

eEpoch

(eEurope Smart Card Charter proof of concept and  
holistic solution)

**Specification for the IAS IOP Interface  
based on  
Functional Mapping of GIF on ESIGN K**

**Schlumberger**

# IAS IOP Interface Specification

Document Control Sheet	
Responsible Author(s):	Mourad Faher
Organization:	Schlumberger
Subject / Title of Document:	IAS IOP Interface definition
Related Task('s):	
Deliverable No.	
Save Date of File:	21/05/2003 (initial .doc file)
Version Number:	1.1
Ref./File Name	IAS_IOP_Interface_v1.1.pdf
Number of Pages	25

Document Distribution			
Membertype	Organization	Name	Distributed
Web page	[Project Web Site]	Internet	DD/MM/YY
<b>Contractors / Partners</b>			
<b>European Commission</b>			
<b>Additional</b>			

## Contents

1.	Introduction.....	6
1.1	Revision log .....	6
1.2	References .....	6
1.3	Abbreviation.....	7
2.	Interfaces enumeration.....	9
3.	Interfaces definition .....	13
3.1	APDU External Interface.....	13
3.1.1	ISO/IEC and E-Sign K subset.....	13
3.1.2	Open Platform APDU Command subset .....	13
3.2	Open Platform APIs interface subset .....	14
3.2.1	Interface <i>openplatform.ProviderSecurityDomain</i> .....	14
3.2.2	Class <i>openplatform.OPSystem</i> .....	15
3.3	IAS interface .....	16
3.4	Java Card API.....	17
3.4.1.1	Commands dedicated to IDENTIFICATION.....	17
3.4.1.2	Commands dedicated to AUTHENTICATION.....	17
3.4.1.3	Commands dedicated to eSIGNATURE .....	17
	ANNEX A .....	19
	ANNEX B .....	24

## List of Figures

FIGURE 1: <i>SCHEMATIC REPRESENTATION OF THE JAVA CARD INTERFACES</i> .....	9
FIGURE 2: <i>SHARING MECHANISM, MULTI-INTERFACE PRINCIPLE</i> .....	24
FIGURE 3: <i>SHARING MECHANISM, SIO REQUESTING PROCESS</i> .....	24
FIGURE 4: <i>MAPPING OF IAS IOP ON JAVA CARD - IMPLEMENTATION APPROACH</i> .....	25
FIGURE 5: <i>OVERVIEW OF OPEN PLATFORM CARD ARCHITECTURE</i> .....	25

## List of Tables

TABLE 1: <i>REVISION LOG</i> .....	6
TABLE 2: <i>REQUIRED OP COMMANDS ACCORDING TO OP CARD SPEC 2.0.1' SECTION 9</i> .....	14

# 1. Introduction

## 1.1 Revision log

Version / Date	Brief Description of Changes, Name
0.1 / 05.02.03	Initial document, Mourad Faher
0.2/ 14.02.03	revision : R. Rosset ; F.Leprieur (SLB)
0.3/ 25.05.03	revision : M.Faher
0.4/ 30.06.03	revision : M.Faher, F.Leprieur
1.0/ 09.07.03	M.Faher
1.0/ 15.07.03	M.Faher – editorial corrections in Annex A
1.1/ 09.02.04	M.Faher – completion of the document

Table 1: *Revision log*

## 1.2 References

- [GIF, Part 3] Global Interoperability Framework for Identification, Authentication and Electronic Signature (IAS) with Smart cards, Version 0.96, August 2002, Smart Card Charter; Part 3 – Recommendation for IOP Specifications
- [ISO/IEC 7816] ISO/IEC FCD 7816, “Information technology – Identification cards- Integrated circuit(s) cards with contact”  
Part 4: Interindustry commands for interchange”, FCD2003  
Part 8: Security related interindustry commands, FCD 2003  
Part 11: Personal verification through biometric methods, FCD 2002  
Part 15: Cryptographic information application, FDIS 2003
- [ESD] Electronic Signature Directive 1999/93/EC
- [JCPS] Java Card Platform Security, Technical White Paper, Sun Microsystems
- [JCRE] Java Card 2.1 Runtime Environment (JCRE) Specification, Final Revision 1.0, February 24, 1999

