/IEC 7816-4) |
| PKCS15 | X | Decided by application issuer (AID is RID ‖ "PKCS-15") |
| ODF | | 0x5031 by default (but see also Section 6.4.1) |
| TokenInfo | | 0x5032 by default (but see also Section 6.4.1) |
| UnusedSpace | | 0x5033 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) | | 0x5034 - 0x5100 (Reserved for future use) |

**Table 1:** File Identifiers

## 6.7 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 6.4.1 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

Copyright © 1999 RSA Laboratories.

application label (tag '0x50') also be present to simplify the man-machine interface (e.g. vendor name in short form). See also Section 6.4.1.

### 6.7.1 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 Identifier' as specified in ISO/IEC 7816-5. 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) shall 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`

## 6.8 Object Management

Although the record-oriented format described in this document simplifies the problem of managing objects in the PKCS #15 application, it does not eliminate it. This section contains some guidelines for dealing with management (adding, removing and modifying) of PKCS #15 objects.

### 6.8.1 Adding (Creating) new objects

The UnusedSpace file may be used to find suitable unused space on a token. 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.

Copyright © 1999 RSA Laboratories.

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.

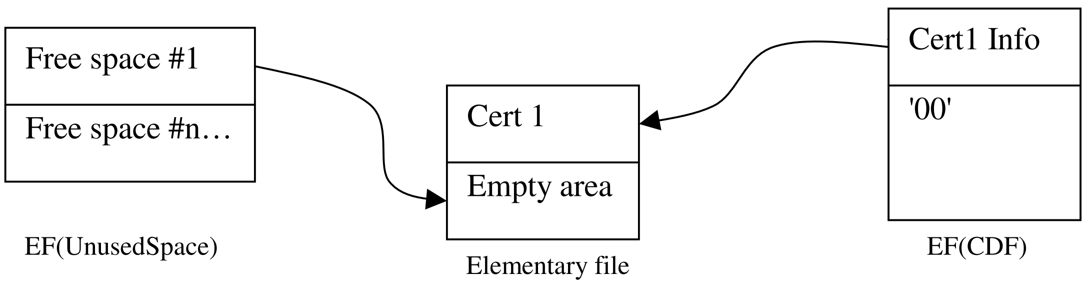


**Figure 11:** Before adding a new certificate

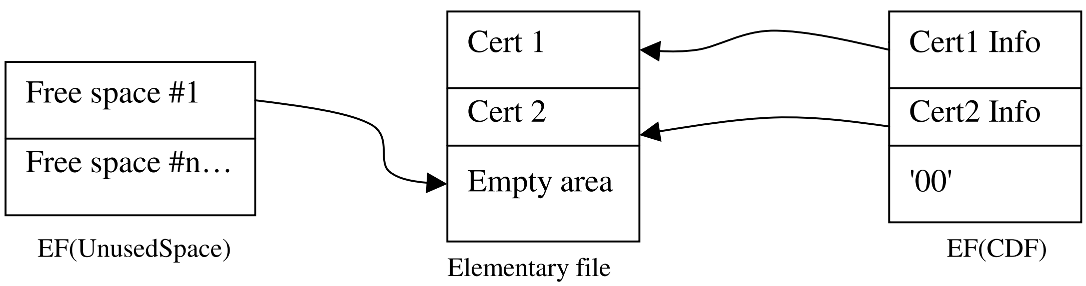


**Figure 12:** After adding a new certificate

### 6.8.2 Removing objects

Once again, sufficient privileges are assumed. In particular, the object in question must be ‘modifiable’ (see Section 7.1.7), and if it is a ‘private’ object (again, see Section 7.1.7), 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

Copyright © 1999 RSA Laboratories.

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.

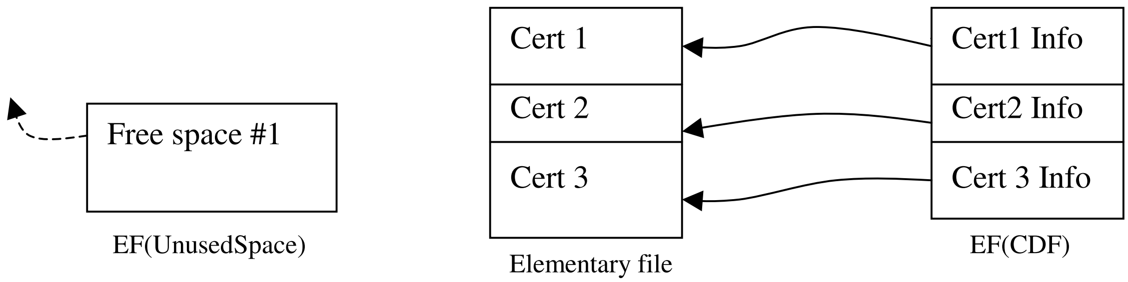


**Figure 13:** Before removing certificate 2

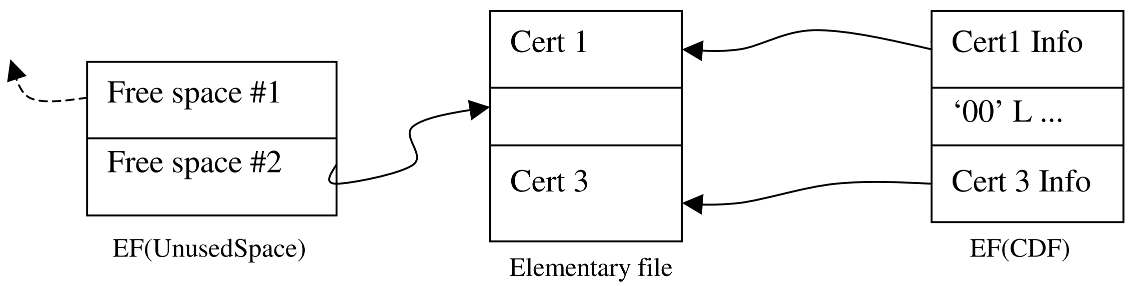


**Figure 14**: After removing certificate 2

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.

### 6.8.3 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.

## 7 Information Syntax in ASN.1

This section contains a detailed description on ASN.1 constructs to be used in PKCS #15 tokens. This section applies to ISO/IEC 7816-4 compliant IC card implementations as

Copyright © 1999 RSA Laboratories.

well as other implementations. If nothing else is mentioned, DER-encoding of values is assumed.

## 7.1 Basic ASN.1 defined types

### 7.1.1 PKCS15Identifier

```
PKCS15Identifier ::= OCTET STRING (SIZE (0..pkcs15-ub-identifier))
```

The **PKCS15Identifier** type is a constrained version of PKCS #11's CKA_ID. It is a token-internal identifier. For cross-reference purposes, two or more objects may have the same **PKCS15Identifier** value. One example of this is a private key and one or more corresponding certificates.

### 7.1.2 PKCS15Reference

```
PKCS15Reference ::= INTEGER (0..pkcs15-ub-reference)
```

This type is used for generic reference purposes.

### 7.1.3 PKCS15Label

```
PKCS15Label ::= UTF8String (SIZE(0..pkcs15-ub-label))
```

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

### 7.1.4 PKCS15ReferencedValue and PKCS15Path

```
PKCS15ReferencedValue ::= CHOICE {
        path PKCS15Path,
        url  PrintableString
        }

PKCS15Path ::= SEQUENCE {
        path    OCTET STRING,
        index   INTEGER (0..pkcs15-ub-index) OPTIONAL,
        length  INTEGER (0..pkcs15-ub-index) OPTIONAL
        } (WITH COMPONENTS {..., index PRESENT, length PRESENT}|
           WITH COMPONENTS {..., index ABSENT, length ABSENT})
```

A **PKCS15ReferencedValue** is a reference to a PKCS15 object value of some kind. This can either be some external reference (captured by the **url** identifier) or a reference to a file on the token (the **path** identifier). In the **PKCS15Path** 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

Copyright © 1999 RSA Laboratories.

'`0`' in a "`READ RECORD`" command). Lengths of fixed records may be found in the **`PKCS15TokenInfo`** file as well (see Section 7.9).

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[5].

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 IETF RFC 2396.

### 7.1.5 PKCS15ObjectValue

```
PKCS15ObjectValue { Type } ::= CHOICE {
        indirect  PKCS15ReferencedValue,
        direct    [0] Type,
        ... -- For future extensions
        } (CONSTRAINED BY {-- if indirection is being used,
           -- then it is expected that the reference points
           -- either to an object of type -- Type -- or (key
           -- case) to a card-specific key file --})
```

The **`PKCS15ObjectValue`** type is intended to catch the choice which can be made between storing a particular PKCS #15 object (key, certificate, etc) 'in-line' or by indirect reference (i.e. by pointing to another location where the value resides). 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.

### 7.1.6 PKCS15PathOrObjects

```
PKCS15PathOrObjects {ObjectType} ::= CHOICE {
        path      PKCS15Path,
        objects   [0] SEQUENCE OF ObjectType,
        ... -- For future extensions
        }
```

---

[5] 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 **`PKCS15CommonKeyAttributes.keyReference`** field shall be used for access to the key, and the presence of the **`PKS15Path.index`** and **`PKCS15Path.length`** depends on the card issuer's discretion (they are not needed for card usage purposes, but may be used for modification purposes).

Copyright © 1999 RSA Laboratories.

The **`PKCS15PathOrObjects`** 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 cards 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.

### 7.1.7 PKCS15CommonObjectAttributes

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

```
PKCS15CommonObjectAttributes ::= SEQUENCE {
        label   PKCS15Label OPTIONAL,
        flags   PKCS15CommonObjectFlags OPTIONAL,
        authId  PKCS15Identifier OPTIONAL,
        ... -- For future extensions
        } (CONSTRAINED BY {-- authId must be present in the IC card
        -- case if flags.private is set. It must equal an
        -- authID in one AuthRecord in the AODF -- })

PKCS15CommonObjectFlags ::= BIT STRING {
        private(0),
        modifiable (1)
        }
```

The **`label`** is the equivalent of the CKA_LABEL present in PKCS #11, 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 6.5.7).

### 7.1.8 PKCS15CommonKeyAttributes

This type contains all attributes common to PKCS #15 keys, except for the **`PKCS15CommonObjectAttributes`**.

Copyright © 1999 RSA Laboratories.

```
PKCS15CommonKeyAttributes ::= SEQUENCE {
        iD             PKCS15Identifier,
        usage          PKCS15KeyUsageFlags,
        native         BOOLEAN DEFAULT TRUE,
        accessFlags    PKCS15KeyAccessFlags OPTIONAL,
        keyReference   PKCS15Reference OPTIONAL,
        startDate      GeneralizedTime OPTIONAL,
        endDate        [0] GeneralizedTime OPTIONAL,
        ... -- For future extensions
        }

PKCS15KeyUsageFlags ::= BIT STRING {
        encrypt          (0),
        decrypt          (1),
        sign             (2),
        signRecover      (3),
        wrap             (4),
        unwrap           (5),
        verify           (6),
        verifyRecover    (7),
        derive           (8),
        nonRepudiation   (9)
        }

PKCS15KeyAccessFlags ::= BIT STRING {
        sensitive        (0),
        extractable      (1),
        alwaysSensitive  (2),
        neverExtractable(3),
        local            (4)
        }
```

The **iD** field must be unique for each key stored in the token, except when a public key object and the corresponding private key object are stored on the token. In this case, the keys must share the same identifier (which may also be shared with a certificate object, see Section 7.1.13).

