TECHNICAL SPECIFICATION

ISO/IEC TS 23465-2

First edition
2023-02

Card and security devices for personal identification — Programming interface for security devices —

Part 2: API definition

Cartes et dispositifs de sécurité pour l'identification personnelle — L'interface du logiciel pour dispositifs de sécurité —

Partie 2: Definition de API

Reference number
ISO/IEC TS 23465-2:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 17, *Cards and security devices for personal identification*.

A list of all parts in the ISO/IEC 23465 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Integrated chip card (ICC) technologies and solutions are widely deployed around the world, but the system for identity tokens and credentials is quickly changing. In this context, the application protocol data unit (APDU) protocol outlined in the ISO/IEC 7816 series is becoming in some cases a hindrance to the integration of ICs in environments such as mobile phones, handheld devices, connected devices (e.g. M2M, IoT) or other applications using security devices.

In addition, several stakeholders are not familiar with, or not very fond of the APDU protocol because of its complexity. They would circumvent its constraints by requesting an abstraction layer hiding IC specifics such as data structures and complexity of the security policies.

A common way to reach this goal in the software development is the definition and application of application programming interface (API) functions to access the IC within the devices. Specific knowledge of ADPU protocols and details of the IC implementation is not necessary anymore. Also, the complexity and details of the implementation of the security model and the security policy can be shifted from the pure application development into the system design of the whole ID management.

However, even solutions based on those kinds of middleware are perceived as cumbersome in some systems. The market looks for a middleware memory footprint to be as low as possible and the acceptance, usage and maintenance of such a system can be simpler.

This document aims to overcome or mitigate those issues by proposing a new approach that preserves ICC functionality and allows a seamless ICC portability onto new systems.

The ISO/IEC 23465 series focuses on a solution by designing an API and a system with the following characteristics:

— It offers a set of API calls related to multi-sectorial ICC functionality, derived from the ISO/IEC 7816 series of other ICC related standards.

— It defines the sub-system to perform the conversion from the API function to the interface of the security device (e.g. APDU-interface), called “proxy”.

— It results in a description of solutions with no middleware or very little middleware memory footprint (i.e. simplified drivers).

— It defines simplified ICC capabilities, description of the discoverability (i.e. with significantly less complexity than ISO/IEC 24727) and provides examples of usages.

The present model is static and future revisions are expected to add live cycle functionality.

Card and security devices for personal identification — Programming interface for security devices —

Part 2: API definition

1 Scope

This document describes the following aspects of the programming interface between the client application dealing with the security device and the proxy, based on the framework outlined in ISO/IEC 23465-1:

— the generic API definition;

— state and security models for use cases;

— class and API definitions of functionality, defined in other standards, e.g. the ISO/IEC 7816 series.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23465-1:2023, *Card and security devices for personal identification — Programming interface for security devices — Part 1: Introduction and architecture description*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23465-1 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
array
indexed list of any data types with a well-known number of members

3.2
boolean
data type used to denote a data item that can only take one of the values TRUE and FALSE

3.3
char
data type containing an 8-bit quantity that encodes a single-byte character from any byte-oriented code set with a numerical value between 0 and 255

3.4
credential
set of data presented as evidence of a claimed or asserted identity and/or entitlements

EXAMPLE A user attribute (see ISO/IEC 19286) signed by the issuer as proof of authenticity is a credential that can be verified by the service provider by validating the electronic signature.

[SOURCE: ISO/IEC 29115:2013, 3.8, modified — Note 1 to entry was deleted. An EXAMPLE was added.]

3.5
integer
data type containing a sequence of digits taken from a number base

Note 1 to entry: Programming languages support integer values as a data type in different flavours, e.g. as signed integer or unsigned integer and in short or long format. To be programming language agnostic this document does not specify any of these different definitions and uses the general type integer for different types. This approach is different to the used interface description language (IDL).[6] It is the responsibility of the application programmer to define the type of integer in a relevant API function call according to the need of the function and the programming language used.

EXAMPLE Digits from the number base 10 (decimal) consisting of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

3.6
octet
data type with 8-bit quantity

3.7
string
data type containing a sequence of characters with a definite length

4 Symbols and abbreviated terms

APDU	application protocol data unit
API	application programming interface
CPLC	Card Production Life Cycle
CPS	Cryptographic Service Provider
DLOA	Digital Letter of Approval
eID	electronic identity
eSE	embedded secure element
eUICC	embedded universal integrated chip card
GP	GlobalPlatform
ID	identification
IDL	interface description language
KMS	Key Management System
OMG	Object Management Group
OS	operating system
OSI	open systems interconnection
PII	personal identifiable information
PIV	Personal Identity Verification
PKCS	Public Key Cryptographic Standard
SD	secure digital (memory card)
TSM	Trusted Service Manager eSE

5 Graduated APIs for security devices

5.1 General

A security device within an electronic device is characterized by its functionality.

A security device may act as

— a means for secure storage of credentials, without any additional functionality other than to retrieve the credentials,

— a means with cryptographic capabilities possibly storing credentials and offers in addition to cryptographic operations with these credentials,

— an eSE-application supporting device, storing the PII and offers eID-application related functionality. This may include related cryptographic capabilities and/or secure storage capabilities for any type of credentials.

Depending on the level of the use cases, different usage models and functionalities shall be considered. This leads to definitions of sub-sets of the full-flavoured APIs.

Figure 1 depicts the situation of an eSE-application supporting security device including the capabilities of a cryptography supporting device with means of a secure storage.

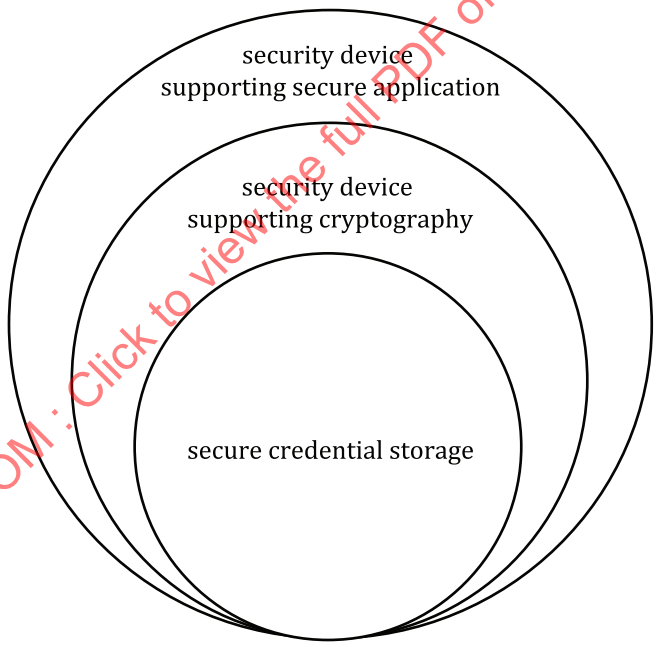

Figure 1 — Possible incorporated security device variants

5.2 Secure credential storage

In this use case the API functionalities are focused on the handling of the credentials. The administration, personalisation and usage of credentials are functions of this reduced API:

— Set data (see 10.4.5.5);

— Get data (see 10.4.5.4).

5.3 Security device supporting cryptography

A security device supporting cryptographic functionality supports a client application in addition to secure credential storage with additional security functionality related to cryptographic operations. There are a lot of cryptographic functions available, which allows an application to perform extended security protocols with the support of the security device. An example for such a cryptographic function is PKCS#11 functionality. [5]

A cryptographic supporting security device normally includes the creation, retrieval and deletion of credentials and is, therefore, also a credential storage.

Functions of security device supporting cryptography are e.g.:

— generate a Key (see 10.4.10.2);

— encrypt (see 10.5.3.2.2);

— decrypt (see 10.5.3.2.4).

5.4 Security device supporting secure application

Some client applications use a security device as a separate and secured application storage. An (ID-) application running on a standalone personal device, e.g. a health card or an ID document can be divided between a secured and unsecured part and be distributed, e.g. in a mobile application. For example, the secured part is located in the security device of the mobile device, the unsecured part is running in the mobile application itself. The combination of both the secured part and the unsecured part are completely running on the mobile device and can be a fully personalized ID-application communicating with a terminal.

A security device supporting ID applications may incorporate, in addition, the functionality of a cryptographic and/or secure storage supporting security device. Examples of functionality of such a security device are:

— PIV application;

— health application;

— mobile driving license.

6 API pre-requisite

6.1 Description language

The APIs defined in this document are outlined and defined with a generic interface description language (IDL). The IDL provides means to describe these interfaces in a language independent manner. Usually, additional language binding appendices are included in the basic descriptions of the IDL outlining how the IDL can be applied in the given language. The interface definition language defined in[6] is applied in this document. Additional information about the IDL is outlined in the informative Annex C.

6.2 Format of an API function

6.2.1 General

A generic API function consists of an explanatory name of the method/function, a list of parameters arguments and response data/values from the method invoke/function call. The name of a method/function, i.e. the API name, is typically understandable and self-explanatory. The terminology also signals the intent of the function. In this document the naming follows the java convention outlined in[3] and[4].

6.2.2 Addressing means

Addressing means allow the client application to use and address any security device on board of the electronic system.

6.2.3 Parameters

Most of method invokes/function calls need their related parameters. The parameters are a sequence of separated variables and shall be defined in each API separately.

6.2.4 Return values

The result of a method/function is returned to the invoker/caller. The type of the return value is method/function related and shall be defined for each API function separately.

The general format of an API method/function looks like:

```
<API_Return_Value> <API_Function_Name> (<Parameters>…)
raises Exeception 1, Exception 2, ….
```

The description of the API uses the following skeleton, defined in Table 1:

Table 1 — Format of the ISO/IEC 23465 API description

API Name	API_Function_Name
API Return value	***any*** API_Return_Value
API Parameter(s)	***in any*** inputparameter1,
	in any inputparameter2,
	…
	inout any inoutputparameter1
	inout any inoutputparameter2
	…
	out any outputparameter1
	out any outputparameter2
	…
Exceptions	Exception1
	Exception2
	…

The API function names standardized in this document use the prefix **`isoIec23465_`**. The case sensitivity follows the rules in conformance with the coding conventions in[3] and[4].

6.2.5 Callback functionality

To allow a non-blocking programming style or un-performant processing, the callback mechanism can apply. Synchronous and ansynchronous program processing are supported by many programming languages. These callback mechanisms may optionally be used.

If this applies, a callback function reference shall be conveyed in the parameter of each method. In the description tables of the methods in Clause 10, the callback function references in the parameter lists are not shown but have to be added if callback is used.

7 API error handling

7.1 General

Security devices using the ISO/IEC 7816 series APDUs indicate the processing status with the responses trailer SW1- SW2, especially for error conditions. In case of the API usage, the more efficient exception handling applies which is offered by modern programming languages.

The communication with the security device is performed by the API and its implementation. Any commands and responses to and from the security device are hidden from the application. In case of ISO/IEC 7816 related security devices, the response trailers are mapped to corresponding exceptions by the proxy.