[GDJC]	Guidelines for Developing Java Card Application on GlobalPlatform Cards, version 1.0, December 2002
[JCAPI]	Java Card 2.1.1 Application Programming Interface, Rev. 1.0, May 2000
[GPOP]	Open Platform Card Specification version 2.0.1', April 2000, GlobalPlatform
[ESIGNF]	ESIGN-F, CEN/ISSS WS/E-Sign Workshop Agreement Group F: Secure Signature-Creation Devices, CEN/ISSS N137, 1 <sup>st</sup> March 2001
[ESIGN]	Application Interface for Smartcards used as Secure Signature Creation Device, Version 1 Release 9, 17.09.2003, CEN/ISSS WS/E-Sign Group K Draft CWA; Part 1 – Basic Requirements
[IAS /E-SIGN K]	Generic IAS Application / System package based on E-Sign K, Version 1.0, Gieseke & Devrient
[CWA 14169]	CEN/ISSS Workshop Agreement – CWA 14169, March 2002, Secure Signature-Creation Devices “EAL4+”
[Mapping]	Functional Mapping of GIF on E-Sign K, version 1.3, 12.05.2003, Gieseke & Devrient
[LoadApplets]	LOAD of APPLETS protocol, Version 1.0, 14.05.2003, Schlumberger

### 1.3 Abbreviation

AID	Application Identifier
AUT	Authentication
C	Certificate
CA	Certification Authority
CRDO	Control Reference Data Object
CRT	Control Reference Template
DO	Data Object
DS	Digital Signature
FCI	File Control Information
FCP	File Control Parameters

FID	File Identifier
IAS	Identification, Authentication, electronic Signature
ICC	Integrated Circuit(s) Card
ICCSN	ICC Serial Number
IFD	Interface Device, e.g. terminal
IOP	Interoperability
MSE	Manage Security Environment command
NA	Non Applicable
PK	Public Key
PSO	Process Security Operation command
RND	Random number
SE	Security Environment
SigSK	Secret Key used for signing
SM	Secure Messaging
SK	Secret Key (equiv. to Private Key)
SN	Serial Number
SP	Service Provider

## 2. Interfaces enumeration

The main IAS functionality evoked in the GIF documents (§ [GIF, Part 3]) allow a description of the interfaces involved in the interaction between the IAS application and the on-card applications (also called GIF's subjects, services or identity token) as well as the off-card applications or Interface Device.

Among the functions defined in the GIF documents, some are suitable to outline the requirements for the IAS interfaces (see Figure 1 for a schematic representation of the JavaCard interfaces). Figure 1 is to be considered in accordance with Figure 5:

1. first, the Card Manager entity is the on-card representative of the Card Issuer on a Java Card Platform,
2. second, the architecture of the Card Manager provides an API to Applications, and it handles APDU command dispatch, application selection and card content management. For instance, when a SELECT command is received, the Card Manager sets the application referenced in the select command to be the selected application, and subsequent application commands shall be dispatched to the selected application. The Issuer Security Domain is part of the Card Manager.