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 ISO/IEC 9594-8 (X.509) **keyUsage** flags for public keys, PKCS #15 flags for public keys, and PKCS #15 flags for private keys, use the following table:

Copyright © 1999 RSA Laboratories.

| **Key usage flags for public keys in X.509 public key certificates** | **Corresponding PKCS15KeyUsageFlags for public keys** | **Corresponding PKCS15KeyUsageFlags 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 |

**Table 2 :** Mapping between PKCS #15 key usage flags and X.509 keyUsage extension flags[6]

The **`native`** field identifies whether the token 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).

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

### 7.1.9 PKCS15CommonPrivateKeyAttributes

This type contains all attributes common to PKCS #15 private keys, except for **`PKCS15CommonKeyAttributes`** and **`PKCS15CommonObjectAttributes`**.

```
PKCS15CommonPrivateKeyAttributes ::= SEQUENCE {
        subjectName    Name OPTIONAL,
        keyIdentifiers [0] SEQUENCE OF PKCS15KeyIdentifier OPTIONAL,
        ... -- For future extensions
        }
```

[6] Implementations should verify that usage of key usage flags on a token is sound, i.e. that all key usage flags for a particular key pair is consistent with Table 2.

Copyright © 1999 RSA Laboratories.

```
PKCS15KeyIdentifier ::= SEQUENCE {
   idType  PKCS15KEY-IDENTIFIER.&id    ({PKCS15KeyIdentifiers}),
   idValue PKCS15KEY-IDENTIFIER.&Value ({PKCS15KeyIdentifiers}{@idType})
}
```

The motivation for the fields of the **PKCS15CommonPrivateKeyAttributes** 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 token that should be used for decrypting the session key. In messages based on the PKCS #7 format, the **IssuerAndSerialNumber** construct may be used, in other schemes other types may be used. This version of this document defines four possible ways of identifying a key (for a definition of the ASN.1 class **PKCS15KEY-IDENTIFIER** see Section 9):

- **pkcs15IssuerAndSerialNumber**: 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.

- **pkcs15SubjectKeyIdentifier**: The value of this type must be an **OCTET STRING** with the same value as the corresponding certificate extension in an X.509 v3 certificate which contains the public key associated with the private key.

- **pkcs15IssuerAndSerialNumberHash**: As for **pkcs15IssuerAndSerialNumber**, but the value is an **OCTET STRING** which contains a SHA-1 hash value of this information in order to preserve space.

- **pkcs15SubjectPublicKeyHash**: A hash for the public key associated with the private key. In the RSA case, the modulus of the public key shall be used[7], 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 **PKCS15CommonKeyAttributes.iD.** This may simplify lookups of certificate – private key pairs.

[7] Note: This is different from the hash method used e.g. in IETF RFC 2459, but it serves the purpose of being independent of certificate format – alternative certificate formats not DER-encoding the public key has been proposed.

Copyright © 1999 RSA Laboratories.

### 7.1.10 PKCS15CommonPublicKeyAttributes

This type contains all attributes common to PKCS #15 public keys, except for **`PKCS15CommonKeyAttributes`** and **`PKCS15CommonObjectAttributes`**.

```
PKCS15CommonPublicKeyAttributes ::= SEQUENCE {
        subjectName Name OPTIONAL,
        ... -- For future extensions
        }
```

The motivation for the fields of the **`PKCS15CommonPublicKeyAttributes`** 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.

### 7.1.11 PKCS15CommonSecretKeyAttributes

This type contains all attributes common to PKCS #15 secret keys, except for **`PKCS15CommonKeyAttributes`** and **`PKCSCommonObjectAttributes`**.

```
PKCS15CommonSecretKeyAttributes ::= SEQUENCE {
        keyLen      INTEGER OPTIONAL, –- keylength (in bits)
        ... -- For future extensions
        }
```

The motivation for the fields of the **`PKCS15CommonSecretKeyAttributes`** 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.

### 7.1.12 PKCS15KeyInfo

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 token). The **`supportedOperations`** field is optional and can be absent on tokens, which do not support any operations with the key. Note the distinction between **`PKCS15KeyUsageFlags`** and **`PKCS15KeyInfo.paramsAndOps.supportedOperations`**: The former indicates the intended *usage* of the key, the latter indicates the operations (if any) the *token* can *perform* with the key.

```
PKCS15KeyInfo {ParameterType, OperationsType} ::= CHOICE {
        reference PKCS15Reference,
        paramsAndOps SEQUENCE {
                parameters          ParameterType,
```

Copyright © 1999 RSA Laboratories.

```
                supportedOperations OperationsType OPTIONAL}
        }
```

### 7.1.13 PKCS15CommonCertificateAttributes

This type contains all attributes common to PKCS #15 certificates, except for the **`PKCS15CommonObjectAttributes`**.

```
PKCS15CommonCertificateAttributes ::= SEQUENCE {
        iD                  PKCS15Identifier,
        authority           BOOLEAN DEFAULT FALSE,
        requestId           PKCS15KeyIdentifier OPTIONAL,
        thumbprint          [0] PKCS15OOBCertHash OPTIONAL,
        ... -- For future extensions
        }
```

The **`iD`** field is only present for X509 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 **`requestId`** 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. In this case, and in addition to the identification methods defined in Section 7.1.9, a fifth alternative may be used:

- **`pkcs15IssuerKeyHash`**: A hash of the public key used to sign the requested certificate. This value may also, in the case of X.509 v3 certificates, be present in the **`authorityKeyIdentifier`** extension in the user’s certificate.

The **`thumbprint`** field is useful from a security perspective when the certificate in question is stored external to the token (the **`url`** choice of **`PKCS15ReferencedValue`**), since it enables a user to verify that no one has tampered with the certificate.

### 7.1.14 PKCS15CommonDataObjectAttributes and PKCS15ApplicationIdentifier

The **`PKCS15CommonDataObjectAttributes`** type contains all attributes common to PKCS #15 data objects, except for the **`PKCS15CommonObjectAttributes`**.

```
PKCS15CommonDataObjectAttributes ::= SEQUENCE {
        applicationName PKCS15Label OPTIONAL,
        applicationOID  OBJECT IDENTIFIER OPTIONAL,
```

Copyright © 1999 RSA Laboratories.

```
        ... -- 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 **PKCS15CommonDataObjectAttributes**.

### 7.1.15 PKCS15CommonAuthenticationObjectAttributes

This type contains all attributes common to PKCS #15 authentication objects, except for the **PKCS15CommonObjectAttributes**.

```
PKCS15CommonAuthenticationObjectAttributes ::= SEQUENCE {
        authId PKCS15Identifier,
        ... –- For future extensions
        }
```

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

### 7.1.16 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 PKCS15CommonObjectAttributes,
        classAttributes        ClassAttributes,
        subClassAttributes     [0] SubClassAttributes OPTIONAL,
        typeAttributes         [1] TypeAttributes
        }
```

## 7.2 The PKCS15Objects type

This type is a placeholder for the various object types defined in this document.

```
PKCS15Objects ::= CHOICE {
        privateKeys         [0] PKCS15PrivateKeys,
        publicKeys          [1] PKCS15PublicKeys,
        trustedPublicKeys   [2] PKCS15PublicKeys,
        secretKeys          [3] PKCS15SecretKeys,
        certificates        [4] PKCS15Certificates,
        trustedCertificates [5] PKCS15Certificates,
        usefulCertificates  [6] PKCS15Certificates,
        dataObjects         [7] PKCS15DataObjects,
```

Copyright © 1999 RSA Laboratories.

```
        authObjects         [8] PKCS15AuthObjects,
        ... -- For future extensions
        }

PKCS15PrivateKeys  ::= PKCS15PathOrObjects {PKCS15PrivateKey}
PKCS15SecretKeys   ::= PKCS15PathOrObjects {PKCS15SecretKey}
PKCS15PublicKeys   ::= PKCS15PathOrObjects {PKCS15PublicKey}
PKCS15Certificates ::= PKCS15PathOrObjects {PKCS15Certificate}
PKCS15DataObjects  ::= PKCS15PathOrObjects {PKCS15Data}
PKCS15AuthObjects  ::= PKCS15PathOrObjects {PKCS15Authentication}
```

In the IC card case, the intention is that EF(ODF) shall consist of a number of data objects (records) of type **`PKCS15Objects`**, 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 **`PKCS15PathOrObject`** type is recommended, this will result in a record-oriented ODF, which simplifies updating[8].

Examples of **`PKCS15Objects`** values can be found in Appendix C.

The **`trustedPublicKeys`** field is intended for implementations on IC cards supporting the ISO/IEC 7816-4 logical file organization. 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`** field shall point to certificates issued to the cardholder. They may or may not be possible to modify by the cardholder.

The **`trustedCertificates`** field is intended for implementations on IC cards supporting the ISO/IEC 7816-4 logical file organization. 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, and can therefore be used as trust chain origins.

[8] 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 **`PKCS15PathOrObjects`**.

Copyright © 1999 RSA Laboratories.

The **`usefulCertificates`** field 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.

## 7.3 The PKCS15PrivateKeys type

This type contains information pertaining to private key objects stored in the token. Since, in the ISO/IEC 7816-4 IC card case, the **`path`** alternative of the **`PKCS15PathOrObjects`** type is to be chosen, **`PKCS15PrivateKeys`** 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 PKCS15PrivateKey`** (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 tokens not supporting the ISO/IEC 7816-4 logical file organization, any of the **`CHOICE`** alternatives of **`PKCS15PathOrObjects`** may be used.

Examples of **`PKCS15PrivateKey`** values can be found in Appendix C.

The **`PKCS15PrivateKey`** structure is as follows:

```
PKCS15PrivateKey ::= CHOICE {
        privateRSAKey     PKCS15PrivateKeyObject {
                          PKCS15PrivateRSAKeyAttributes},
        privateECKey  [0] PKCS15PrivateKeyObject {
                          PKCS15PrivateECKeyAttributes},
        privateDHKey  [1] PKCS15PrivateKeyObject {
                          PKCS15PrivateDHKeyAttributes},
        privateDSAKey [2] PKCS15PrivateKeyObject {
                          PKCS15PrivateDSAKeyAttributes},
        privateKEAKey [3] PKCS15PrivateKeyObject {
                          PKCS15PrivateKEAKeyAttributes},
        ... -- For future extensions
        }

PKCS15PrivateKeyObject {KeyAttributes} ::= PKCS15Object {
                                 PKCS15CommonKeyAttributes,
                                 PKCS15CommonPrivateKeyAttributes,
                                 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 (**`PKCS15CommonObjectAttributes`**, **`PKCS15CommonKeyAttributes`** and **`PKCS15CommonPrivateKeyAttributes`**) and, in addition the particular key type's attributes.

Copyright © 1999 RSA Laboratories.

### 7.3.1 Private RSA key objects

```
PKCS15PrivateRSAKeyAttributes ::= SEQUENCE {
        value         PKCS15ObjectValue {PKCS15RSAPrivateKey},
        modulusLength INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo       PKCS15KeyInfo {PKCS15RSAParameters,
                          PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15RSAPrivateKey ::= SEQUENCE {
        modulus         [0] INTEGER OPTIONAL, -- n
        publicExponent  [1] INTEGER OPTIONAL, -- e
        privateExponent [2] INTEGER OPTIONAL, -- d
        prime1          [2] INTEGER OPTIONAL, -- p
        prime2          [3] INTEGER OPTIONAL, -- q
        exponent1       [4] INTEGER OPTIONAL, -- d mod (p-1)
        exponent2       [5] INTEGER OPTIONAL, -- d mod (q-1)
        coefficient     [6] 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:

- **`PKCS15PrivateRSAKeyAttributes.value`**: The value shall, in the IC card case, be a path to a file containing either a value of type **`PKCS15RSAPrivateKey`** or (in the case of a card capable of performing on-chip RSA encryption) some card specific representation of a private RSA key. As mentioned, this will be indicated in the **`PKCS15CommonKeyAttributes.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 **`PKCS15RSAPrivateKey`** 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.