7.2 Exceptions

The possible exceptions for each API method/function are outlined and explained in the API definition of each function.

Exceptions which are not dedicated to a specific method and are thrown, e.g. by the runtime environment or other components of the software system, are not listed explicitly. An example can be the exception “FunctionNotImplementedExeception”.

8 Security device identification

8.1 Security device attributes

As outlined in ISO/IEC 23465-1, this series of standards applies to systems where applications make use of security devices. It is possible that the system has access to more than one security device. A specific security device is characterized by a set of attributes. These attributes reflect high level information about the security device and shall allow a calling instance to identify the assigned security device. A set of attributes is proposed in Table 2.

Table 2 — Security device attributes

Type of information	Security device attribute	Type	Aim of information / data	Comment
Security device information	SecuritydeviceID	octet [ ]	Unique ID of the security device, Identifier of object of type SecurityDevice (see 10.4.3). can be absent or occur once or more times	Identification number, e.g. Card identification number For privacy reasons this ID can be replaced with a random number
	Security device capability_profile_ID	octet [ ]	Comprehensive description and abstraction of the capabilities of the security device and optionally the device, containing all information below	Profile can be used alternatively to describe SE (online availability), URL or OID can be possible
	Security device type	integer	Type of the security device	The types have to be defined in a separate table
	Security device owner	octet [ ]	Identifier of owner or issuer of the SE, also integrated-security-device owner	IIN Issuer Identification number
	Security device certification	octet [ ]	Information about certification of chip, OS and application (high level)	Data structure, e.g. GlobalPlatform DLOA
	Open Mobile API support	octet [ ]	Information regarding the support of OMAPI	Array of octet
	NFC support	boolean	Information regarding support of NFC	Yes/no
	CSP certification	octet [ ]	Information about certification of Cryptographic Service Provider CSP	Data structure including version and scheme
Chip information	Chip certification	octet [ ]	Information about the certification of the chip if other high-level information is not available	Array of octet
	Available chip memory	integer	Usable memory on the chip. In GP, this information is optional, it shall become mandatory.	Number of bytes on the security device
OS information	Available security device memory	integer	Usable memory in the OS.	Number of bytes in the OS
	Security device OS type, OS version, GP version and support	octet [ ]	OS-type and -version, GP-support and -version	TLV structure Java Card/native OS, OS-specification. GP specification, including version number
	SE_OS_Cryptography	octet [ ]	List of supported cryptographic algorithms/protocols	Bit map
GP information	Card Recognition Data	octet [ ]	Structure defined by GP-Services	TLV structure, see [8]
	Card Capability Data	octet [ ]	Structure defined by GP-Services	TLV structure, see [8]
Security device support	Key-Management-System information	octet [ ]	Information about the used key management system	relevant for the electronic system
	Discretionary Data	octet [ ]	Any data defined by the issuer	Array of octet

The security device attributes contain information derived from the different available security devices and additional data from the electronic system. The proxy collects the security device attribute from a security device and exposes it at the API (see Table 3). Security device information may also retrieved by usage of the Open Mobile API which is outlined in informative Annexes A and B.

8.2 Security device entry

In complex systems, an application has the choice to use at least one of several security devices. The securityDeviceEntry is a structure consisting of the securityDeviceNumber as a unique identification and numbering element and the descriptive attributes dedicated to the security device (see Table 2). The securityDeviceNumber assigned by the proxy, allows the selection of the security device for the further usage within the application. A securityDeviceEntry is defined as structure consisting of security device ID and the associated SdAttribute, mentioned in Table 3.

Table 3 — securityDeviceEntry

```
struct securityDeviceEntry {
                integer securityDeviceNumber;
                octet securityDeviceAttribute [ ];
        }
```

Subclause 10.3.1 describes a generic API function to get the list of available Security device entries in the electronic system.

9 Data model definition

9.1 General

The API used by a client software acts on instances of objects related to the security device system. In general, the API functions are methods and functions dealing with, and are assigned to, these objects. The general data model of a security device application is outlined in the class diagram in Figure 2.

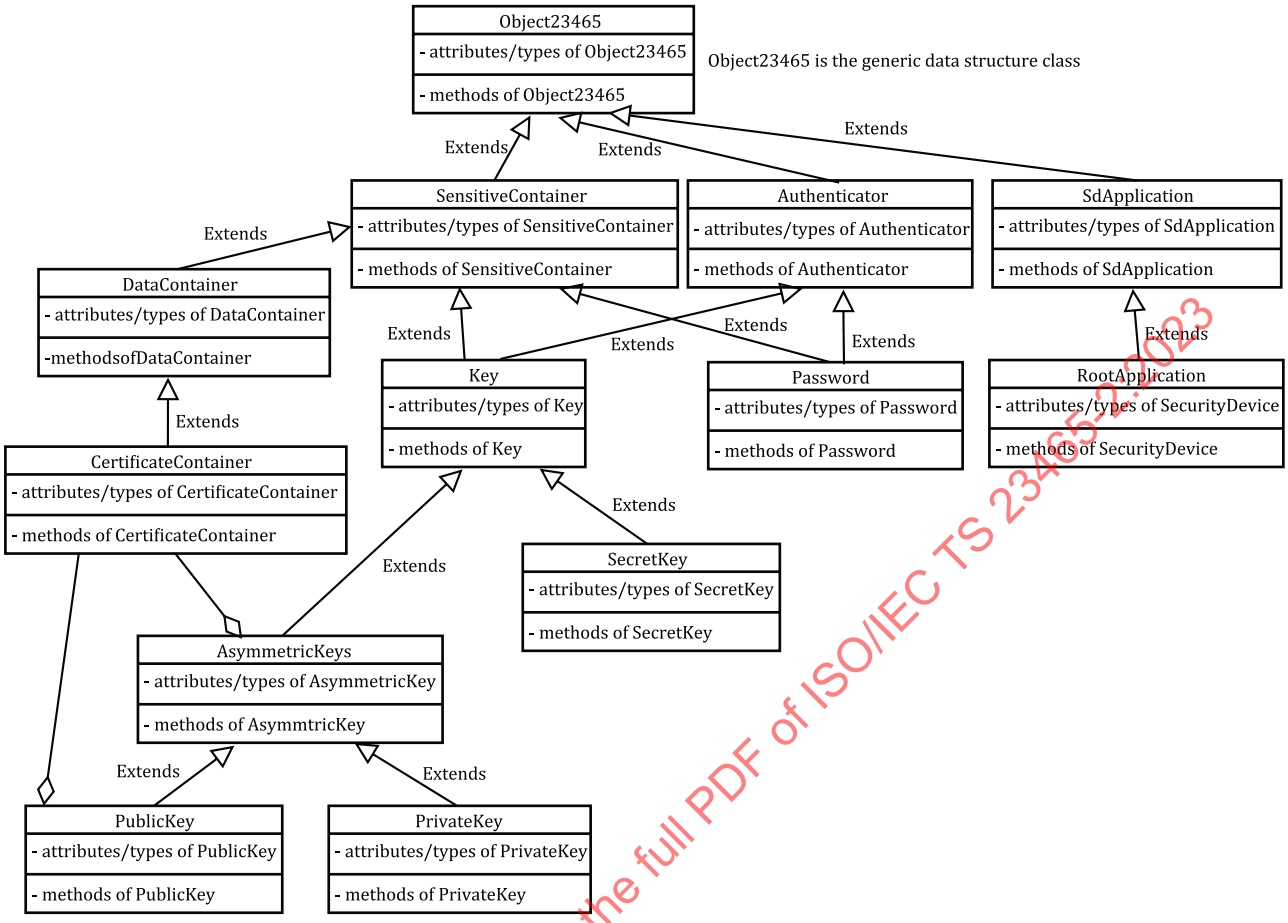

Figure 2 — General class diagram

9.2 Attributes and types

This document does not specify the implementation of classes from Figure 2. In general, neither class attributes nor instance attributes are specified in this document. The API defines getters and setters for handling relevant information.

9.3 Methods

The methods of each class are outlined in subclause 10.4.

9.4 References/instances

Addressing objects of the security device is achieved by using references to the instances of classes depicted in Figure 2. Programming languages often uses handles as the references to resources. Instead of these handles, references to class instances are used in the ISO/IEC 23465 series of standards.

ISO/IEC TS 23465-3[1)] describes the mechanisms of instantiation of the physical objects in the referenced security device.

1) Under preparation. Stage at the time of publication: ISO/IEC DTS 23465-3.

10 API definition

10.1 General

The API functions in this document fulfil the general requirements outlined in ISO/IEC 23465-1:2023, Annex A. This results in the following:

— In general, each API function has an inverse API function.

— Each API function acting on objects on the security device shall obey the access rules for action on such object, the application or proxy shall fulfil the access rules in advance.

— An API function acting on an object shall address the object within the function or has already selected it before.

— A selected area on the security device storing the relevant object for an application is identified and referenced by a reference to the instance, described in 9.4.

— Most of the API functions outlined in this API description are methods of data classes.

Subclause 10.4 describes the supported classes and also outlines UML diagrams. Each of these diagrams is separated into three parts describing

— the class name, and

— the public methods usable as API functions.

10.2 List of defined API functions

All defined API functions defined in Clause 10 are listed in Table 4:

Table 4 — Defined API functions

API function name	Link to the related subclause
isoIec23465_append	10.4.5.8
isoIec23465_ getSecurityDeviceList	10.3.1
isoIec23465_checkSupportedAlgorithm	10.4.9.3
isoIec23465_clearPWSecurityStatus	10.4.8.2
isoIec23465_clearSecurityStatus	10.4.2.2
isoIec23465_clearSecurityStatus	10.4.7.2
isoIec23465_clearSecurityStatusOfAllApplication	10.4.3.2
isoIec23465_connectSecurityDevice	10.3.2
isoIec23465_decrypt	10.5.3.2.4
isoIec23465_decryptInit	10.5.3.2.3
isoIec23465_deriveKey	10.5.3.5.5
isoIec23465_dhKeyAgreementInit	10.5.3.7.1
isoIec23465_dhSharedSecret	10.5.3.7.2
isoIec23465_digest	10.5.3.6.2
isoIec23465_digestInit	10.5.3.6.1
isoIec23465_disconnect	10.4.3.3
isoIec23465_encrypt	10.5.3.2.2
isoIec23465_encryptInit	10.5.3.2.1
isoIec23465_eraseSensitiveData	10.4.4.4
isoIec23465_generateKeyInstance	10.5.3.5.1