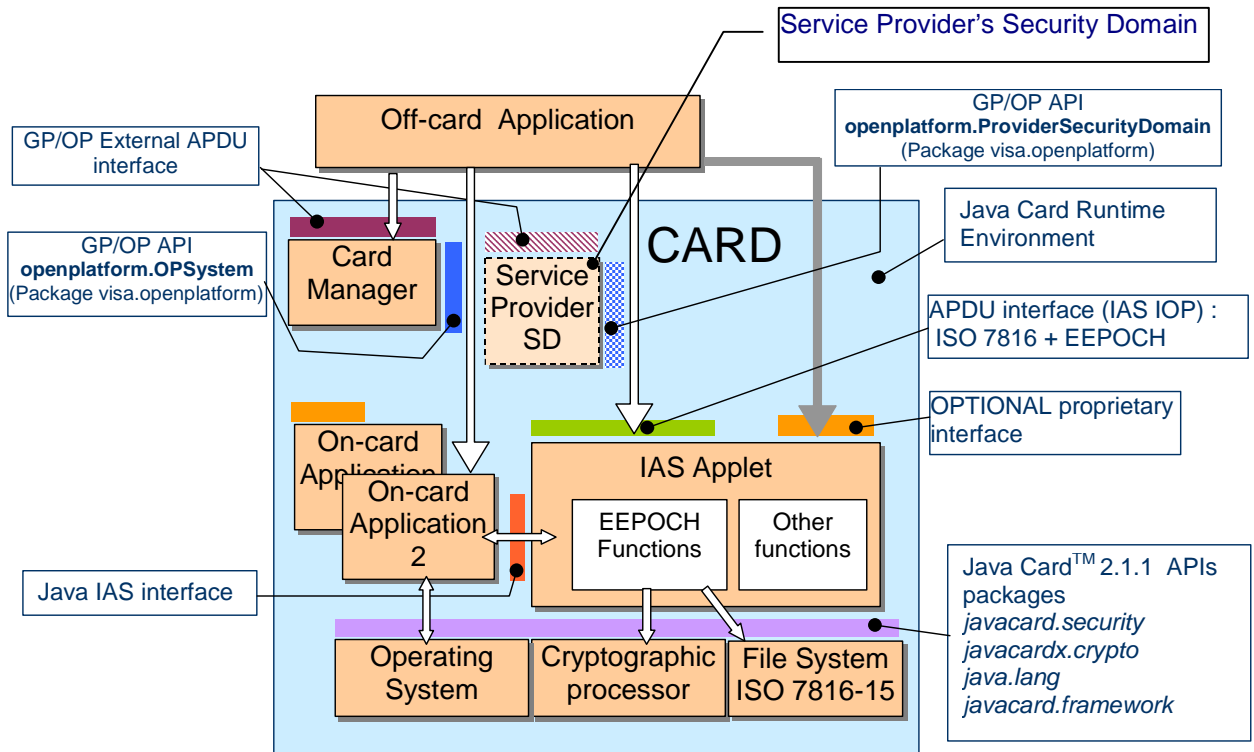


Figure 1: schematic representation of the Java Card Interfaces

GIF part 3 - §4.1.1 Generic IAS function, Architecture and role:

- The IAS application must be accessible to off-card applications through standardized invocation methods (APDUs / ISO 7816, Java oriented solutions for Java cards).

This definition corresponds to an **APDU external interface** laying on ISO/IEC 7816-4,8. On Java Card platforms, the APDU interface will also support some Global Platform Open Platform specific commands known as **Open Platform APDU commands** (§ [GPOP] section 9) of which the availability depends on the Card Life Cycle State.

GIF part 3 - §4.1.1 Generic IAS function, Architecture and role:

- The IAS application can be made accessible to other card applications under the assumption that the card's operating environment supports a secured inter-application communication protocol.

This definition may be related to the Java Card Runtime Environment that supports the inter-application communication protocol through the process called “Sharing Mechanism” such as the IAS application provides services as subcontractor for GIF's subjects. Those services being naturally Identification, Authentication and Signature. This interaction will hide from the on-card applets the underlying procedures of cryptographic computations and File System access and management, whereby alleviating the implementation of on-card applets. The inter-application communication mechanism will rely on the **Java Card Application Programming Interface (§ [JCAPI])**, and particularly the public interface *Shareable* of the package *javacard.framework*. In effect, the *Shareable* interface serves to identify all shared methods. Only the methods specified in a shareable interface are available through the firewall. Implementation classes (applets, or IAS application in this case) can implement any number of shareable interfaces and can extend other shareable implementation classes.