- **`PKCS15PrivateRSAKeyAttributes.modulusLength`**: On many tokens, one must be able to format data to be signed prior to sending the data to the token. 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.

- **`PKCS15PrivateRSAKeyAttributes.keyInfo:`** Information about parameters that applies to this key (NULL in the case of RSA keys) and operations the token can carry out with this key. In the IC card case, the **`reference`** alternative of a **`PKCS15KeyInfo`** 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.

### 7.3.2 Private Elliptic Curve key objects

```
PKCS15PrivateECKeyAttributes ::= SEQUENCE {
```

Copyright © 1999 RSA Laboratories.

```
        value   PKCS15ObjectValue {PKCS15ECPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15ECParameters,
                              PKCS15PublicKeyOperations} OPTIONAL,

        ... -- For future extensions
        }

PKCS15ECPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

- **`PKCS15PrivateECKeyAttributes.value`**: The value shall, in the IC card case, be a path to a file containing either a value of type **`PKCS15ECPrivateKey`** or (in the case of a card capable of performing on-chip EC operations) some card specific representation of a private EC key. As mentioned, this will be indicated in the **`PKCS15CommonKeyAttributes.native`** field. In other cases, the application issuer is free to choose any alternative.

- **`PKCS15PrivateECKeyAttributes.keyInfo`**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **`reference`** alternative of a **`PKCS15KeyInfo`** 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.

### 7.3.3 Private Diffie-Hellman key objects

```
PKCS15PrivateDHKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DHPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15DHParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DHPrivateKey ::= INTEGER –- Diffie-Hellman exponent
```

The semantics of these types is as follows:

- **`PKCS15PrivateDHKeyAttributes.value`**: The value shall, in the IC card case, be a path to a file containing either a value of type **`PKCS15DHPrivateKey`** 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. As mentioned, this will be indicated in the **`PKCS15CommonKeyAttributes.native`** field. In other cases, the application issuer is free to choose any alternative.

- **`PKCS15PrivateDHKeyAttributes.keyInfo`**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **`reference`** alternative of a **`PKCS15KeyInfo`** 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.

Copyright © 1999 RSA Laboratories.

### 7.3.4 Private Digital Signature Algorithm key objects

```
PKCS15PrivateDSAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DSAPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15DSAParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DSAPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

- **PKCS15PrivateDSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **PKCS15DSAPrivateKey** or (in the case of a card capable of performing on-chip DSA operations) some card specific representation of a private DSA key. As mentioned, this will be indicated in the **PKCS15CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PKCS15PrivateDSAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **reference** alternative of a **PKCS15KeyInfo** 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.

### 7.3.5 Private KEA key objects

```
PKCS15PrivateKEAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15KEAPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15KEAParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15KEAPrivateKey ::= INTEGER
```

The semantics of these types is as follows:

- **PKCS15PrivateKEAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of the **PKCS15KEAPrivateKey** type or (in the case of a card capable of performing on-chip KEA operations) some card specific representation of a private KEA key. As mentioned, this will be indicated in the **PKCS15CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PKCS15PrivateKEAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **reference** alternative of a **PKCS15KeyInfo** must be used, and the reference

Copyright © 1999 RSA Laboratories.

shall "point" to a particular entry in EF(TokenInfo), see below. The field is not needed if the information is available through other means.

## 7.4 The PKCS15PublicKeys type

This data structure contains information pertaining to public key objects stored in the token. Since, in the IC card case, the **`path`** alternative of the **`PKCS15PathOrObjects`** type is to be chosen, **`PKCS15PublicKeys`** 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 PKCS15PublicKey`** (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 tokens not supporting the ISO/IEC 7816-4 logical file organization, any of the **`CHOICE`** alternatives of **`PKCS15PathOrObjects`** may be used.

```
PKCS15PublicKey ::= CHOICE {
        publicRSAKey     PKCS15PublicKeyObject {
                          PKCS15PublicRSAKeyAttributes},
        publicECKey  [0] PKCS15PublicKeyObject {
                          PKCS15PublicECKeyAttributes},
        publicDHKey  [1] PKCS15PublicKeyObject {
                          PKCS15PublicDHKeyAttributes},
        publicDSAKey [2] PKCS15PublicKeyObject {
                          PKCS15PublicDSAKeyAttributes},
        publicKEAKey [3] PKCS15PublicKeyObject {
                          PKCS15PublicKEAKeyAttributes},
        ... -- For future extensions
        }

PKCS15PublicKeyObject {KeyAttributes} ::= PKCS15Object {
                                 PKCS15CommonKeyAttributes,
                                 PKCS15CommonPublicKeyAttributes,
                                 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 (**`PKCS15CommonObjectAttributes, PKCS15CommonKeyAttributes`** and **`PKCS15CommonPublicKeyAttributes`**) and in addition the particular public key type's attributes.

### 7.4.1 Public RSA key objects

```
PKCS15PublicRSAKeyAttributes ::= SEQUENCE {
        value          PKCS15ObjectValue {PKCS15RSAPublicKey},
        modulusLength  INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo        PKCS15KeyInfo {PKCS15RSAParameters,
                             PKCS15PublicKeyOperations} OPTIONAL,
        ... –- For future extensions
```

Copyright © 1999 RSA Laboratories.

```
        }

PKCS15RSAPublicKey ::= SEQUENCE {
        modulus         INTEGER, -- n
        publicExponent  INTEGER  -- e
        }
```

The semantics of the fields is as follows:

- **`PKCS15PublicRSAKeyAttributes.value`**: The value shall, in the IC card case, be a path to a file containing either a value of type **`PKCS15RSAPublicKey`** 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 **`PKCS15CommonKeyAttributes.native`** field. In other cases, the application issuer is free to choose any alternative.

- **`PKCS15PublicRSAKeyAttributes.modulusLength`**: On many tokens, one must be able to format data to be encrypted prior to sending the data to the token. 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.

- **`PKCS15PublicRSAKeyAttributes.keyInfo:`** Information about parameters that applies to this key (**`NULL`** in the case of RSA keys) and operations the token can carry out with this key. In the IC card case, the **`reference`** alternative of a **`PKCS15KeyInfo`** 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.

### 7.4.2 Public Elliptic Curve key objects

```
PKCS15PublicECKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15ECPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15ECParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15ECPublicKey ::= PKCS15ECPoint
```

The semantics of these types is as follows:

- **`PKCS15PublicECKeyAttributes.value`**: The value shall, in the IC card case, be a path to a file containing either a value of type **`PKCS15ECPublicKey`** 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 **`PKCS15CommonKeyAttributes.native`** field. In other cases, the application issuer is free to choose any alternative.

- **`PKCS15PublicECKeyAttributes.keyInfo`**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card

Copyright © 1999 RSA Laboratories.

case, the **reference** alternative of a **PKCS15KeyInfo** 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.

### 7.4.3 Public Diffie-Hellman key objects

```
PKCS15PublicDHKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DHPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15DHParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DHPublicKey ::= PKCS15DiffieHellmanPublicNumber
```

The semantics of these types is as follows:

- **PKCS15PublicDHKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **PKCS15DHPublicKey** 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 **PKCS15CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PKCS15PublicDHKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **reference** alternative of a **PKCS15KeyInfo** 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.

### 7.4.4 Public Digital Signature Algorithm objects

```
PKCS15PublicDSAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DSAPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15DSAParameters,
                PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DSAPublicKey ::= INTEGER
```

The semantics of these types is as follows:

- **PKCS15PublicDSAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of type **PKCS15DSAPublicKey** 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

Copyright © 1999 RSA Laboratories.

**PKCS15CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PKCS15PublicDSAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **reference** alternative of a **PKCS15KeyInfo** 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.

### 7.4.5 Public KEA key objects

```
PKCS15PublicKEAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15KEAPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15KEAParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15KEAPublicKey ::= INTEGER
```

The semantics of these types is as follows:

- **PKCS15PublicKEAKeyAttributes.value**: The value shall, in the IC card case, be a path to a file containing either a value of the **PKCS15KEAPublicKey** type 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 **PKCS15CommonKeyAttributes.native** field. In other cases, the application issuer is free to choose any alternative.

- **PKCS15PublicKEAKeyAttributes.keyInfo**: Information about parameters that applies to this key and operations the token can carry out with this key. In the IC card case, the **reference** alternative of a **PKCS15KeyInfo** 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.

## 7.5 The PKCS15SecretKeys type

This data structure contains information pertaining to secret keys stored in the token. Since, in the IC card case, the **path** alternative of the **PKCS15PathOrObjects** type is to be chosen, **PKCS15SecretKeys** 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 PKCS15SecretKey** (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.

Copyright © 1999 RSA Laboratories.

In the case of tokens not supporting the ISO/IEC 7816-4 logical file organization, any of the **`CHOICE`** alternatives of **`PKCS15PathOrObjects`** may be used.

```
PKCS15SecretKey ::= CHOICE {
        genericSecretKey     PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc2key           [0] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc4key           [1] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        desKey           [2] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        des2Key          [3] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        des3Key          [4] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        castKey          [5] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        cast3Key         [6] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        cast128Key       [7] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc5Key           [8] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        ideaKey          [9] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        skipjackKey     [10] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        batonKey        [11] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        juniperKey      [12] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc6Key          [13] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        otherKey        [14] PKCS15OtherKey,
        ... -- For future extensions
        }

PKCS15SecretKeyObject {KeyAttributes} ::= PKCS15Object {
                         PKCS15CommonKeyAttributes,
                         PKCS15CommonSecretKeyAttributes,
                         KeyAttributes}

PKCS15OtherKey ::= SEQUENCE {
        keyType OBJECT IDENTIFIER,
        keyAttr PKCS15SecretKeyObject {PKCS15GenericSecretKeyAttributes}
        }
```

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 (**`PKCS15CommonObjectAttributes`**, **`PKCS15CommonKeyAttributes`** and **`PKCS15CommonSecretKeyAttributes`**) 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, **`PKCS15GenericSecretKeyAttributes`**, defined below.

Copyright © 1999 RSA Laboratories.

### 7.5.1 Generic secret key objects

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

```
PKCS15GenericSecretKeyAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { OCTET STRING },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

- **`PKCS15GenericSecretKeyAttributes.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 **`PKCS15ObjectValue`** type.

### 7.5.2 Tagged key objects

These key objects represent keys of various types. In the case of tokens capable of performing cryptographic computations with keys of certain types, the key representation is token specific (indicated by the **`PKCS15CommonKeyAttributes.native`** field). Otherwise, the key shall be stored as an **`OCTET STRING`**, as indicated above.