Table 4 *(continued)*

API function name	Link to the related subclause
isoIec23465_generateKeyPair	10.4.10.2
isoIec23465_getAccessCondition	10.4.1.3
isoIec23465_getAccessRules	10.4.1.2
isoIec23465_getActualSize	10.4.5.2
isoIec23465_getCertificate	10.4.12.2
isoIec23465_getContent	10.4.5.4
isoIec23465_getIdentifier	10.4.1.4
isoIec23465_getIssuerCertificate	10.4.6.2
isoIec23465_getKeyAttributes	10.5.2.2
isoIec23465_getMaximumSize	10.4.5.3
isoIec23465_getPrivateKey	10.4.13.2
isoIec23465_getPublicKey	10.4.12.3
isoIec23465_getRandom	10.5.3.9
isoIec23465_getRetryCounter	10.4.8.3
isoIec23465_getSecurityStatusEvaluationCounter	10.4.7.3
isoIec23465_getStartValueRetryCounter	10.4.8.4
isoIec23465_getStartValueSecurityStatusEvaluationCounter	10.4.7.4
isoIec23465_getSupportedAlgorithm	10.4.9.2
isoIec23465_insert	10.4.5.6
isoIec23465_isErased	10.4.4.2
isoIec23465_remove	10.4.5.9
isoIec23465_selectObject	10.4.2.3
isoIec23465_selectSecurityDeviceApplication	10.4.3.4
isoIec23465_setContent	10.4.5.5
isoIec23465_setPrivateKey	10.4.13.3
isoIec23465_setPublicKey	10.4.12.4
isoIec23465_setPassphrase	10.4.8.5
isoIec23465_setSensitiveData	10.4.4.3
isoIec23465_sign	10.5.3.3.2
isoIec23465_signInit	10.5.3.3.1
isoIec23465_unblock	10.4.8.6
isoIec23465_unWrapKey	10.5.3.5.4
isoIec23465_update	10.4.5.7
isoIec23465_verifyPassword	10.4.8.7
isoIec23465_verifySignature	10.5.3.4.2
isoIec23465_verifySignatureInit	10.5.3.4.1
isoIec23465_erasePrivateKey	10.4.13.4
isoIec23465_wrapKey	10.5.3.5.3

10.3 API function for managing, addressing and identifying security devices

10.3.1 Method isoIec23465_ getSecurityDeviceList

A security device is characterized by its securityDeviceAttribute, specified in subclause 8.1. The securityDeviceNumber is used as an identification and addressing means. An application requests the

list of available security devices in the system and retrieves with this functionality the list of available security devices or (equivalently) a list of smart card readers with a security device inserted.

The API function is shown in Table 5.

Table 5 — Signature of method isoIec23465_ getSecurityDeviceList

API Name	isoIec23465_ getSecurityDeviceList
API Return Value	***struct*** securityDeviceEntry []
API Parameter(s)	—
Exceptions	—

The list of securityDeviceEntry is used by the application to select the appropriate security device. The API function returns an empty list of securityDeviceEntry if no security device entries is available.

10.3.2 Method isoIec23465_connectSecurityDevice — Connection to the security device

The application uses the definite securityDeviceNumber from the list of securityDeviceEntry to identify and address the security device that shall be used. As shown in Table 6, the method uses the securityDeviceNumber as the input parameter, addresses the security device and establishes a connection to the security device. A successful connection is acknowledged with the base reference to the security device's RootApplication.

Table 6 — Signature of method isoIec23465_connectSecurityDevice ()

API Name	isoIec23465_connectSecurityDevice
API Return Value	***Object23465*** RootApplication
API Parameter(s)	***in integer*** securityDeviceNumber
Exceptions	ConnectionFailedException
	InvalidParameterException

The API function throws ConnectionFailedException if the security device cannot be connected. An InvalidParameterException is thrown if the given securityDeviceNumber cannot be associated with an available security device.

10.4 API function derived from data model

10.4.1 Basic Class Object

10.4.1.1 General

The basic class Object23465 is depicted in Figure 3.

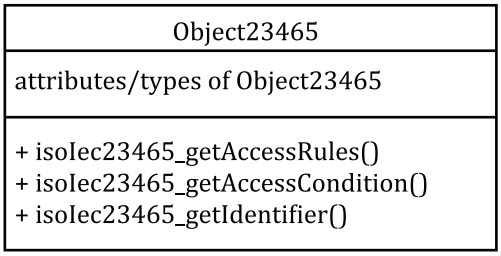

Figure 3 — Object23465 class

10.4.1.2 Method isoIec23465_getAccessRules

Table 7 shows the method description for retrieving all acess rules by which the object is protected from unauthorized usage.

Table 7 — Signature of method isoIec23465_getAccessRules

API Name	isoIec23465_getAccessRules()
API Return Value	***struct*** AccessRules [ ]
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The attribute AccessRules is related to the definition of security attributes in ISO/IEC 7816-4. An API function, with dedicated access to the object on the security device, can only be processed if the corresponding access condition for this access mode is fulfilled. Otherwise the access is denied by the security device.

The return value of the method is the corresponding list of AccessRules.

The API function throws SecurityStatusNotSatisfiedException if the access rule for this operation is not fulfilled.

10.4.1.3 Method isoIec23465_getAccessCondition

Table 8 shows the method description for retrieving the access condition for a given access mode by which anobject is protected from unauthorized usage.

Table 8 — Signature of method isoIec23465_getAccessCondition

API Name	isoIec23465_getAccessCondition()
API Return Value	***string*** AccessCondition [ ]
API Parameter(s)	***in string*** accessMode
Exceptions	SecurityStatusNotSatisfiedException

The attributes accessMode and accessCondition are related to the definition of security attributes in ISO/IEC 7816-4. An API function with dedicated access to the object on the security device can only processed if the corresponding access condition for this access is fulfilled. Otherwise the access is prohibited.

The return value of the method is the corresponding AccessCondition related to the addressed object.

The API function throws SecurityStatusNotSatisfiedException if the access rule for this operation is not fulfilled.

10.4.1.4 Method isoIec23465_getIdentifier

Table 9 shows the method description for retrieving an attribute from implementing classes storing its identifiers.

Table 9 — Signature of method isoIec23465_getIdentifier

API Name	isoIec23465_getIdentifier()
API Return Value	***struct*** { octet identifier[ ] } identifierList[ ]
API Parameter(s)	—

Table 9 *(continued)*

Exceptions	SecurityStatusNotSatisfiedException

The method returns all available identifiers of an object as a list of Identifier.

The instances of objects have zero, one or more identifier used to identify the object during a selection process. Each identifier identifying an object is an arbitrary non-empty octet string.

The return value is an array of all identifiers of the instance of this interface, possibly empty if an instance has no identifier

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.2 Class SecurityDeviceApplication

10.4.2.1 General

Class SecurityDeviceApplication, as shown in Figure 4, extends class Object23465 and is the base for all applications residing in a security device. Typically, a security device hosts one or more SecurityDeviceApplications. Each SecurityDeviceApplication typically hosts one or more children, e.g. DataContainer storing information. A SecurityDeviceApplication is the ISO/IEC 23465 series equivalent of ISO/IEC 7816-4 terms DF, application DF and application.

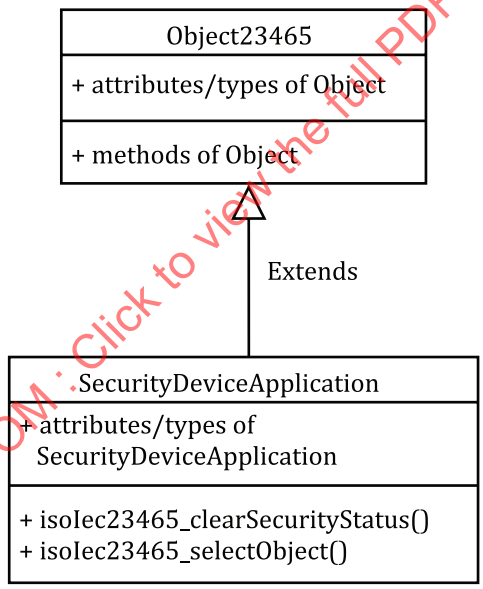

Figure 4 — SecurityDeviceApplication class

10.4.2.2 Method isoIec23465_clearSecurityStatus

Table 10 shows the method description for clearing the security status of a SecurityDeviceApplication.

Table 10 — Signature of method isoIec23465_ clearSecurityStatus

API Name	isoIec23465_clearSecurityStatus()
API Return Value	***void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method clears the security status of all children of this securityDeviceApplication of type Authenticator (see subclause 10.4.7) regardless of their access condition according to clearing the authenticator security status.

NOTE 1 Implementations behave such that if the access condition for this method is fulfilled then the security status of all children of type Authenticator is cleared regardless of whether the security status for method Authenticator.isoIec23465_clearSecurityStatus() is fulfilled or not.

NOTE 2 According to ISO/IEC 7816-4, the security status is related to the application usually referenced by MF or DF. Elementary files or data object related to this application are also related to this security status. Access conditions are approved according to this security status.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.2.3 Method isoIec23465_selectObject

Table 11 shows the method description for selecting an object within a SecurityDeviceApplication.

Table 11 — Signature of method isoIec23465_selectObject

API Name	isoIec23465_selectObject()
API Return Value	***Object23465*** Object
API Parameter(s)	***in octet*** identifier[]
Exceptions	ObjectNotFoundException, SecurityStatusNotSatisfiedException

The method selects an object of this SecurityDeviceApplication with a given identifier. If such a child exists, the reference to the object is returned, otherwise the exception ObjectNotFoundException is thrown.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

NOTE isoIec23465_selectObject is used to select any selectable entity related to the SecurityDeviceApplication.

10.4.3 Class RootApplication

10.4.3.1 General

The class RootApplication, as shown in Figure 5, extends class SecurityDeviceApplication. Each security device contains exactly one instance of RootApplication, storing global attributes and possibly global objects of a security device The class RootApplication is the ISO/IEC 23465 series equivalent of ISO/IEC 7816-4 term MF.

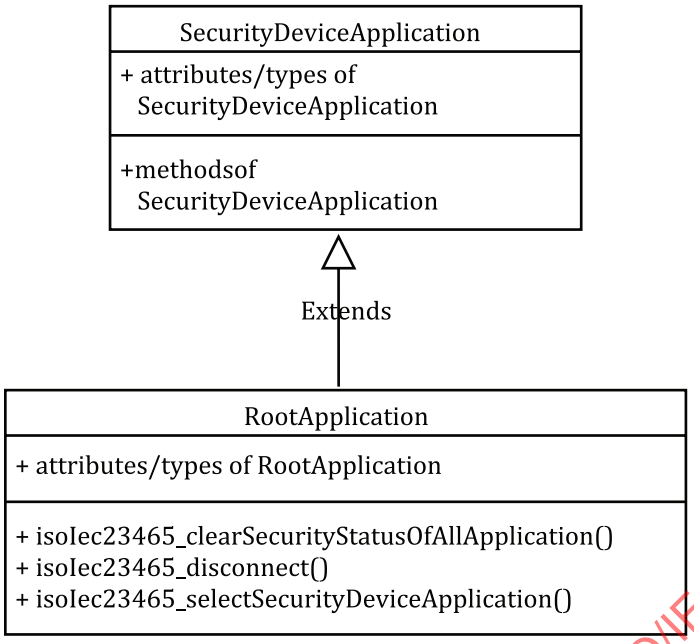

Figure 5 — RootApplication class

10.4.3.2 Method isoIec23465_clearSecurityStatusOfAllApplication()

Table 12 shows the method description for clearing the security status of all SdApplication within the security device.