### 7.5.3 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 **`PKCS15OtherKey`** type shall contain an object identifier identifying the type of the key and the usual secret key attributes.

## 7.6 The PKCS15Certificates type

This data structure contains information pertaining to certificate objects stored in the token. Since, in the IC card case, the **`path`** alternative of the **`PKCS15PathOrObjects`** type is to be chosen, **`PKCS15Certificates`** 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 PKCS15Certificate`** (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 tokens not supporting the ISO/IEC 7816-4 logical file organization, any of the **`CHOICE`** alternatives of **`PKCS15PathOrObjects`** may be used.

Copyright © 1999 RSA Laboratories.

Examples of this type can be found in Appendix C.

```
PKCS15Certificate ::= CHOICE {
        x509Certificate              PKCS15CertificateObject {
                      PKCS15X509CertificateAttributes},
        x509AttributeCertificate [0] PKCS15CertificateObject {
                      PKCS15X509AttributeCertificateAttributes},
        spkiCertificate          [1] PKCS15CertificateObject {
                      PKCS15SPKICertificateAttributes},
        pgpCertificate           [2] PKCS15CertificateObject {
                      PKCS15PGPCertificateAttributes},
        wtlsCertificate          [3] PKCS15CertificateObject {
                      PKCS15WTLSCertificateAttributes},
        x9-68Certificate         [4] PKCS15CertificateObject {
                      PKCS15X9-68CertificateAttributes},
        ... -- For future extensions
        }

PKCS15CertificateObject {CertAttributes} ::= PKCS15Object {
                              PKCS15CommonCertificateAttributes,
                              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 (**PKCS15CommonObjectAttributes** and **PKCS15CommonCertificateAttributes**) and in addition the particular certificate type's attributes.

### 7.6.1 X.509 certificate objects

```
PKCS15X509CertificateAttributes ::= SEQUENCE {
        value        PKCS15ObjectValue { PKCS15X509Certificate },
        subject      Name OPTIONAL,
        issuer       [0] Name OPTIONAL,
        serialNumber CertificateSerialNumber OPTIONAL,
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **PKCS15X509CertificateAttributes.value**: The value shall, in the IC card case, be a **PKCS15ReferencedValue** 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.

- **PKCS15X509CertificateAttributes.subject**, **PKCS15X509CertificateAttributes.issuer** and **PKCS15X509CertificateAttributes.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.

Copyright © 1999 RSA Laboratories.

### 7.6.2 X.509 attribute certificate Objects

```
PKCS15X509AttributeCertificateAttributes ::= SEQUENCE {
        value         PKCS15ObjectValue { PKCS15AttributeCertificate },
        issuer        GeneralNames OPTIONAL,
        serialNumber  CertificateSerialNumber OPTIONAL,
        attrTypes     [0] SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **`PKCS15X509AttributeCertificateAttributes.value`**: The value shall, in the IC card case, be a **`PKCS15ReferencedValue`** 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.

- **`PKCS15X509AttributeCertificateAttributes.issuer`** and **`PKCS15X509AttributeCertificateAttributes.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.

- **`PKCS15X509AttributeCertificateAttributes.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.

### 7.6.3 SPKI (Simple Public Key Infrastructure[9]) certificate objects

```
PKCS15SPKICertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

- **`PKCS15SPKICertificateAttributes.value`**: The value shall, in the IC card case, be a **`pkcs15ReferencedValue`** identifying either a file containing a base64[10] encoded 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.

---

[9] See the Internet working draft "draft-ietf-spki-cert-structure-05.txt" for more information.

[10] See IETF RFC 1421.

Copyright © 1999 RSA Laboratories.

### 7.6.4 PGP (Pretty Good Privacy) certificate objects

```
PKCS15PGPCertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the field is as follows:

- **`PKCS15PGPCertificateAttributes.value`**: The value shall, in the IC card case, be a **`pkcs15ReferencedValue`** identifying either a file containing a base64 encoded PGP certificate[11] 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.

### 7.6.5 WTLS certificate objects

```
PKCS15WTLSCertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **`PKCS15WTLSCertificateAttributes.value`**: The value shall, in the IC card case, be a **`PKCS15ReferencedValue`** 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.

### 7.6.6 ANSI X9.68[12] lightweight certificate objects

```
PKCS15X9-68CertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }
```

The semantics of the fields is as follows:

- **`PKCS15X9-68CertificateAttributes.value`**: The value shall, in the IC card case, be a **`PKCS15ReferencedValue`** identifying either a file containing a DER or PER encoded ANSI X9.68 certificate at the given location, or a url pointing to some

---

[11] As defined in Section 7 of IETF RFC 1991.

[12] The exact format of X9.68 certificates is currently being defined by ANSI.

Copyright © 1999 RSA Laboratories.

location where the certificate in question can be found. In other cases, the application issuer is free to choose any alternative.

## 7.7 The PKCS15DataObjects type

This data structure contains information pertaining to data objects stored in the token. Since, in the IC card case, the **`path`** alternative of the **`PKCS15PathOrObjects`** type is to be chosen, **`PKCS15DataObjects`** 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 PKCS15DataObject`** (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 tokens not supporting the ISO/IEC 7816-4 logical file organization, any of the **`CHOICE`** alternatives of **`PKCS15PathOrObjects`** may be used.

Examples of this type can be found in Appendix C.

```
PKCS15Data ::= CHOICE {
        opaqueDO            PKCS15DataObject {PKCS15Opaque},
        externalIDO     [0] PKCS15DataObject {PKCS15ExternalIDO},
        oidDO           [1] PKCS15DataObject {PKCS15OidDO},
        ... -- For future extensions
        }

PKCS15DataObject {DataObjectAttributes} ::= PKCS15Object {
                                 PKCS15CommonDataObjectAttributes,
                                 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 (**`PKCS15CommonObjectAttributes`** and **`PKCS15CommonDataObjectAttributes`**) and in addition the particular data object type's attributes.

### 7.7.1 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.

```
PKCS15Opaque ::= PKCS15ObjectValue {PKCS15-OPAQUE.&Type}
```

Copyright © 1999 RSA Laboratories.

### 7.7.2 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.

```
PKCS15ExternalIDO ::= PKCS15ObjectValue {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.

### 7.7.3 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`**.

```
PKCS15OidDO ::= SEQUENCE {
        id    OBJECT IDENTIFIER,
        value PKCS15ObjectValue {PKCS15-OPAQUE.&Type}
}
```

## 7.8 The PKCS15AuthenticationObject type

This data structure, only relevant to tokens capable of authenticating token-holders, contains information about how the token-holder authentication shall be carried out. Since, in the IC card case, the **`path`** alternative of the **`PKCS15PathOrObjects`** type is to be chosen, **`PKCS15AuthenticationObject`** 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 PKCS15AuthenticationObject`** (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.

Examples of this type can be found in Appendix C.

```
PKCS15Authentication ::= CHOICE {
        pin       PKCS15AuthenticationObject { PKCS15PinAttributes },
        ... -- For future extensions, e.g. biometric authentication
            -- objects
        }

PKCS15AuthenticationObject {AuthObjectAttributes} ::= PKCS15Object {
                             PKCS15CommonAuthenticationObjectAttributes,
                             NULL,
                             AuthObjectAttributes}
```

Copyright © 1999 RSA Laboratories.

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 (**`PKCS15CommonObjectAttributes`** and **`PKCS15CommonAuthenticationObjectAttributes`**) and in addition the particular authentication object type's attributes. Each authentication object must have a distinct **`PKCS15CommonAuthenticationObjectAttributes.authID`**, enabling unambiguous authentication object lookup for private objects.

### 7.8.1 Pin Objects

```
PKCS15PinAttributes ::= SEQUENCE {
        pinFlags      PKCS15PinFlags,
        pinType       PKCS15PinType,
        minLength     INTEGER
                    (pkcs15-lb-minPinLength..pkcs15-ub-minPinLength),
        storedLength  INTEGER
                    (pkcs15-lb-minPinLength..pkcs15-ub-storedPinLength),
        maxLength     INTEGER OPTIONAL,
        pinReference  [0] PKCS15Reference OPTIONAL,
        padChar       OCTET STRING (SIZE(1)) OPTIONAL,
        lastPinChange GeneralizedTime OPTIONAL,
        path          PKCS15Path OPTIONAL,
        ... -- For future extensions
        }

PKCS15PinFlags ::= 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)
        } (CONSTRAINED BY { -- 'unblockingPin' and 'soPIN' cannot both
           -- be set -- })

PKCS15PinType ::= ENUMERATED {bcd, ascii-numeric, utf8, ...
                  -- bcd = one nibble contains one digit
                  -- ascii-numeric = one byte contains one ASCII digit
                  -- utf8 = password is stored in UTF8 encoding
                  }
```

The semantics of these fields is as follows:

- **`PKCS15PinAttributes.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 token (see below)
    - **`local`**, meaning that the PIN is local to the PKCS #15 application[13]

[13] A pin which is not 'local' is considered 'global'. A local PIN may only be used to protect data within the PKCS #15 application. For a local PIN the lifetime of a verification is not guaranteed and it may have to be

Copyright © 1999 RSA Laboratories.

- **change-disabled**, meaning that it is not possible to change the PIN
- **unblock-disabled**, meaning that it is not possible to unblock the PIN
- **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 token
- **unblockingPin**, meaning that the PIN may be used for unblocking purposes
- **soPin**, meaning that the PIN is a Security Officer PIN (in the PKCS #11 sense)[14]
- **disable-allowed**, meaning that the PIN might be disabled.

- **PKCS15PinAttributes.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 encoded digit) or
  - **utf8** (Each character is encoded in accordance with UTF8).
- **PKCS15PinAttributes.minLength**: Minimum length (in characters) of new PINs (if allowed to change).
- **PKCS15PinAttributes.storedLength**: Stored length on token (in bytes). Used to deduce the number of padding characters needed.
- **PKCS15PinAttributes.maxLength**: On some tokens, PINs are not padded, and there is therefore a need to know the maximum PIN length (in characters) allowed.
- **PKCS15PinAttributes.pinReference**: This optional field is a token-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.
- **PKCS15PinAttributes.padChar**: Padding character to use (usually '$FF_{16}$' or '$00_{16}$'). Not needed if **pinFlags** indicates that padding is not needed for this token. If the **PKCS15PinAttributes.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.
- **PKCS15PinAttributes.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)

---

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.

[14] 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.

Copyright © 1999 RSA Laboratories.

'**`000000000000Z`**'. As another example, a PIN changed on January 6, 1999 at 1934 (7 34 PM) UTC would have a **`lastPinChange`** value of '**`19990106193400Z`**'.

- **`PKCS15PinAttributes.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 token-holder verification must always be possible to perform without a prior '`SELECT`' operation.

#### 7.8.1.1 Transforming a supplied PIN

The steps taken to transform a user-supplied PIN to something presented to the token 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 [RFC 2279]: *x = UTF8(PIN).* Then, if the **`case-sensitive`** flag 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 pack the characters as BCD (see Section 3) digits: *x = BCD(PIN)*
   c) If the PIN is an **`ascii-numeric`** PIN, verify that each character is a digit in the current codepage and –if needed– convert the characters to ascii[15] digits: *x = ASCII(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).*

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

### 7.9 The PKCS #15 Information File, EF(TokenInfo)

This file, only relevant to ISO/IEC 7816-4 compliant IC cards, contains general information about the PKCS #15 application and the token it resides on. It's data structure is defined as follows:

```
PKCS15TokenInfo ::= SEQUENCE {
  version        INTEGER {v1(0)} (v1,...),
  serialNumber   OCTET STRING,
  manufacturerID PKCS15Label OPTIONAL,
  label          [0] PKCS15Label OPTIONAL,
  tokenflags     PKCS15TokenFlags,
```

[15] See ANSI X3.4.

Copyright © 1999 RSA Laboratories.

```
  seInfo         SEQUENCE OF PKCS15SecurityEnvironmentInfo OPTIONAL,
  recordInfo     PKCS15RecordInfo OPTIONAL,
  supportedAlgorithms [1] SEQUENCE OF PKCS15AlgorithmInfo OPTIONAL,
  ... -- For future extensions
  } (CONSTRAINED BY { -- Each PKCS15AlgorithmInfo.reference value must
be unique --})

PKCS15TokenFlags ::= BIT STRING {
        readonly       (0),
        loginRequired  (1),
        prnGeneration  (2),
        eidCompliant   (3)
        }

PKCS15SecurityEnvironmentInfo ::= SEQUENCE {
        se    INTEGER (0..127),
        owner OBJECT IDENTIFIER,
        ... -- For future extensions
        }

PKCS15RecordInfo ::= 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
     }

PKCS15AlgorithmInfo ::= SEQUENCE {
     reference           PKCS15Reference,
     algorithm           PKCS15-ALGORITHM.&id({PKCS15AlgorithmSet}),
     parameters          PKCS15-ALGORITHM.&Parameters(
                             {PKCS15AlgorithmSet}{@algorithm}),
     supportedOperations PKCS15-ALGORITHM.&Operations(
                             {PKCS15AlgorithmSet}{@algorithm})
     }
```

The interpretation of these fields should be as follows:

- **`PKCS15TokenInfo.version:`** This field contains the number of the particular version of this specification the token application is based upon. For this version of this document, the value of **`version`** shall be 0 (**`v1`**).

- **`PKCS15TokenInfo.serialNumber:`** This field shall contain the token's unique serial number, for IC card issued in accordance with ISO/IEC 7812-1 and coded in accordance with ISO/IEC 8583. An example of this field can be found in Appendix C.

- **`PKCS15TokenInfo.manufacturerID`**: This optional field shall, when present, contain identifying information about the application issuer, UTF8-encoded.

Copyright © 1999 RSA Laboratories.

- **`PKCS15TokenInfo.label`**: This optional field shall, when present, contain identifying information about the application.

- **`PKCS15TokenInfo.tokenflags`**: This field contains information about the token *per se*. Flags include: If the whole PKCS #15 application is read-only, if login (i.e. authentication) is required before accessing any data, if the token supports pseudo-random number generation and if the token conforms to the electronic identification profile of this specification, specified in Annex B.

- **`PKCS15TokenInfo.seInfo`**: This optional field is intended to convey information about pre-set security environments on the card, and the owner of these environments. The definition of these environments is currently out of scope for this document.

- **`PKCS15TokenInfo.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 **`PKCS15RecordInfo`** 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 **`PKCS15RecordInfo`** can either be absent or set to zero.

- **`PKCS15TokenInfo.supportedAlgorithms`**: The intent of this optional field is to indicate cryptographic algorithms, associated parameters and operations supported by the card. The **`reference`** field of **`PKCS15AlgorithmInfo`** 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 *token* can perform with a particular algorithm.

# 8 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 PKCS15Framework.

```
PKCS15Framework {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
pkcs-15(15) modules(1) pkcs15-framework(1)}

-- This module has been checked for conformance with the ASN.1 standard
-- by the OSS ASN.1 Tools

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS All --
-- All types and values defined in this module is exported for use in other
```

Copyright © 1999 RSA Laboratories.

```
-- ASN.1 modules.

IMPORTS

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

Name
        FROM InformationFramework informationFramework

Certificate, AttributeCertificate, CertificateSerialNumber
        FROM AuthenticationFramework authenticationFramework

GeneralNames
        FROM CertificateExtensions certificateExtensions

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 ::= 32
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

-- 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

-- Basic types

PKCS15Identifier ::= OCTET STRING (SIZE (0..pkcs15-ub-identifier))

PKCS15Reference ::= INTEGER (0..pkcs15-ub-reference)

PKCS15Label ::= UTF8String (SIZE(0..pkcs15-ub-label))

PKCS15ReferencedValue ::= CHOICE {
        path PKCS15Path,
        url  PrintableString
        }

PKCS15Path ::= 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})
```

Copyright © 1999 RSA Laboratories.

```
PKCS15ObjectValue { Type } ::= CHOICE {
        indirect PKCS15ReferencedValue,
        direct   [0] Type
        } (CONSTRAINED BY {-- if indirection is being used,
           -- then it is expected that the reference points
           -- either to an object of type -- Type -- or (key
           -- case) to a card-specific key file --})

PKCS15PathOrObjects {ObjectType} ::= CHOICE {
        path      PKCS15Path,
        objects   [0] SEQUENCE OF ObjectType
        }

-- Attribute types

PKCS15CommonObjectAttributes ::= SEQUENCE {
        label   PKCS15Label OPTIONAL,
        flags   PKCS15CommonObjectFlags OPTIONAL,
        authId  PKCS15Identifier OPTIONAL,
        ... -- For future extensions
        } (CONSTRAINED BY {-- authId must be present in the IC card
        -- case if flags.private is set. It must equal an
        -- authID in one AuthRecord in the AODF -- })

PKCS15CommonObjectFlags ::= BIT STRING {
        private(0),
        modifiable (1)
        }

PKCS15CommonKeyAttributes ::= SEQUENCE {
        iD            PKCS15Identifier,
        usage         PKCS15KeyUsageFlags,
        native        BOOLEAN DEFAULT TRUE,
        accessFlags   PKCS15KeyAccessFlags OPTIONAL,
        keyReference  PKCS15Reference OPTIONAL,
        startDate     GeneralizedTime OPTIONAL,
        endDate       [0] GeneralizedTime OPTIONAL,
        ... -- For future extensions
        }

PKCS15KeyUsageFlags ::= BIT STRING {
        encrypt         (0),
        decrypt         (1),
        sign            (2),
        signRecover     (3),
        wrap            (4),
        unwrap          (5),
        verify          (6),
        verifyRecover   (7),
        derive          (8),
        nonRepudiation  (9)
        }

PKCS15KeyAccessFlags ::= BIT STRING {
        sensitive       (0),
        extractable     (1),
        alwaysSensitive (2),
        neverExtractable(3),
        local           (4)
        }

PKCS15CommonPrivateKeyAttributes ::= SEQUENCE {
        subjectName    Name OPTIONAL,
        keyIdentifiers [0] SEQUENCE OF PKCS15KeyIdentifier OPTIONAL,
        ... -- For future extensions
        }

PKCS15KeyIdentifier ::= SEQUENCE {
        idType  PKCS15KEY-IDENTIFIER.&id    ({PKCS15KeyIdentifiers}),
```

Copyright © 1999 RSA Laboratories.

```
        idValue PKCS15KEY-IDENTIFIER.&Value ({PKCS15KeyIdentifiers}{@idType})
        }

PKCS15KeyIdentifiers PKCS15KEY-IDENTIFIER ::= {
        pkcs15IssuerAndSerialNumber|
        pkcs15SubjectKeyIdentifier|
        pkcs15IssuerAndSerialNumberHash|
        pkcs15SubjectKeyHash|
        pkcs15IssuerKeyHash,
        ... -- For future extensions
        }

PKCS15KEY-IDENTIFIER ::= CLASS {
        &id  INTEGER UNIQUE,
        &Value
} WITH SYNTAX {
        SYNTAX &Value IDENTIFIED BY &id
}

pkcs15IssuerAndSerialNumber PKCS15KEY-IDENTIFIER::=
        {SYNTAX PKCS15-OPAQUE.&Type IDENTIFIED BY 1}
        -- As defined in RFC [CMS]
pkcs15SubjectKeyIdentifier PKCS15KEY-IDENTIFIER ::=
        {SYNTAX OCTET STRING IDENTIFIED BY 2}
        -- From x509v3 certificate extension
pkcs15IssuerAndSerialNumberHash PKCS15KEY-IDENTIFIER ::=
        {SYNTAX OCTET STRING IDENTIFIED BY 3}
        -- Assumes SHA-1 hash of DER encoding of IssuerAndSerialNumber
pkcs15SubjectKeyHash PKCS15KEY-IDENTIFIER ::=
        {SYNTAX OCTET STRING IDENTIFIED BY 4}
        -- Hash method defined in Section 7.
pkcs15IssuerKeyHash PKCS15KEY-IDENTIFIER ::=
        {SYNTAX OCTET STRING IDENTIFIED BY 5}
        -- Hash method defined in Section 7.

PKCS15CommonPublicKeyAttributes ::= SEQUENCE {
        subjectName Name OPTIONAL,
        ... -- For future extensions
        }

PKCS15CommonSecretKeyAttributes ::= SEQUENCE {
        keyLen      INTEGER OPTIONAL, -- keylength (in bits)
        ... -- For future extensions
        }

PKCS15KeyInfo {ParameterType, OperationsType} ::= CHOICE {
        reference PKCS15Reference,
        paramsAndOps SEQUENCE {
                     parameters          ParameterType,
                     supportedOperations OperationsType OPTIONAL}
        }

PKCS15CommonCertificateAttributes ::= SEQUENCE {
        iD                  PKCS15Identifier,
        authority           BOOLEAN DEFAULT FALSE,
        requestId           PKCS15KeyIdentifier OPTIONAL,
        thumbprint          [0] PKCS15OOBCertHash OPTIONAL,
        ... -- For future extensions
        }

PKCS15CommonDataObjectAttributes ::= SEQUENCE {
        applicationName PKCS15Label OPTIONAL,
        applicationOID  OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        } (WITH COMPONENTS {..., applicationName PRESENT}|
           WITH COMPONENTS {..., applicationOID  PRESENT})

PKCS15CommonAuthenticationObjectAttributes ::= SEQUENCE {
        authId          PKCS15Identifier,
```