Table 12 — Signature of method isoIec23465_clearSecurityStatusOfAllApplication

API Name	isoIec23465_clearSecurityStatusOfAllApplication()
API Return Value	***void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method clears, recursively, the security status of type Authenticator of this instance and of all descendants regardless of their access rules. If the access condition for this operation is not fulfilled, the method throws SecurityStatusNotSatisfiedException.

Assume the access condition for RootApplication.isoIec23465_clearSecurityStatusOfAllApplication() is set to ALWAYS and that security device contains Authenticator where Authenticator.isoIec23465_clearSecurityStatus() has an access condition set to interface NEVER. Then the implementation shall behave such that calling Authenticator.isoIec23465_clearSecurityStatus() throws a SecurityStatusNotSatisfiedException, but calling RootApplication.isoIec23465_clearSecurityStatusOfAllApplication() doesn't throw an exception and clears the security status of all objects.

NOTE According to ISO/IEC 7816-4, the security status is related to the application usually referenced by MF or DF. Elementary files or data object related to this application are also related to this security status. Access conditions are approved according to this related security status.

10.4.3.3 Method isoIec23465_disconnect()

Table 13 shows the method description for disconnecting from a security device.

Table 13 — Signature of method isoIec23465_disconnect

API Name	isoIec23465_disconnect()
API Return Value	***void***
API Parameter(s)	—
Exceptions	—

This method closes the logical connection to this SecurityDevice. No access rules shall apply to this functionality and it can always be used by applications.

10.4.3.4 Method isoIec23465_selectSecurityDeviceApplication()

Table 14 shows the method description for selecting a SecurityDeviceApplication by one of its identifiers.

Table 14 — Signature of method isoIec23465_selectSdApplication

API Name	isoIec23465_selectSecurityDeviceApplication()
API Return Value	***Object23465*** securityDeviceApplication
API Parameter(s)	***in octet*** identifier[]
Exceptions	ObjectNotFoundException

The method selects a securityDeviceApplication by one of its (unique) identifiers given in the parameter. Within the security device, all securityDeviceApplication are searched to find one which has an identifier equal to the given identifier. If such a securityDeviceApplication exists, the reference of this securityDeviceApplication is returned. If there is no appropriate securityDeviceApplication found the exception ObjectNotFoundException shall be thrown.

NOTE 1 Intentional selection is always allowed. Therefore, no access rules are checked for a select operation.

NOTE 2 Contrary to isoIec23465_selectObject, this method is used to select a securityDeviceApplication which is never part of another securityDeviceApplication.

10.4.4 Class SensitiveContainer

10.4.4.1 General

This class provides means to set or erase sensitive content in implementing classes. Classes storing sensitive data (e.g. key-material) implement this class in order to provide means for erasing that sensitive data without deleting the instance of that object.

A SensitiveContainer has a non-volatile status with two states “erased” and “not erased”. Method isErased() shows that status. Methods setSensitiveData() and eraseSensitiveData() switch between those states.

Class SensitiveContainer, as shown in Figure 6, is a basis for objects storing sensitive data.

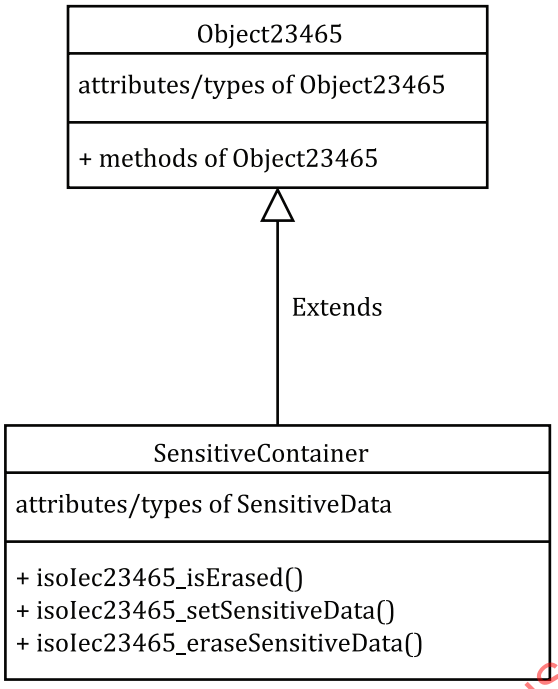

Figure 6 — SensitiveContainer class

10.4.4.2 Method isoIec23465_isErased()

Table 15 shows the method description for retrieving information, whether the sensitive data in a container is securely erased or still securely set.

Table 15 — Signature of method isoIec23465_isErased

API Name	isoIec23465_isErased()
API Return Value	***boolean***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The methods return the status of sensitive content as a boolean value. If the sensitive data is erased, the method returns true, otherwise false.

If sensitive data is erased, it is possible that some methods could not be performed (see isoIec23465_eraseSensitiveData()-method, 10.4.4.4).The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.4.3 Method isoIec23465_setSensitiveData()

Table 16 shows the method description for setting sensitive data.

Table 16 — Signature of method isoIec23465_setSensitiveData

API Name	isoIec23465_setSensitiveData ()
API Return Value	***void***
API Parameter(s)	***in any*** data []
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentNotErasedException
	Exception23465

This method uses data from the corresponding input parameter to set certain atttributes in the corresponding object. For a successful execution the status of sensitive, content shall be “not erased”.

Implemented classes give more information about which attributes are set and how parameter data is interpreted. After a successful performance of this method, isoIec23465_isErased()-method shall return FALSE. This method is the counterpart of isoIec23465_ eraseSensitiveData().

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The exception SensitiveContentNotErasedException is thrown if sensitive content is not in status "erased". Exception23465 may occur if classes throw additional exceptions.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentNotErasedException;

— Exception23465.

NOTE Usually, containers provide setters for content. Typically, such setters are used in a sense of "change content". The method isoIec23465_setSensitiveData (any[]) has a sense of "set something which is actually not set". It seems worth it to distinguish between these situations and thus have distinguished methods where each of those methods has its own access rule protecting it.

10.4.4.4 Method isoIEC23465_eraseSensitiveData()

Table 17 shows the method description for wiping sensitive data.

Table 17 — Signature of method isoIec23465_eraseSensitiveData

API Name	isoIec23465_eraseSensitiveData ()
API Return Value	***void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method erases sensitive content. For a successful execution of this method, it is irrelevant if the status of sensitive content is “erased” or “not erased” as a pre-condition.

Implemented classes give more information about which data is erased and what methods could not be performed if sensitive data is erased. After a successful performance of this method, isoIec23465_ isErased()-method shall return TRUE. This method is the counterpart of isoIec23465_setSensitiveData ().

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.5 DataContainer

10.4.5.1 General

This class describes the container type used for storing arbitrary information, e.g. data. A container has the following behaviour:

— A container has a body used for storing arbitrary information.

— The content of this body can be read and changed.

— The data container shall ensure that upon changing the body, the maximum size for the body is not exceeded.

— The actual size of body is less than its maximum size in case the maximum storage capacity of a data container is not used.

— Offset parameter used in some methods shall be treated as follows:

  — Offset zero addresses the first octet in the body.

  — Offset (i+1) addresses the octet immediately after the octet at offset i.

  — Negative offsets and offsets larger than actualSize shall be internally re-calculated according to the formula “offset = offset mod actualSize”.

NOTE With valid negative offsets it is easy to address octets at the end of body, e.g. offset (-1) addresses the last octet in the body.

The class DataContainer as shown in Figure 7, extends class SensitiveContainer and is the basis for storing arbitrary data, which typically is not interpreted by a security device.

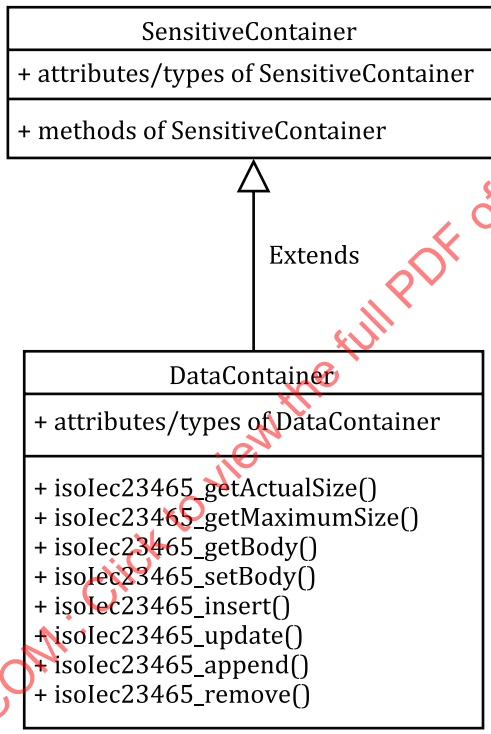

Figure 7 — DataContainer class

10.4.5.2 Method isoIec23465_getActualSize()

Table 18 shows the method description for retrieving the actual size of the body.

Table 18 — Signature of method isoIec23465_getActualSize

API Name	isoIec23465_getActualSize()
API Return Value	***integer*** containerLength
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentNotErasedException

This method retrieves the actual number of octets in the body of the container. The number is returned as an integer.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased".

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentNotErasedException.

10.4.5.3 Method isoIec23465_getMaximumSize()

Table 19 shows the method characteristics for retrieving the maximum size of body.

Table 19 — Signature of method isoIec23465_getMaximumSize

API Name	isoIec23465_getMaximumSize()
API Return Value	***integer*** MaximumNumberOfOctets
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method retrieves the maximum size of octets in the body of this Container and returns it in an integer.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.5.4 Method isoIec23465_getBody()

Table 20 shows the method description for retrieving the actual content of body.

Table 20 — Signature of method isoIec23465_getBody

API Name	isoIec23465_getBody()
API Return Value	***any*** content[]
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentNotErasedException

This method retrieves the complete body of the Container which is returned as an array of octets.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased".

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentNotErasedException.

10.4.5.5 Method isoIec23465_setBody()

Table 21 shows the method description for setting the new content of body.

Table 21 — Signature of method isoIec23465_setBody

API Name	isoIec23465_setBody()
API Return Value	***void***
API Parameter(s)	***in any*** newContent[]
Exceptions	SecurityStatusNotSatisfiedException
	OutOfMemoryException

The method replaces the actual content of body by the given one. The parameter newContent[] contains the new content of body.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws OutOfMemoryException if the number of octets in newContent is greater than maximumSize.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— OutOfMemoryException.

10.4.5.6 Method isoIec23465_insert ()

Table 22 shows the method description for inserting new content into the body of the Container.

Table 22 — Signature of method isoIec23465_insert

API Name	isoIec23465_insert ()
API Return Value	***void***
API Parameter(s)	***in any*** newContent[ ]
	in integerBodyOffset
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveDataErasedException
	OutOfMemoryException

The method insert the newContent[ ] into the actual content of body at the defined BodyOffset which indicates an octet in the body.