Copyright © 1999 RSA Laboratories.

```
        ... -- For future extensions
        }

-- PKCS15 Objects

PKCS15Object {ClassAttributes, SubClassAttributes, TypeAttributes} ::=
    SEQUENCE {
        commonObjectAttributes PKCS15CommonObjectAttributes,
        classAttributes        ClassAttributes,
        subClassAttributes     [0] SubClassAttributes OPTIONAL,
        typeAttributes         [1] TypeAttributes
        }

PKCS15Objects ::= CHOICE {
        privateKeys         [0] PKCS15PrivateKeys,
        publicKeys          [1] PKCS15PublicKeys,
        trustedPublicKeys   [2] PKCS15PublicKeys,
        secretKeys          [3] PKCS15SecretKeys,
        certificates        [4] PKCS15Certificates,
        trustedCertificates [5] PKCS15Certificates,
        usefulCertificates  [6] PKCS15Certificates,
        dataObjects         [7] PKCS15DataObjects,
        authObjects         [8] PKCS15AuthObjects,
        ... -- For future extensions
        }

PKCS15PrivateKeys  ::= PKCS15PathOrObjects {PKCS15PrivateKey}
PKCS15SecretKeys   ::= PKCS15PathOrObjects {PKCS15SecretKey}
PKCS15PublicKeys   ::= PKCS15PathOrObjects {PKCS15PublicKey}
PKCS15Certificates ::= PKCS15PathOrObjects {PKCS15Certificate}
PKCS15DataObjects  ::= PKCS15PathOrObjects {PKCS15Data}
PKCS15AuthObjects  ::= PKCS15PathOrObjects {PKCS15Authentication}

PKCS15PrivateKey ::= CHOICE {
        privateRSAKey     PKCS15PrivateKeyObject {
                          PKCS15PrivateRSAKeyAttributes},
        privateECKey  [0] PKCS15PrivateKeyObject {
                          PKCS15PrivateECKeyAttributes},
        privateDHKey  [1] PKCS15PrivateKeyObject {
                          PKCS15PrivateDHKeyAttributes},
        privateDSAKey [2] PKCS15PrivateKeyObject {
                          PKCS15PrivateDSAKeyAttributes},
        privateKEAKey [3] PKCS15PrivateKeyObject {
                          PKCS15PrivateKEAKeyAttributes},
        ... -- For future extensions
        }

PKCS15PrivateKeyObject {KeyAttributes} ::= PKCS15Object {
                                 PKCS15CommonKeyAttributes,
                                 PKCS15CommonPrivateKeyAttributes,
                                 KeyAttributes}

PKCS15PrivateRSAKeyAttributes ::= SEQUENCE {
        value         PKCS15ObjectValue {PKCS15RSAPrivateKey},
        modulusLength INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo       PKCS15KeyInfo {PKCS15RSAParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
 ... -- For future extensions
        }

PKCS15RSAPrivateKey ::= 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
```

Copyright © 1999 RSA Laboratories.

```
        } (CONSTRAINED BY
             {-- must be possible to reconstruct modulus and
              -- privateExponent from selected fields --})

PKCS15PrivateECKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15ECPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15ECParameters,
                              PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15ECPrivateKey ::= INTEGER

PKCS15PrivateDHKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DHPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15DHParameters,
                              PKCS15PublicKeyOperations} OPTIONAL,
         ... -- For future extensions
        }

PKCS15DHPrivateKey ::= INTEGER -- Diffie-Hellman exponent

PKCS15PrivateDSAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DSAPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15DSAParameters,
                              PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DSAPrivateKey ::= INTEGER

PKCS15PrivateKEAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15KEAPrivateKey},
        keyInfo PKCS15KeyInfo {PKCS15KEAParameters,
                              PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15KEAPrivateKey ::= INTEGER

PKCS15PublicKey ::= CHOICE {
        publicRSAKey     PKCS15PublicKeyObject {
                          PKCS15PublicRSAKeyAttributes},
        publicECKey  [0] PKCS15PublicKeyObject {
                          PKCS15PublicECKeyAttributes},
        publicDHKey  [1] PKCS15PublicKeyObject {
                          PKCS15PublicDHKeyAttributes},
        publicDSAKey [2] PKCS15PublicKeyObject {
                          PKCS15PublicDSAKeyAttributes},
        publicKEAKey [3] PKCS15PublicKeyObject {
                          PKCS15PublicKEAKeyAttributes},
        ... -- For future extensions
        }

PKCS15PublicKeyObject {KeyAttributes} ::= PKCS15Object {
                                PKCS15CommonKeyAttributes,
                                PKCS15CommonPublicKeyAttributes,
                                KeyAttributes}

PKCS15PublicRSAKeyAttributes ::= SEQUENCE {
        value          PKCS15ObjectValue {PKCS15RSAPublicKey},
        modulusLength  INTEGER, -- modulus length in bits, e.g. 1024
        keyInfo        PKCS15KeyInfo {PKCS15RSAParameters,
                            PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15RSAPublicKey ::= SEQUENCE {
        modulus         INTEGER, -- n
```

Copyright © 1999 RSA Laboratories.

```
        publicExponent  INTEGER  -- e
        }

PKCS15PublicECKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15ECPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15ECParameters,
                                   PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15ECPublicKey ::= PKCS15ECPoint

PKCS15PublicDHKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DHPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15DHParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DHPublicKey ::= PKCS15DiffieHellmanPublicNumber

PKCS15PublicDSAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15DSAPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15DSAParameters,
                PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15DSAPublicKey ::= INTEGER

PKCS15PublicKEAKeyAttributes ::= SEQUENCE {
        value   PKCS15ObjectValue {PKCS15KEAPublicKey},
        keyInfo PKCS15KeyInfo {PKCS15KEAParameters,
                               PKCS15PublicKeyOperations} OPTIONAL,
        ... -- For future extensions
        }

PKCS15KEAPublicKey ::= INTEGER

PKCS15SecretKey ::= CHOICE {
        genericSecretKey     PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc2key           [0] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc4key           [1] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        desKey           [2] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        des2Key          [3] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        des3Key          [4] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        castKey          [5] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        cast3Key         [6] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        cast128Key       [7] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc5Key           [8] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        ideaKey          [9] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        skipjackKey     [10] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        batonKey        [11] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        juniperKey      [12] PKCS15SecretKeyObject
                             {PKCS15GenericSecretKeyAttributes},
        rc6Key          [13] PKCS15SecretKeyObject
```

Copyright © 1999 RSA Laboratories.

```
                             {PKCS15GenericSecretKeyAttributes},
        otherKey        [14] PKCS15OtherKey,
        ... -- For future extensions
        }

PKCS15SecretKeyObject {KeyAttributes} ::= PKCS15Object {
        PKCS15CommonKeyAttributes,
        PKCS15CommonSecretKeyAttributes,
        KeyAttributes}

PKCS15GenericSecretKeyAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { OCTET STRING },
        ... -- For future extensions
        }

PKCS15OtherKey ::= SEQUENCE {
        keyType OBJECT IDENTIFIER,
        keyAttr PKCS15SecretKeyObject {PKCS15GenericSecretKeyAttributes}
        }

PKCS15Certificate ::= CHOICE {
        x509Certificate              PKCS15CertificateObject {
                      PKCS15X509CertificateAttributes},
        x509AttributeCertificate [0] PKCS15CertificateObject {
                      PKCS15X509AttributeCertificateAttributes},
        spkiCertificate          [1] PKCS15CertificateObject {
                      PKCS15SPKICertificateAttributes},
        pgpCertificate           [2] PKCS15CertificateObject {
                      PKCS15PGPCertificateAttributes},
        wtlsCertificate          [3] PKCS15CertificateObject {
                      PKCS15WTLSCertificateAttributes},
        x9-68Certificate         [4] PKCS15CertificateObject {
                      PKCS15X9-68CertificateAttributes},
        ... -- For future extensions
        }

PKCS15CertificateObject {CertAttributes} ::= PKCS15Object {
                              PKCS15CommonCertificateAttributes,
                              NULL,
                              CertAttributes}

PKCS15X509CertificateAttributes ::= SEQUENCE {
        value        PKCS15ObjectValue { PKCS15X509Certificate },
        subject      [0] Name OPTIONAL,
        issuer       [1] Name OPTIONAL,
        serialNumber CertificateSerialNumber OPTIONAL,
        ... -- For future extensions
        }

PKCS15X509AttributeCertificateAttributes ::= SEQUENCE {
        value         PKCS15ObjectValue { PKCS15AttributeCertificate },
        issuer        GeneralNames OPTIONAL,
        serialNumber  CertificateSerialNumber OPTIONAL,
        attrTypes     [0] SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
        ... -- For future extensions
        }


PKCS15SPKICertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

PKCS15PGPCertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

PKCS15WTLSCertificateAttributes ::= SEQUENCE {
```

Copyright © 1999 RSA Laboratories.

```
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

PKCS15X9-68CertificateAttributes ::= SEQUENCE {
        value PKCS15ObjectValue { PKCS15-OPAQUE.&Type },
        ... -- For future extensions
        }

PKCS15Data ::= CHOICE {
        opaqueDO            PKCS15DataObject {PKCS15Opaque},
        externalIDO     [0] PKCS15DataObject {PKCS15ExternalIDO},
        oidDO           [1] PKCS15DataObject {PKCS15OidDO},
        ... -- For future extensions
         }

PKCS15DataObject {DataObjectAttributes} ::= PKCS15Object {
                                 PKCS15CommonDataObjectAttributes,
                                 NULL,
                                 DataObjectAttributes}

PKCS15Opaque ::= PKCS15ObjectValue {PKCS15-OPAQUE.&Type}

PKCS15ExternalIDO ::= PKCS15ObjectValue {PKCS15-OPAQUE.&Type}
        (CONSTRAINED BY {-- All data objects must be defined in accordance
                         -- with ISO/IEC 7816-6 --})

PKCS15OidDO ::= SEQUENCE {
        id    OBJECT IDENTIFIER,
        value PKCS15ObjectValue {PKCS15-OPAQUE.&Type}
        }

PKCS15Authentication ::= CHOICE {
        pin PKCS15AuthenticationObject {PKCS15PinAttributes},
        ... -- For future extensions, e.g. biometric authentication
            -- objects
        }

PKCS15AuthenticationObject {AuthObjectAttributes} ::= PKCS15Object {
                             PKCS15CommonAuthenticationObjectAttributes,
                             NULL,
                             AuthObjectAttributes}

PKCS15PinAttributes ::= SEQUENCE {
        pinFlags      PKCS15PinFlags,
        pinType       PKCS15PinType,
        minLength     INTEGER
                       (pkcs15-lb-minPinLength..pkcs15-ub-minPinLength),
        storedLength  INTEGER
                    (pkcs15-lb-minPinLength..pkcs15-ub-storedPinLength),
        maxLength     INTEGER OPTIONAL,
        pinReference  [0] PKCS15Reference OPTIONAL,
        padChar       OCTET STRING (SIZE(1)) OPTIONAL,
        lastPinChange GeneralizedTime OPTIONAL,
        path          PKCS15Path OPTIONAL,
        ... -- For future extensions
        }

PKCS15PinFlags ::= 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)
        } (CONSTRAINED BY { -- 'unblockingPin' and 'soPIN' cannot both
```

Copyright © 1999 RSA Laboratories.

```
          -- be set --})

PKCS15PinType ::= ENUMERATED {bcd, ascii-numeric, utf8, ...
                  -- bcd = one nibble contains one digit
                  -- ascii-numeric = one byte contains one ASCII digit
                  -- utf8 = password is stored in UTF8 encoding
                  }

PKCS15TokenInfo ::= SEQUENCE {
  version             INTEGER {v1(0)} (v1,...),
  serialNumber        OCTET STRING,
  manufacturerID      PKCS15Label OPTIONAL,
  label               [0] PKCS15Label OPTIONAL,
  tokenflags          PKCS15TokenFlags,
  seInfo              SEQUENCE OF PKCS15SecurityEnvironmentInfo OPTIONAL,
  recordInfo          [1] PKCS15RecordInfo OPTIONAL,
  supportedAlgorithms [2] SEQUENCE OF PKCS15AlgorithmInfo OPTIONAL,
  ... -- For future extensions
  } (CONSTRAINED BY { -- Each PKCS15AlgorithmInfo.reference value
     -- must be unique --})

PKCS15TokenFlags ::= BIT STRING {
        readonly       (0),
        loginRequired  (1),
        prnGeneration  (2),
        eidCompliant   (3)
        }

PKCS15SecurityEnvironmentInfo ::= SEQUENCE {
        se    INTEGER (0..127),
        owner OBJECT IDENTIFIER,
        ... -- For future extensions
        }

PKCS15RecordInfo ::= 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
        }

PKCS15AlgorithmInfo ::= SEQUENCE {
    reference           PKCS15Reference,
    algorithm           PKCS15-ALGORITHM.&id({PKCS15AlgorithmSet}),
    parameters          PKCS15-ALGORITHM.&Parameters({PKCS15AlgorithmSet}{@algorithm}),
    supportedOperations PKCS15-ALGORITHM.&Operations({PKCS15AlgorithmSet}{@algorithm})
    }

PKCS15-ALGORITHM ::= CLASS {
        &id INTEGER UNIQUE,
        &Parameters,
        &Operations PKCS15Operations
} WITH SYNTAX {
  PARAMETERS &Parameters OPERATIONS &Operations ID &id}

pkcs15-alg-null      PKCS15-ALGORITHM ::= {
        PARAMETERS NULL OPERATIONS {{generate-key}} ID -1}
pkcs15-alg-rsa       PKCS15-ALGORITHM ::= {
        PARAMETERS PKCS15RSAParameters OPERATIONS
        {PKCS15PublicKeyOperations} ID 0}
pkcs15-alg-dsa        PKCS15-ALGORITHM ::= {
        PARAMETERS PKCS15DSAParameters  OPERATIONS
        {PKCS15PublicKeyOperations} ID 1}
pkcs15-alg-dh        PKCS15-ALGORITHM ::= {
        PARAMETERS PKCS15DHParameters  OPERATIONS
        {PKCS15PublicKeyOperations} ID 2}
```

Copyright © 1999 RSA Laboratories.

```
pkcs15-alg-ec        PKCS15-ALGORITHM ::= {
        PARAMETERS PKCS15ECParameters  OPERATIONS
        {PKCS15PublicKeyOperations} ID 3}
pkcs15-alg-kea        PKCS15-ALGORITHM ::= {
        PARAMETERS PKCS15KEAParameters  OPERATIONS
        {PKCS15PublicKeyOperations} ID 5}

PKCS15AlgorithmSet PKCS15-ALGORITHM ::= {
        pkcs15-alg-null   |
        pkcs15-alg-rsa    |
        pkcs15-alg-ec     |
        pkcs15-alg-dh     |
        pkcs15-alg-dsa    |
        pkcs15-alg-kea,
        ... -- For future extensions
        }

PKCS15PublicKeyOperations ::= PKCS15Operations

PKCS15Operations ::= 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
        }

PKCS15OOBCertHash   ::= OOBCertHash -- See IETF RFC 2510

PKCS15RSAParameters ::= NULL

PKCS15ECParameters  ::= Parameters -- See ANSI X9.62

PKCS15DHParameters  ::= DomainParameters -- See ANSI X9.42

PKCS15DSAParameters ::= DomainParameters -- See ANSI X9.42

PKCS15KEAParameters ::= DomainParameters -- See ANSI X9.42

PKCS15ECPoint       ::= ECPoint -- See ANSI X9.62

PKCS15DiffieHellmanPublicNumber ::= DiffieHellmanPublicNumber -- See ANSI X9.42

PKCS15X509Certificate ::= Certificate -- See X.509

PKCS15AttributeCertificate ::= AttributeCertificate -- See X.509

PKCS15-OPAQUE ::= TYPE-IDENTIFIER

-- Misc

PKCS15DIRRecord ::= [APPLICATION 1] SEQUENCE {
        aid   [APPLICATION 15] OCTET STRING,
        label [APPLICATION 16] UTF8String OPTIONAL,
        path  [APPLICATION 17] OCTET STRING,
        ddo   [APPLICATION 19] PKCS15DDO OPTIONAL
        }

PKCS15DDO ::= SEQUENCE {
        oid           OBJECT IDENTIFIER,
        odfPath       PKCS15Path OPTIONAL,
        tokenInfoPath [0] PKCS15Path OPTIONAL,
        unusedPath    [1] PKCS15Path OPTIONAL,
        ... -- For future extensions
        }
```

Copyright © 1999 RSA Laboratories.

```
PKCS15UnusedSpace ::= SEQUENCE {
        path   PKCS15Path (WITH COMPONENTS
                   {..., index PRESENT, length PRESENT}),
        authId PKCS15Identifier OPTIONAL
        }

END
```

# 9 Intellectual property considerations

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

RC2 and RC5 are trademarks of RSA Data Security.

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

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

Copyright © 1999 RSA Laboratories.

# Appendix A: File Access Conditions (Informative)

## A.1 Scope

This appendix is only applicable to IC card implementations.

## A.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).

## A.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 **`PKCS15Operations`** 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.

| **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 |

Copyright © 1999 RSA Laboratories.

| | | |
|---|---|---|
| | | 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 |

**Table 3:** File access methods

Note that 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 IC card types.

| **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`' |

**Table 4:** Possible access conditions

The following access conditions are recommended for files related to the PKCS #15 application[16]:

| **File** | **DF** | **R/O card** | **R/W card** |
|---|---|---|---|
| MF | X | `Create: SYS`<br>`Delete: NEV` | `Create: SYS`<br>`Delete: SYS` |
| DIR | | `Read: ALW`<br>`Update: SYS`<br>`Append: SYS` | Same as for R/O card. |
| PIN files | | `Read: NEV`<br>`Update: NEV`<br>`Append: NEV` | `Read: NEV`<br>`Update: CHV`<br>`Append: NEV` |

[16] 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').

Copyright © 1999 RSA Laboratories.

| | | | |
|---|---|---|---|
| PKCS15 | X | `Create: SYS`<br>`Delete: NEV` | `Create: CHV \| SYS`<br>`Delete: CHV \| SYS` |
| TokenInfo | | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` | Same as for R/O card. |
| ODF | | `Read: ALW`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW`<br>`Update: SYS \| NEV`<br>`Append: SYS \| NEV` |
| AODFs | | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` | `Read: ALW`<br>`Update: CHV \| SYS \| NEV`<br>`Append: CHV \| SYS \| NEV` |
| PrKDFs, PuKDFs, SKDFs, CDFs and DODFs | | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW \| CHV`<br>`Update: CHV`<br>`Append: CHV` |
| Trusted CDFs<br>Trusted PuKDFs | | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW \| CHV`<br>`Update: SYS \| NEV`<br>`Append: SYS \| NEV` |
| Key files[17] | | `Read: NEV`<br>`Update: NEV`<br>`Append: NEV`<br>`Crypt: CHV` | `Read: NEV`<br>`Update: CHV \| SYS \| NEV`<br>`Append: CHV \| SYS \| NEV`<br>`Crypt: CHV` |
| Other EFs | | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW \| CHV`<br>`Update: CHV`<br>`Append: CHV` |

**Table 5:** Recommended file access conditions

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) are allowed to 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.

[17] Files containing private or secret keys and the token supports crypto-related commands for these files

Copyright © 1999 RSA Laboratories.

# Appendix B: An Electronic Identification Profile of PKCS #15 (Normative)

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 tokens and for host-side applications making use of EID tokens.

## B.1 PKCS #15 objects

- Private Keys: A PKCS #15 token 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 tokens 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 tokenholder 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. Tokens 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 token, 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 token, depending on the application issuer’s discretion. There is no requirement for host-side applications to handle these keys.

Copyright © 1999 RSA Laboratories.

- Certificates: For each private key at least one corresponding certificate should be stored in the token. The certificates must be of type **`PKCS15X509Certificate`**. If an application issuer stores CA certificates on a token 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 token issuer (or not modifiable at all). This implies usage of the **`trustedCertificates`** choice in the **`PKCS15Objects`** type. User certificates for which private keys exist on the token should be profiled in accordance with IETF RFC 2459. Host-side applications are required to recognize and be able to use the **`PKCS15X509Certificate`** type.

- Data objects: No requirements. Objects of this type may or may not be present on the token, 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[18], 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 token.

  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.

## B.2 Other files

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

---

[18] Future versions of this profile may also include support for biometric authentication methods.

Copyright © 1999 RSA Laboratories.

### B.3 Constraints on ASN.1 types

Unless otherwise mentioned, conforming applications are required to recognize[19] and parse all **`OPTIONAL`** fields. The following constraints applies for tokens and applications claiming conformance to this EID profile:

- **`PKCS15CommonObjectAttributes.label`** must be present for all certificate objects.
- **`PKCS15CommonKeyAttributes.startDate`** must not be present.
- **`PKCS15CommonKeyAttributes.endDate`** must not be present.
- **`PKCS15CommonPrivateKeyAttributes.subjectName`** must not be present.
- **`PKCS15CommonPrivateKeyAttributes.keyIdentifiers`** must be recognized by host-side applications but need not be interpreted.
- **`PKCS15CommonCertificateAttributes.requestID`** must be recognized by host-side applications but need not be interpreted.
- **`PKCS15X509CertificateAttributes.subject`** must not be present.
- **`PKCS15X509CertificateAttributes.issuer`** must not be present.
- **`PKCS15X509CertificateAttributes.serialNumber`** must not be present.
- **`PKCS15PinAttributes.lastPinChange`** must be recognized by host-side applications but need not be interpreted.

### B.4 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 that it is possible for **`PKCS15Path`** 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.

---

[19] 'Recognize' means "being able to proceed also when the field is present, but not necessarily being able to interpret the field's contents."

Copyright © 1999 RSA Laboratories.

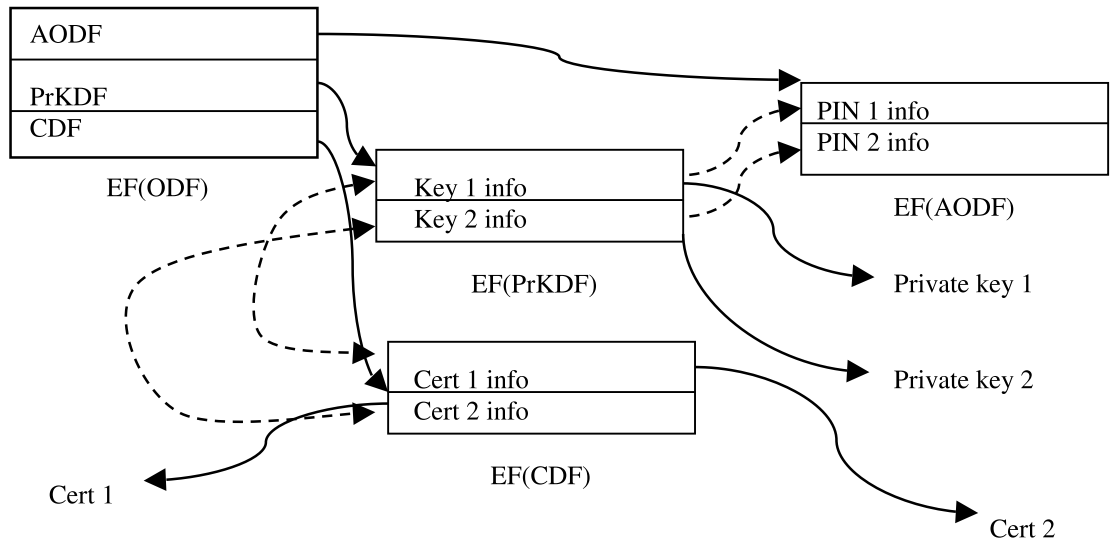


**Figure 15:** IC card file relationships in DF(PKCS15). Dashed arrows indicate cross-references.

## B.5 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).

| **File** | **Access Conditions, R-O token** | **Access Conditions. R-W token** |
|---|---|---|
| MF | `Create: SYS`<br>`Delete: NEV` | `Create: SYS`<br>`Delete: SYS` |
| EF(DIR) | `Read: ALW`<br>`Update: SYS`<br>`Append: SYS` | `Read: ALW`<br>`Update: SYS`<br>`Append: SYS` |
| PIN files | `Read: NEV`<br>`Update: NEV`<br>`Append: NEV` | `Read: NEV`<br>`Update: CHV`<br>`Append: NEV` |
| DF(PKCS15) | `Create: SYS`<br>`Delete: NEV` | `Create: CHV \| SYS`<br>`Delete: SYS` |
| EF(TokenInfo) | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` |
| EF(ODF) | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` | `Read: ALW`<br>`Update: SYS`<br>`Append: SYS` |
| AODFs | `Read: ALW`<br>`Update: NEV`<br>`Append: NEV` | `Read: ALW`<br>`Update: NEV`<br>`Append: CHV \| SYS` |