EXAMPLE If the old content of the body is '0104' and the method isoIec23465_insertContent('0203', 1) is performed, then the new content is '01020304'.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased". The method throws OutOfMemoryException if the number of octets in body plus number of octet in newContent is greater than maximumSize.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentNotErasedException;

— OutOfMemoryException.

10.4.5.7 Method isoIec23465_update()

Table 23 shows the method description for updating (part of) the content with newContent.

Table 23 — Signature of method isoIec23465_update

API Name	isoIec23465_update()
API Return Value	***void***
API Parameter(s)	***in any*** newContent[ ]
	in integer BodyOffset
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveDataErasedException
	OutOfMemoryException

The method replaces octets of body with octets from parameter newContent. The first octet in newContent replaces the octet at position offset in body. The following octets from newContent (if any) replace the following octets in body or extend the body.

EXAMPLE 1 If the old content of body is '01020304' and the method update('f2f3', 1) is performed, then the new content is '01f2f304'.

EXAMPLE 2 If the old content of body is '010203' and the method update('f304', 2) is performed, then the new content is '0102f304'.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased". The method throws OutOfMemoryException if (effective) offset plus number of octets in newContent is greater than maximumSize.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentNotErasedException;

— OutOfMemoryException.

10.4.5.8 Method isoIec23465_append()

Table 24 shows the method description for appending newContent to the content.

Table 24 — Signature of method isoIec23465_append

API Name	isoIec23465_ append ()
API Return Value	***void***
API Parameter(s)	***in any*** newContent[ ]
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentNotErasedException
	OutOfMemoryException

The method appends octets from parameter newContent at the body with octets.

EXAMPLE 1 If the content of body is '01020304' and the method append('f2f3') is performed, then the new content is '01020304f2f3'.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased". The method throws OutOfMemoryException if number of octets of body plus number of octets in newContent is greater than maximumSize.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveDataErasedException;

— OutOfMemoryException.

10.4.5.9 Method isoIec23465_remove()

Table 25 shows the method description for removing addressed octets from the body of the DataContainer.

Table 25 — Signature of method isoIec23465_remove

API Name	isoIec23465_remove ()
API Return Value	***void***
API Parameter(s)	***in integer***Offsetfirst,
	in integer offsetLast
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveDataErasedException
	IllegalArgumentException

The method removes the octet addressed by offsetFirst and the octet addressed by offsetLast and all octets in between.

EXAMPLE 1 If the old content of body is '01020304' and the method remove(1, 2) is performed, then the new content is '0104'.

EXAMPLE 2 In case the complete body is to be removed, the following call is always appropriate: remove(0, -1).

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. The method throws SensitiveDataErasedException if the status of this container (seen as a SensitiveContainer, see 10.4.4.2) is "erased". The method throws IllegalArgumentException if (effective) offsetFirst is greater than (effective) offsetLast.

In order to protect instance attributes from leaking the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveDataErasedException;

— IllegalArgumentException.

10.4.6 CertificateContainer

10.4.6.1 General

This class describes a special type of DataContainer storing certificates.

Typically, a certificate contains a public key and such attributes of that public key, which are relevant for client applications. A certification authority signs this information and that digital signature is also part of the certificate.

In case the digital signature of a certificate is verified, the public key of the issuing certification authority shall be known. In order to provide the issuer's public key, a CertificateContainer contains a

reference to the CertificateContainer containing the issuer's certificate that stores the issuer's public key.

Class CertificateContainer, as shown in Figure 8, extends class DataContainer.

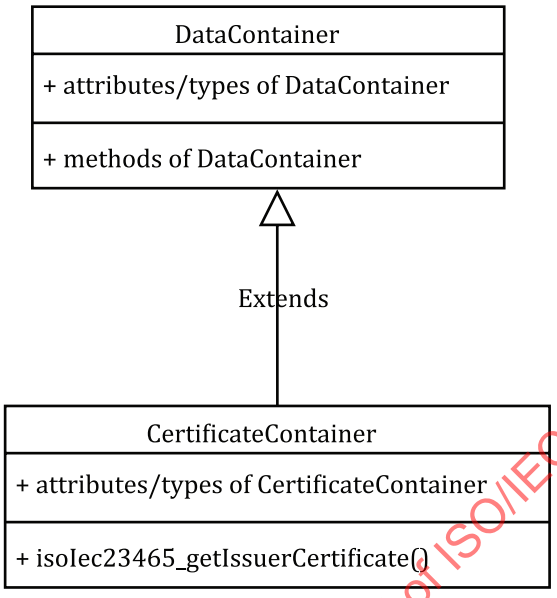

Figure 8 — CertificateContainer class

10.4.6.2 Method isoIec23465_getIssuerCertificate()

Table 26 shows the method description for retrieving the CertificateContainer that contains the issuer's certificate.

Table 26 — Signature of method isoIec23465_getIssuerCertificate

API Name	isoIec23465_getIssuerCertificate()
API Return Value	***CertificateContainer*** IssuerCertificate
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException
	ObjectNotFoundException

This method retrieves the CertificateContainer containing the issuer's certificate that stores the issuer's public key necessary for verifying the content of a CertificateContainer.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

The method throws ObjectNotFoundException if this container doesn't reference the issuer's certificate.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— ObjectNotFoundException.

10.4.7 Authenticator

10.4.7.1 General

This class is the base for data types used for authentication, e.g. passwords or authentication keys.

The class uses the securityStatusEvaluationCounter, which counts how often the security status of these related data types has been applied to protect a method. The start value for this counter is startValueSecurityStatusEvaluationCounter and defines the maximum tries of security evaluation.

The class defines the constant INFINITY representing a value indicating that the security status evaluation counter has the value "infinite". The startValueSecurityStatusEvaluationCounter shall be an integer in the range [1, n] or INFINITY.

The securityStatusEvaluationCounter is decremented for each status evaluation and is in the range between [0,startValueSecurityStatusEvaluationCounter].

NOTE If startValueSecurityStatusEvaluationCounter is INFINITY and setting the security status is successful then securityStatusEvaluationCounter is also set to INFINITY.

The class Authenticator, as shown in Figure 9, is a basis class for authentication types.

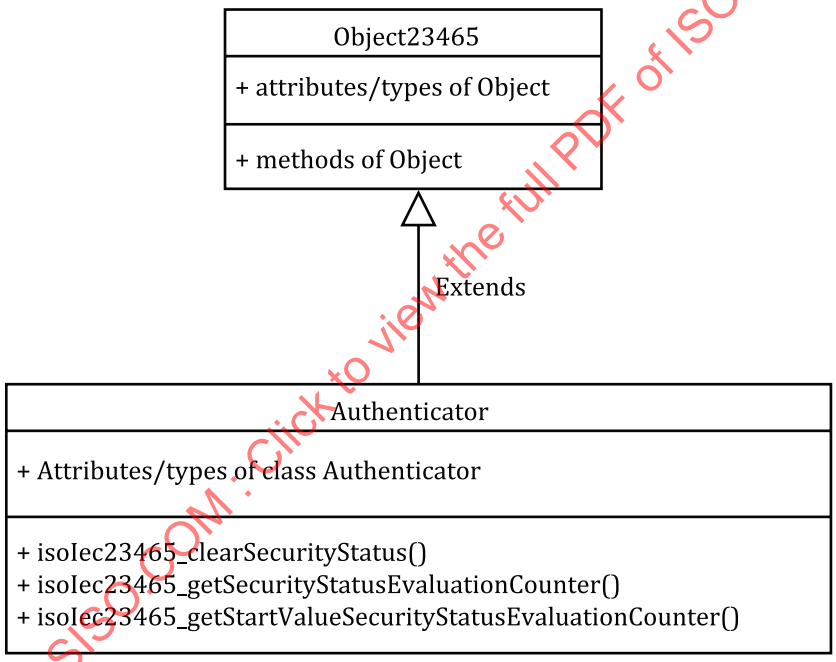

Figure 9 — Authenticator class

10.4.7.2 Method isoIec23465_clearSecurityStatus()

Table 27 shows the method description for clearing the security status.

Table 27 — Signature of method isoIec23465_clearSecurityStatus

API Name	isoIec23465_clearSecurityStatus()
API Return Value	***Void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method clears the security status of an authentication object. This is the counterpart of setting the security status, e.g. by a user verification or an authentication protocol.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.7.3 Method isoIec23465_getSecurityStatusEvaluationCounter()

Table 28 shows the method description for getting the actual value of the security status evaluation counter.

Table 28 — Signature of method isoIec23465_getSecurityStatusEvaluationCounter

API Name	isoIec23465_getSecurityStatusEvaluationCounter()
API Return Value	***integer*** actualSecurityStatusEvaluationCounter
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method returns the actual value of security status evaluation counter.

When the security status for authentication object is set, this method shall return the actual value of the security status evaluation counter for this authentication object. Otherwise the method returns zero. If the startValueSecurityStatusEvaluationCounter is INFINITY, the method returns INFINITY.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.7.4 Method isoIec23465_getStartValueSecurityStatusEvaluationCounter()

Table 29 shows the method description for getting the start value of SecurityStatusEvaluationCounter.

Table 29 — Signature of method isoIec23465_getStartValueSecurityStatusEvaluationCounter

API Name	isoIec23465_getStartValueSecurityStatusEvaluationCounter()
API Return Value	***integer*** startValueSecurityStatusEvaluationCounter
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method shall return the start value of the security status evaluation counter for this authentication object. The start value is a positive integer or INFINITY.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.8 Password

10.4.8.1 General

This class describes the password type used for user verification.

A password has the following behaviour:

— A password contains a passphrase, usually only known to the user of the secure device.

— A retry counter is dedicated to the password, counting successive unsuccessful verification attempts.

— A security evaluation counter and its start value, as defined in 10.4.7.

— Upon a successful verification with isoIec23465_verifyPassword(string)

  — the security status evaluation counter is set to its start value,

  - the retry counter is set to its start value.
- Upon an unsuccessful isoIec23465_verifyPassword(String)
  - the volatile security status of this password is cleared,
  - the retry counter is decremented.
- In order to be in accordance with this document, an implementation shall meet the following requirements:
  - Start value retry counter shall be an integer from range [1, x] and x shall be 15 or higher.
  - Retry counter is an integer in range [0, startValueRetryCounter].

Class Password, as shown in Figure 10 extends the classes Authenticator and SensitiveContainer and serves the purpose of user verification.