Copyright © 1999 RSA Laboratories.

| | | |
|---|---|---|
| PrKDFs, PuKDFs, SKDFs, CDFs and DODFs | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW \| CHV`<br>`Update: CHV \| SYS \| NEV`<br>`Append: SYS \| CHV` |
| Trusted CDFs | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV` | `Read: ALW \| CHV`<br>`Update: SYS \| NEV`<br>`Append: SYS \| NEV` |
| Key files (see footnote for `Table 5`) | `Read: NEV`<br>`Update: NEV`<br>`Append: NEV`<br>`Crypt: CHV` | `Read: NEV`<br>`Update: CHV \| SYS \| NEV`<br>`Append: CHV \| SYS \| NEV`<br>`Crypt: CHV` |
| Other EFs in the PKCS15 directory | `Read: ALW \| CHV`<br>`Update: NEV`<br>`Append: SYS \| NEV`<br>`Crypt: CHV (when applic.)` | `Read: ALW \| CHV`<br>`Update: CHV \| SYS \| NEV`<br>`Append: CHV \| SYS \| NEV`<br>`Crypt: CHV (when applic.)` |

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

**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.

Copyright © 1999 RSA Laboratories.

# Appendix C: Examples (Informative)

Note that, similar to Section 6.1, when this section talks about or describes "contents" of IC card files, this is just a shorthand notation for "the contents of files as it appears to someone using standard IC card commands in accordance with ISO/IEC 7816-4 to access them".

All examples are shown both in the formal value notation defined in ISO/IEC 8824-1 and in DER encoding.

## C.1 Example of EF(DIR)

Example contents for a PKCS #15 application template on an IC card using indirect application selection. A non-standard file path for EF(UnusedSpace) is defined, `/3F00/5015/4320`.

Value notation:

```
{
    aid   'A000000063504B43532D3135'H,
    label "RSA DSI", -- UTF8 Encoded
    path  '3F005015'H,
    ddo   {
        oid {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
                pkcs-15(15) applications(4) eid(1)},
        -- Example OID, not for actual use
        unusedPath {
            path '3F0050154320'H
        }
    }
}
```

DER encoding:

```
61354F0C A0000000 63504B43 532D3135 50075253 41204453 4951043F 00501573
16060A2A 864886F7 0D010F04 01A10804 063F0050 154320
```

## C.2 Example of a whole PKCS15 application

The IC card in this example has on-chip support for RSA and DES-EDE-CBC algorithm computation in addition to pseudo-random number generation. It is assumed that this information can be deduced from the card's ATR string. As a consequence of this, the TokenInfo file contains no **`supportedAlgorithms`** field. The PKCS15 application is profiled for use in an electronic identification environment, in compliance with Appendix B, and has two RSA key pairs and two certificates. One private key is for digital signature purposes only and is protected with a separate authentication object (a PIN). There is also a private data object belonging to an application named 'APP'. The total overhead for

Copyright © 1999 RSA Laboratories.

storing the PKCS #15 relevant information is in this case 374 bytes, but without the data object belonging to the 'APP' application it would have been 333 bytes.

### C.2.1 EF(TokenInfo)

Value notation:

```
{
    version      v1,
    serialNumber '159752222515401240'H,
    manufacturerID "Acme, Inc.",
    tokenflags {prnGeneration, eidCompliant}
}
```

DER encoding:

```
301E0201 00040915 97522225 15401240 0C0A4163 6D652C20 496E632E 03020430
```

The total size of the data is 32 bytes.

### C.2.2 EF(ODF)

Value notation:

```
{
    privateKeys  : path : {
        path '4401'H -- Reference by file identifier
    },
    certificates : path : {
        path '4402'H -- Reference by file identifier
    },
    dataObjects  : path : {
        path '4403'H -- Reference by file identifier
    },
    authObjects  : path : {
        path '4404'H -- Reference by file identifier
    }
}
```

DER encoding (as specified, outermost **`SEQUENCE OF`** omitted):

```
A0063004 04024401 A4063004 04024402 A7063004 04024403 A8063004 04024404
```

As can be seen, the ODF simply consists of four records, and the total size of the data is 32 bytes.

### C.2.3 EF(PrKDF)

Value notation:

Copyright © 1999 RSA Laboratories.

```
{
    privateRSAKey : {
        commonObjectAttributes {
            label  "KEY1",
            flags  {private},
            authId '01'H
        },
        classAttributes {
            iD     '45'H,
            usage  {decrypt, sign, unwrap},
            -- By default 'native' RSA key
        },
        subClassAttributes {
            keyIdentifiers {
                {
                    idType  4, -- Subject key hash
                    idValue OCTET STRING : '4321567890ABCDEF'H
                        -- Faked value
                }
            }
        },
        typeAttributes {
            value indirect : path : {
                path '4B01'H -- Reference by file identifier
            },
            modulusLength 1024
        }
    },
    privateRSAKey : {
        commonObjectAttributes {
            label  "KEY2",
            flags  {private},
            authId '02'H
        },
        classAttributes {
            iD     '46'H,
            usage  {nonRepudiation, sign},
            -- By default 'native' RSA key
        },
        subClassAttributes {
            keyIdentifiers {
                {
                    idType  4, -- Subject key hash
                    idValue OCTET STRING : '1234567890ABCDEF'H
                        -- Faked value
                }
            }
        },
        typeAttributes {
            value indirect : path : {
                path '4B02'H -- Reference by file identifier
            },
            modulusLength 1024
        }
    }
}
```

Copyright © 1999 RSA Laboratories.

DER encoding (as specified, outermost **SEQUENCE OF** omitted):

```
303B300D 0C044B45 59310302 07800401 01300704 01450302 0264A013 3011A00F
300D0201 04040843 21567890 ABCDEFA1 0C300A30 0404024B 01020204 00303C30
0D0C044B 45593203 02078004 01023008 04014603 03062040 A0133011 A00F300D
02010404 08123456 7890ABCD EFA10C30 0A300404 024B0202 020400
```

The content of files 3F00/5015/4B01 and 3F00/5015/4B02 is completely card-specific. Operations possible to perform with keys in these files may either be deduced by looking at the contents of the TokenInfo file or by external knowledge of the card in question (ATR). The size of the data is 123 bytes (one record of 61 bytes and one record of 62 bytes).

### C.2.4 EF(CDF)

Value notation:

```
{
    x509Certificate : {
        commonObjectAttributes {
            label  "CERT1",
            flags  {}, -- Not private, read-only
        },
        classAttributes {
            iD     ’45’H
            -- By default not an authority
        },
        typeAttributes {
            value indirect : path : {
                path ’4331’H -- Reference by file identifier
            }
        }
    },
    x509Certificate : {
        commonObjectAttributes {
            label  "CERT2",
            flags  {}, -- Not private, read-only
        },
        classAttributes {
            iD     ’46’H
            -- By default not an authority
        },
        typeAttributes {
            value indirect : path : {
                path ’4332’H -- Reference by file identifier
            }
        }
    }
}
```

DER encoding (as specified, outermost **SEQUENCE OF** omitted):

Copyright © 1999 RSA Laboratories.

```
301B300A 0C054345 52543103 01003003 040145A1 08300630 04040243 31301B30
0A0C0543 45525432 03010030 03040146 A1083006 30040402 4332
```

Files 3F00/5015/4331 and 3F00/5015/4332 should contain DER-encoded certificate structures in accordance with ISO/IEC 9594-8. The size of the data is 58 bytes (two records of 29 bytes each).

### C.2.5 EF(AODF)

Value notation:

```
{
    pin : {
        commonObjectAttributes {
            label "PIN1",
            flags {private}
        },
        classAttributes {
            authId ’01’H -- Binds to KEY1
        },
        typeAttributes {
            pinFlags     {change-disabled, initialized, needs-padding},
            pinType      bcd,
            minLength    4,
            storedLength 8,
            padChar      ’FF’H
            -- path not given, implicitly PIN file in MF
        }
    },
    pin : {
        commonObjectAttributes {
            label "PIN2",
            flags {private}
        },
        classAttributes {
            authId ’02’H -- Binds to KEY1
        },
        typeAttributes {
            pinFlags     {change-disabled, initialized, needs-padding},
            pinType      bcd,
            minLength    4,
            storedLength 8,
            padChar      ’FF’H,
            path {
                path ’3F0050150100’H -- Reference by absolute path
            }
        }
    }
}
```

DER encoding (as specified, outermost **SEQUENCE OF** omitted):

Copyright © 1999 RSA Laboratories.

```
3025300A 0C045049 4E310302 07803003 040101A1 12301003 02022C0A 01000201
04020108 0401FF30 2F300A0C 0450494E 32030207 80300304 0102A11C 301A0302
022C0A01 00020104 02010804 01FF3008 04063F00 50150100
```

The content of files 3F00/5015/0100 and 3F00/0000 is card specific and not specified in PKCS #15. The total size of the data is 88 bytes (one record of length 39 bytes, the other of length 49 bytes).

### C.2.6 EF(DODF)

Value notation:

```
{
    opaqueDO : {
        commonObjectAttributes {
            label "OBJECT1",
            flags {private, modifiable},
            authId ’02’H -- Binds to PIN2
        },
        classAttributes {
            applicationName "APP"
        },
        typeAttributes indirect : path : {
            path ’4431’H, -- Reference by file identifier
            index 64,
            length 48
        }
    }
}
```

DER encoding (outermost **`SEQUENCE OF`** omitted, as specified):

```
30273010 0C074F42 4A454354 31030206 C0040102 30050C03 415050A1 0C300A04
02443102 01408001 30
```

The size of the data is 41 bytes (one record). The data entry in file 3F00/5015/4431 is to be found 64 bytes from the beginning of the file and is 48 bytes long.

Copyright © 1999 RSA Laboratories.

## 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 X9.45, 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
(781) 687-7000
(781) 687-7213 (fax)
pkcs-editor@rsa.com
http://www.rsa.com/rsalabs/pubs/PKCS

Copyright © 1999 RSA Laboratories.