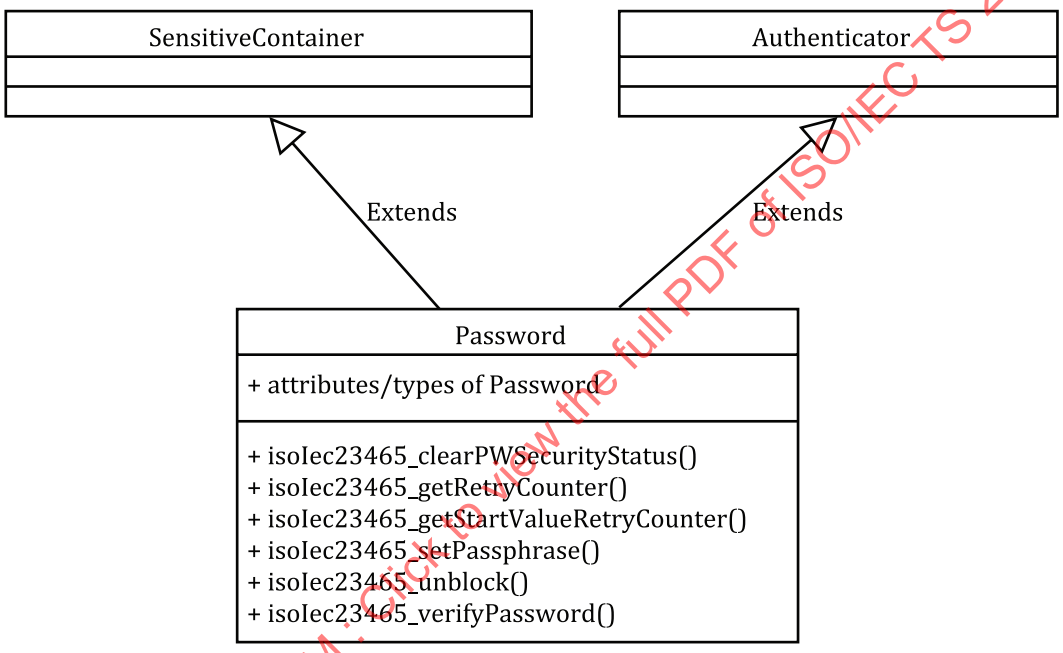

Figure 10 — Password class

A Password has a volatile status, shown in Figure 11.

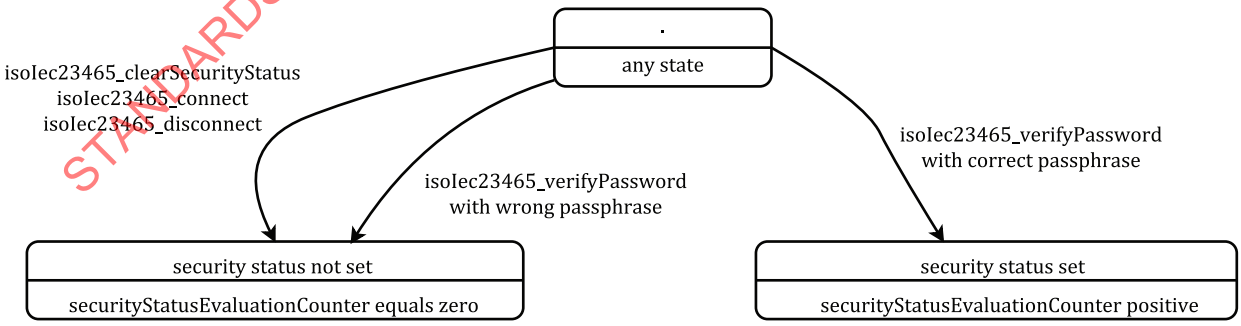

Figure 11 — Volatile status of a Password

Regardless of the actual state, clearing the security status, connecting to a security device or disconnecting from a security devise clears the security status, i.e. the security status evaluation counter is set to zero.

Upon a successful user verification, the security status is set, i.e. the security status evaluation counter is set to its start value, a positive integer or INFINITY.

10.4.8.2 Method isoIec23465_clearPWSecurityStatus()

Table 30 shows the method description for clearing the security status of a password.

Table 30 — Signature of method isoIec23465_clearPWSecurityStatus

API Name	isoIec23465_clearPWSecurityStatus()
API Return Value	***void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method clears the security status of the password and is the counterpart of user verification.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.8.3 Method isoIec23465_getRetryCounter()

Table 31 shows the method description for retrieving the actual value of retry counter.

Table 31 — Signature of method isoIec23465_getRetryCounter

API Name	isoIec23465_getRetryCounter()
API Return Value	***integer*** ActualValueRetryCounter
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method returns the actual value of the retry counter. If the value is positive, then user verification is not blocked. If the value is zero, then user verification is blocked.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.8.4 Method isoIec23465_getStartValueRetryCounter()

Table 32 shows the method description for retrieving the start value of the retry counter.

Table 32 — Signature of method isoIec23465_getStartValueRetryCounter

API Name	isoIec23465_getStartValueRetryCounter()
API Return Value	***integer*** StartValueRetryCounter
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method returns the start value of the retry counter.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.8.5 Method isoIec23465_setPassphrase()

Table 33 shows the method description for setting the passphrase used for user verification.

Table 33 — Signature of method isoIec23465_setPassphrase

API Name	isoIec23465_setPassphrase()
API Return Value	***void***
API Parameter(s)	***in string*** NewPassPhrase
Exceptions	SecurityStatusNotSatisfiedException

This method replaces the actual value of passphrase by the NewPassPhrase conveyed in the API parameter and sets the status of the object to not-erased. The new value of passphrase by which a user wants to be verified in the future is an arbitrary UTF-8 string.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

NOTE Intentionally this method works even if the Password is in status "erased".

10.4.8.6 Method isoIec23465_unblock()

Table 34 shows the method description for unblocking a password.

Table 34 — Signature of method isoIec23465_unblock

API Name	isoIec23465_unblock()
API Return Value	***void***
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method sets the retry counter to its start value.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.8.7 Method isoIec23465_verifyPassword()

Table 35 shows the method description for user verification.

Table 35 — Signature of method isoIec23465_verifyPassword

API Name	isoIec23465_verifyPassword()
API Return Value	***boolean***
API Parameter(s)	***in string*** Passphrase
Exceptions	PasswordBlockedException
	SecurityStatusNotSatisfiedException
	SensitiveContentErasedException

This method compares the given passphrase with the passphrase stored by this password object. If the comparison is successful the method returns TRUE, FALSE otherwise.

A successful comparison sets the security status of the password, the retry counter is set to its start value. If the comparison is not successful then the security status of the password is cleared, and the retry counter is decremented.

This method is the counterpart of isoIec23465_clearSecurityStatus().

The method throws PasswordBlockedException when the actual value of retry counter is zero. The exception SensitiveContentErasedException is thrown when the sensitiveContent is in the status

"erased". The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentErasedException;

— PasswordBlockedException.

10.4.8.8 Method isoIec23465_setSensitiveData () for passphrase

The method of SensitiveContainer is applied to set the content of passphrase to the given one and sets the status of the object to not-erased. The passphrase is an octet string representation of a UTF-8 string.

Details of usage of the method are outlined in 10.4.4.3.

10.4.8.9 Method isoIec23465_eraseSensitiveData() for passphrase

The method of SensitiveContainer is applied to erase the content of passphrase securely. The status of the object is transferred to status "erased".

Details of usage of the method are outlined in 10.4.4.4.

10.4.9 Key

10.4.9.1 General

The key class describes the key type used for any cryptographic operations. The key class, as shown in Figure 12, extends the SensitiveContainer class (see 10.4.4). The key type is defined by the object identifier.

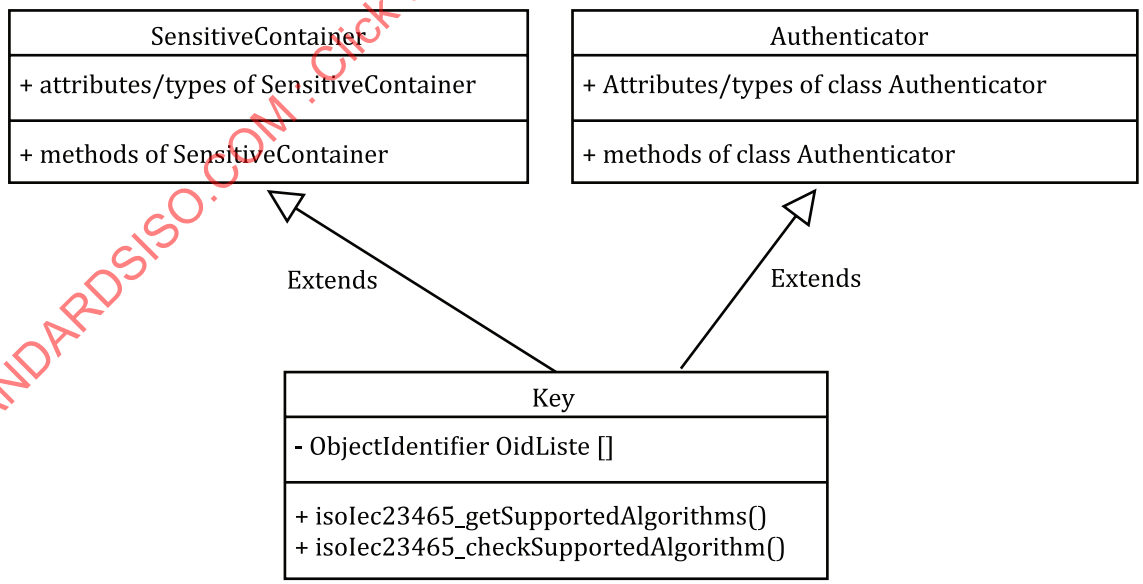

Figure 12 — Key class

10.4.9.2 Method isoIec23465_getSupportedAlgorithm()

Table 36 shows the method description for retrieving the OID list of the key describing the supported algorithms.

Table 36 — Signature of method isoIec23465_getSupportedAlgorithm

API Name	isoIec23465_getSupportedAlgorithm()
API Return Value	***string*** ObjectIdentifierList[ ]
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

The method returns the OID list for identification of supported algorithms the key supplies.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.9.3 Method isoIec23465_checkSupportedAlgorithm()

Table 37 shows the method description for checking a specific algorithm this key supports.

Table 37 — Signature of method isoIec23465_getSupportedAlgorithm

API Name	isoIec23465_getSupportedAlgorithm()
API Return Value	***boolean*** AlgorithmSupported
API Parameter(s)	***in string*** AlgorithmToBeChecked
Exceptions	SecurityStatusNotSatisfiedException

The method returns the checking result that the ObjectIdentifier is in the OID list of the key.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.4.10 AsymmetricKey

10.4.10.1 General

This class is the base class used for any asymmetric key types and algorithms. Figure 13 depicts the dependency of AsymmetricKey class which extends the Key class. There is also an aggregation to CertificateContainer dealing with asymmetric keys.

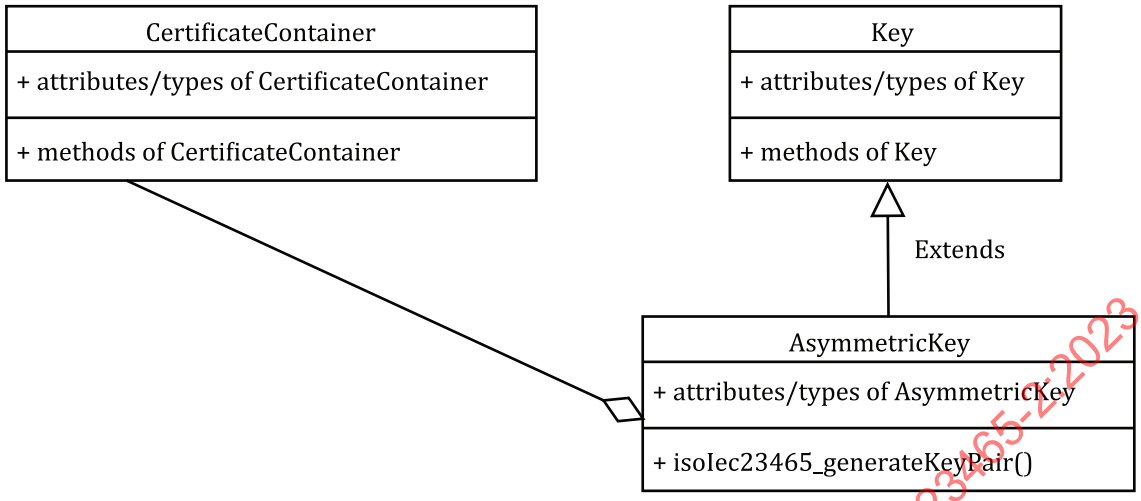

Figure 13 — AsymmetricKey class

10.4.10.2 Method isoIec23465_generateKeyPair()

With this API function an asymmetric key pair will be generated in the security device. The key information related to the key is conveyed in the parameters. Relevant instances of Keys are generated internally, holding the key relevant attributes, OIDs, etc. The method allows the generation, e.g. ECC or RSA key pairs. As outlined in Table 38 the successful generation is signaled by the boolean return value.

Table 38 — Signature of method isoIec23465_generateKeyPair()

API Name	isoIec23465_generateKeyPair()
API Return Value	**boolean** KeyGenerated
API Parameter(s)	**in struct** KeyInformation{ integer KeyIdentifier, string KeyName; string Oidlist[ ]; }
Exceptions	SecurityStatusNotSatisfiedException
	WrongParameterException

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. Usage of wrong parameters in the method generates the exception WrongParameterException.

10.4.11 SecretKey class

10.4.11.1 General

This class provides methods for symmetric key types and algorithms. Figure 14 depicts the dependency of the SecretKey class which extends the Key class (10.4.9).

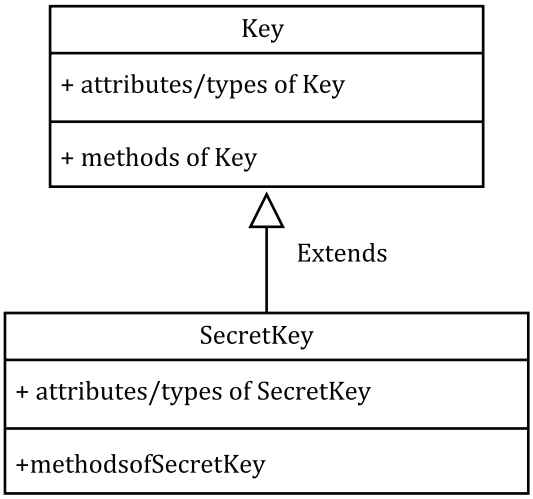

Figure 14 — SecretKey class

Methods of the SecretKey class are outlined in subclause 10.5, describing the different cryptographic key usage methods in detail.

10.4.11.2 Key Attributes

In addition to the common key attributes for some key types, supplemental attributes are specified. Table 39 lists possible additional attributes which determine the usage of the key.

Table 39 — Secret key attributes

Key attribute	Data type	Meaning
b_encrypt	boolean	Flag indicating an encryption key
b_decrypt	boolean	Flag indicating a decryption key
b_sign	boolean	Flag indicating a key supporting signature calculation
b_verify	boolean	Flag indicating a key supporting signature verification
b_wrap	boolean	Flag indicating a key supporting wrapping keys
b_unwrap	boolean	Flag indicating a key supporting unwrapping keys
keycheckvalue	octet[]	Key checksum

10.4.12 PublicKey

10.4.12.1 General

This class describes the public key type used for cryptographic operations. Class PublicKey, as shown in Figure 15, extends AsymmetricKey class (see 10.4.10). There is also an aggregation to CertificateContainer, dealing with public keys.

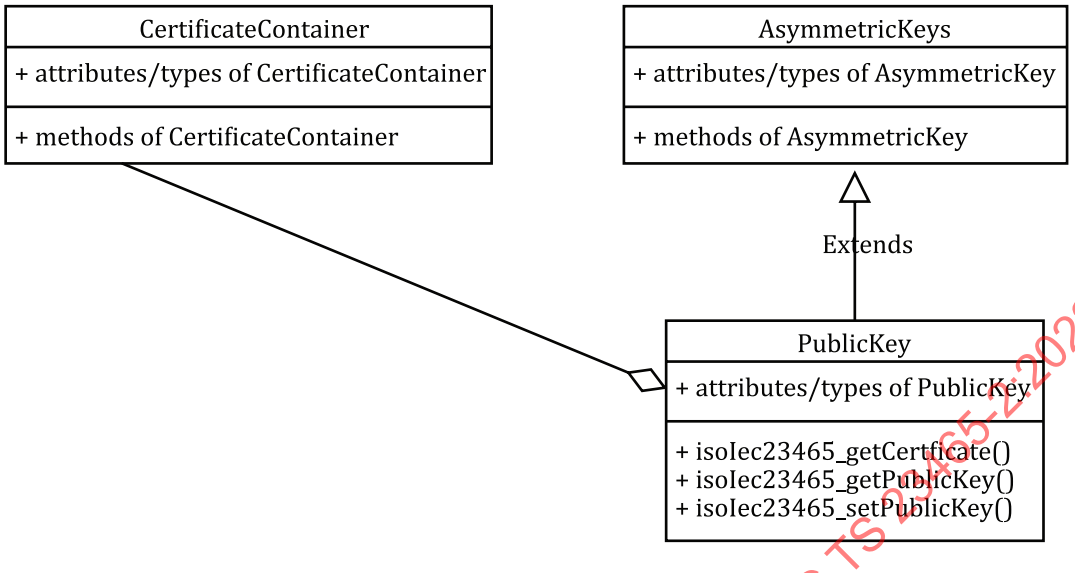

Figure 15 — PublicKey class

Further methods of the PublicKey class are outlined in subclause 10.5, describing the different cryptographic key usage methods in detail.

10.4.12.2 Method isoIec23465_getCertificate()

Table 40 shows the method description for retrieving the container storing a certificate.

Table 40 — Signature of method isoIec23465_getCertificate

API Name	isoIec23465_getCertificate()
API Return Value	***Object23465*** CertificateContainer
API Parameter(s)	—
Exceptions	ObjectNotFoundException
	SecurityStatusNotSatisfiedException

The method returns the CertificateContainer with the certificate for this public key.

The method throws ObjectNotFoundException if this container doesn't reference the issuer's certificate.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— ObjectNotFoundException.

10.4.12.3 Method isoIec23465_getPublicKey()

Table 41 shows the method description for retrieving the public part of an asymmetric key pair.

Table 41 — Signature of method isoIec23465_getPublicKey

API Name	isoIec23465_getPublicKey()
API Return Value	***struct*** PublicKeyComponents

Table 41 *(continued)*

API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentErasedException

The method returns the public part of an asymmetric key pair.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. A SensitiveContentErasedException is thrown if the key value is not (yet) set, e.g. the key value is neither generated inside the secure device nor personalized.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentErasedException.

10.4.12.4 Method isoIec23465_setPublicKey()

Table 42 shows the method description for setting the public part of an asymmetric key pair.

Table 42 — Signature of method isoIec23465_setPublicKey

API Name	isoIec23465_setPublicKey()
API Return Value	**boolean**

API Parameter(s)	**in struct** PublicKeyComponents{
	...;
	}

Exceptions	SecurityStatusNotSatisfiedException
	WrongParameterException

The method sets the public part of an asymmetric key pair. The key components are conveyed in the parameters of the method and shall be suited to the key type.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. If the structure of the key components does not suit to the key type, a WrongParameterException is thrown.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— WrongParameterException.

10.4.13 PrivateKey

10.4.13.1 General

This class describes the private key type used for cryptographic operations. Class PrivateKey as shown in Figure 16, extends the class AsymmetricKey.

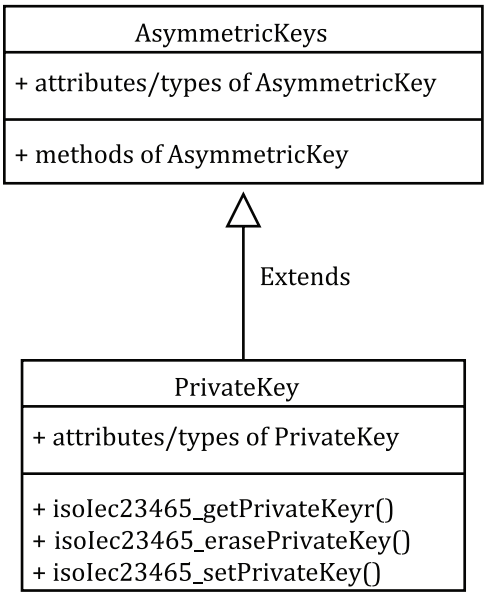

Figure 16 — PrivateKey class

Further methods of the PrivateKey class are outlined in clause 10.5, describing the different cryptographic key usage methods in detail.

10.4.13.2 Method isoIec23465_getPrivateKey()

Table 43 shows the method description for retrieving the private part of an asymmetric key pair.

Table 43 — Signature of method isoIec23465_getPrivateKey

API Name	isoIec23465_getPrivateKey()
API Return Value	**octet** PrivateKey[ ]
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException
	SensitiveContentErasedException

This method returns the private part of an asymmetric key pair.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled. A SensitiveContentErasedException is thrown, if the key value is not (yet) set, e.g. the key value is neither generated inside the secure device nor personalized.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException;

— SensitiveContentErasedException.

10.4.13.3 Method isoIec23465_setPrivateKey()

Table 44 shows the method description for setting the key material in the SensitiveContainer.

Table 44 — Signature of method isoIec23465_setPrivateKey

API Name	isoIec23465_setPrivateKey()

Table 44 *(continued)*

API Return Value	**boolean**
API Parameter(s)	**octet** PrivateKey[ ]
Exceptions	SecurityStatusNotSatisfiedException

This method is used to insert the private key, e.g. can possibly be used with the generation of a key pair.

The method is the counterpart to isoIec23465_erasePrivatKey().

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

In order to protect instance attributes from leaking, the priority of thrown exception shall be as follows (from high to low):

— SecurityStatusNotSatisfiedException.

10.4.13.4 Method isoIec23465_erasePrivateKey()

Table 45 shows the method description for wiping the content of body.

Table 45 — Signature of method isoIec23465_erasePrivateKey

API Name	isoIec23465_erasePrivateKey()
API Return Value	**void**
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method erases sensitive content containing the private key. All octets of key material are set to '00' and the object is in status erased until a new key is generated by isoIec23465_generateKeyPair() or isoIec23465_setPrivateKey() which is the counterpart of this method.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.5 API functions for cryptographic operation

10.5.1 General

Since the security device per definition stores secrets securely and offers reliable cryptographic services to the external world the isoIec23465-API offers dedicated functionality of the security device's applications, usable when implemented in the security device. In addition to the availability of crypto-functionality in the security device, the related data of the appropriate key material and the access rights to perform the functions have to be in place. The API functions for possible crypto functionality defined in this document are outlined in Table 46.

NOTE Further functionality and additional cryptographic support can extend Table 46 in the future.

Table 46 — Cryptographic API requirement

Cryptographic operation	Variant of functionality	Possible required parameters
Encryption	EncryptInit;	integer CipherKeyReference; octet OID_CipherAlgorithmAndPaddingFormat[]; octet InitialCipherValue[];
	Encrypt;	octet InputAndOutputData[]; integer NumberOfDdataToBeProcessed; boolean ToBeContinued;
Decryption	DecryptInit;	integer DecipherKeyReference; octet OID_DecipherAlgorithmAndPaddingFormat[]; octet Initial cipher value[];
	Decrypt;	octet InputAndOutputData[]; integer NumberOfDataToBeProcessed; boolean ToBeContinued;
Signing and MACing	SignInit;	octet OID_SignatureAlgorithmAndMessageDigest[]; integer SignatureKeyReference;
	Sign;	octet Data[]; integer NumberDataToBeProcess; boolean ToBeContinued;
Verifying digital signatures and MACs	VerifyInit;	octet OID_SignatureAlgorithmAndMessageDigest[]; integer SignatureKeyReference;
	Verify;	octet Data[]; integer NumberDataToBeProcess; boolean ToBeContinued;
Key management	GenerateKeyInstance;	integer KeyIdentifier; octet Key_Name[]; integer NumberKeyNameLength; struct KeyAttributes [ ]
	GenerateKeyPair;	Object23465 KeyReference; struct PrivateKey SpecificPrivateKey struct PublicKey SpecificPublicKey
	WrapKey;	Object23465 KeyToBeWrapped; Object WrappingKey; octet WrapperMechanism;
	UnwrapKey;	Object23465 WrappedKey; Object UnWrappingKey; octet WrapperMechanism;
	DeriveKey;	Object23465 BaseKey; octet DerivationData[] integer NumberDerivationData integer DerivationAlgorithm

Table 46 *(continued)*

Cryptographic operation	Variant of functionality	Possible required parameters
Random number generation	SeedRandom; GenerateRandom;	integer Random_Class integer Number_request_Random_Data octet RandomData[]
Message digesting	Digest;	integer MD_Algorithm; octet InputAndOutputData[ ]; integer NumberOfDataToBeProcessed; boolean ToBeContinued;
Key agreement	DiffieHellman Init	integer KeyAgreementAlgorithm integer ReferenceStaticPrivateKey integer ReferenceDomainParameterSet octet PublicKeyData[ ] integer NumberPublicKeyData
	DiffieHellman Agreed Shared Secret	octet Key_Agreement_Data[ ] integer NumberKeyAgreementData integer NumberSharedSecretData
Multi-step authentication protocols	PACE	octet CanMrz[ ]

Many of these functionalities require the usage of specific keys. In these cases, the appropriate functions of the API of this document are bound to the related key material. The methods that are assigned to specific key classes (SecretKey, PrivateKey, PublicKey) have to be implemented accordingly.

Some functions are not bound to a specific key and key usage. Examples are message digest or random number generation. Such functionality needs separated API methods independent from the key classes.

10.5.2 Extension of key class

10.5.2.1 General

In subclause 10.4.9 the key class was introduced. The class contains the list of OIDs defining the algorithms and processing modes the key is usable for and also the member function to get this OID-List. Additional information for keys is commonly used, which shall be also available at least usable at the application interface.

10.5.2.2 Common key attributes

The attributes of the class are extended to provide additional information for cryptographic operations for which the key is used. The common attributes described in Table 47 are usable for each key type. From the application's perspective the choice of keys, algorithms and mechanisms is essential for the usage of a complex security device.

Table 47 — Common key attributes

Attribute	Data type	Meaning
Key type	enumeration	private, public, secret_key
Key ID	integer	Unique key identifier
Key name	string	Key name for selection

Table 47 *(continued)*

Attribute	Data type	Meaning
Key mechanism list	struct	bit map definition for usability of the key (derivation, tbd.)

10.5.2.3 Method isoIec23465_getKeyAttributes()

Table 48 shows the method description for retrieving the common key attributes.

Table 48 — Signature of method isoIec23465_getKeyAttributes

API Name	isoIec23465_getKeyAttributes()
API Return Value	**struct** commonKeyAttribute (see Table 47)
API Parameter(s)	—
Exceptions	SecurityStatusNotSatisfiedException

This method allows the retrieval of the common key attributes for a specific key.

The method throws SecurityStatusNotSatisfiedException if the access condition for this operation is not fulfilled.

10.5.3 Interface methods related to keys

10.5.3.1 General

The outlined methods are assigned to the appropriate Key classes, specified in 10.4.9 to 10.4.12. Secret keys are used with symmetric key cryptography, e.g. DES or AES algorithm. Private and public keys are used in asymmetric cryptography, e.g. RSA or ECC.

The following definition of interface methods are related to cryptographic operations and most of them are bound to the appropriate keys, but not all are applicable to all key types.

10.5.3.2 Encipher/Decipher

10.5.3.2.1 Method isoIec23465_encryptInit()

This method provides access to the security device service which enciphers arrays of octets, conveyed by the application in the following isoIec23465_encrypt() functions. The API function as outlined in Table 49 conveys key information the object identifier for applied algorithms and padding methods and the initial value for the encryption process.

Table 49 — Signature of method isoIec23465_encryptInit

API Name	isoIec23465_encryptInit()
API Return Value	***boolean***
API Parameter(s)	***in integer*** CipherKeyReference; ***in octet*** OIDCipherAlgorithmAndPaddingFormat[ ]; ***in octet*** InitialCipherValue[ ];
Exceptions	SecurityStatusNotSatisfiedException
	KeyNotFoundException
	KeyUsageNotPermittedException
	WrongCommandParameterException

The method returns the success of getting the instance of the key storing the initial values and offering the subsequent encryption.

The SecurityStatusNotSatisfiedException applies when the the access condition for this operation is not fulfilled. If the key reference is invalid, the KeyNotfoundException occurs. In the case the OID is not suitable or not found, a KeyUsageNotPermittedExecption is thrown. If the Parameters are wrong or not complete the WrongCommandParameterException is returned.

10.5.3.2.2 Method isoIec23465_encrypt()

This method provides access to the security device service to encrypt arrays of octets, conveyed by the input parameters. The API function as outlined in Table 50 conveys the data in the data array, the number of data octets to be encrypted and the flag for further subsequent encryption calls. The encrypted data is returned in the same data array. The number of encrypted data summed up for all subsequent encryption calls are returned in return value of the method.

Table 50 — Signature of method isoIec23465_encrypt

API Name	isoIec23465_encrypt()
API Return Value	***integer*** NumberOfEncryptedData
API Parameter(s)	***in octet*** InputAndOutputData[]; ***in integer*** NumberOfDdataToBeProcessed; ***in boolean*** ToBeContinued;
Exceptions	SecurityStatusNotSatisfiedException
	KeyUsageNotInitializedException
	WrongCommandParameterException

With setting of the flag ToBeContinued=FALSE the method encrypts the last portion of the conveyed data and returns the complete number of all octets of all encrypted method calls and closes the encryption process with the deletion of the key information inserted by the encryptInit method.

The SecurityStatusNotSatisfiedException applies when the the access condition for this operation is not fulfilled. In the case the isoIec23465_encryptInit() was not performed, a KeyUsageNotInitializedException is thrown. If the Parameters are wrong or not complete, the WrongCommandParameterException is returned.

10.5.3.2.3 Method isoIec23465_decryptInit()

This method provides access to the security device service which deciphers arrays of octets, conveyed by the application in the following isoIec23465_decrypt() functions. The API function as outlined in Table 51 conveys key information, the object identifier for applied algorithm and padding method and the initial value for the decryption process.

Table 51 — Signature of method isoIec23465_decryptInit

API Name	isoIec23465_decryptInit()
API Return Value	***boolean***
API Parameter(s)	***in integer*** DecipherKeyReference; ***in octet*** OIDDecipherAlgorithmAndPaddingFormat[]; ***in octet*** InitialDecipherValue[];
Exceptions	SecurityStatusNotSatisfiedException
	KeyNotFoundException
	KeyUsageNotPermittedException
	WrongCommandParameterException

The method returns the success of getting the instance of the key storing the initial values and offering the subsequent decryption.

The SecurityStatusNotSatisfiedException applies when the the access condition for this operation is not fulfilled. If the key reference is invalid, a KeyNotfoundException occurs. In the case the OID is not suitable or not found, the KeyUsageNotPermittedException is thrown. If the Parameters are wrong or not complete, a WrongCommandParameterException is returned.

10.5.3.2.4 Method isoIec23465_decrypt()

This method provides access to the security device service to decrypt arrays of octets, conveyed in the input parameters. The API function as outlined in Table 52 conveys the data array, the number of data octets to be encrypted and the flag for further subsequent decryption method calls. The decrypted data are returned in the same data array. The number of decrypted data summed up for all subsequent decryption calls are returned in return value of the method.

Table 52 — Signature of method isoIec23465_decrypt

API Name	isoIec23465_decrypt()
API Return Value	***integer*** NumberOfDecryptedData
API Parameter(s)	***in octet*** InputAndOutputData[]; ***in integer*** NumberOfDdataToBeProcessed; ***in boolean*** ToBeContinued;
Exceptions	SecurityStatusNotSatisfiedException
	KeyUsageNotInitializedException
	WrongCommandParameterException

With setting of the flag ToBeContinued=FALSE the method decrypts the last portion of the conveyed data and returns the complete number of all octets of all decrypt method calls and closes the decryption process with the deletion of the key information inserted by the decryptInit method.

The SecurityStatusNotSatisfiedException applies when the the access condition for this operation is not fulfilled. In the case the isoIec23465_decryptInit() was not performed, a KeyUsageNotInitializedException is thrown. If the Parameters are wrong or not complete, the WrongCommandParameterException is returned.

10.5.3.3 Signing and MACing

10.5.3.3.1 Method isoIec23465_signInit()

This method provides access to the security device service, which calculates a digital signature for an array of octets conveyed by the application in the following isoIec23465_sign() functions. The API function as outlined in Table 53 conveys key information, the object identifier for the applied algorithm and message digesting method.

Table 53 — Signature of method isoIec23465_signInit

API Name	isoIec23465_signInit()
API Return Value	***boolean***
API Parameter(s)	***in octet*** OIDSignatureAlgorithmAndMessageDigest[]; ***in integer*** SignatureKeyReference;