A graphical description of the sharing mechanism model is provided in Annex B (Figure 2, Figure 3, Figure 4).

As a simple example of this sharing usage, a Security Domain application may implement an interface that prototypes security and cryptographic methods, and that inherits from the *javacard.framework.Shareable* interface, which is part of the *javacard.framework* package. Therefore, the Security Domain will provide security and cryptographic services for Secure Messaging to the calling applets registered with this Security Domain. This is a kind of client/server interaction through the JCRE, the Security Domain being the server application.

The interface *Shareable* evoked in the previous examples is part of the **Java Card Application Programming Interface [JCAPI]**.

GIF part 3 - §4.1.1 Generic IAS function, Architecture and role:

- The IAS application must have access to the smart card's cryptographic and security libraries in order to be able to perform its tasks.

On a Java Card platform, the Java Card packages (§ [JC-API]) provide methods for the IAS Application to handle cryptographic and security libraries, by accessing the related low-level procedures through the *javacard.security* and the *javacardx.crypto* packages that are part of the **Java Card 2.1.1 API interface [JC-API]**. See Figure 1 of this document. Nevertheless, to remain compliant with GIF (§ [GIF, Part 3] section 4.1.4) that asserts that “*The IAS application uses security and cryptographic primitives provided by the smart card but does not control, manage or has any liability for the smart card security.*”, we will consider the Java Card 2.1.1 API interface usage for access to security and cryptographic functions as related to a low-level implementation and therefore out of the scope of the present document, and besides, we will consider that the settings and values modification related to the cryptographic library management are to be achieved through the External APDU interface.

GIF part 3 - §4.1.2 IAS and subjects

- IAS IOP Definition: a process through which the Smart Card provides services (Identification, Authentication, Signature) to an off-card application about any of the subjects managed .

The IAS interoperability definition according to GIF is too generic and is related to the use of the card such as different services may be operated by the IAS application.

With regards to the interfaces involved in EEPOCH and pictured on Figure 1, a Java Card hosting an IAS Application exposes the following interfaces :

- APDU External interface also called "IAS IOP interface": it is made of a subset of ISO7816 commands + OP/GP commands subset + EEPOCH-specific commands (like Tachograph command of kind “PSO: Hash of File”, or like GET DATA command with specific data field template to retrieve on-chip generated Key of 2048-bit)
- Java IAS interface : this is the entry point [`processIasService( )`] allowing Applets to address the IAS Application with APDU received from the outside world. Its use is based on the sharing mechanism.
- GP/OP API interface : based on openplatform packages that provide the interface *ProviderSecurityDomain* and the class *OPSystem*.
- Java Card API (v.2.1.1):
- OPTIONAL proprietary interface up to the issuer: however, it should be kept in mind that a platform certification, if any, will require the determination of the security target according to the boundary of the IAS Application (at least). So

the more extensions are made to the IAS interface, the more IAS-enabled cards are exposed to attacks.

### 3. Interfaces definition

#### 3.1 APDU External Interface

As mentioned previously, this interface encompasses a subset of ISO commands on which E-SIGN K relies, a subset of Global Platform OP commands and two EPOCH specific commands (GET DATA and PSO:HASH OF FILE). The PSO:HASH OF FILE is applied according to the Tachograph recommendation but is not considered mandatory.

##### 3.1.1 ISO/IEC and E-Sign K subset

The lists of macro-commands identified by GIF with their E-Sign K counterpart are identified in the document [Mapping]. E-SIGN K application being intended for the use on both native cards and Java Card Platform, those commands are part of the APDU external Interface and are implementing [ISO/IEC 7816].

NOTE : For interoperability purposes, the IFD should not need to know beforehand any of the cryptographic objects that are to be used in the service operated by the current application. The ICC is assumed to deliver to the IFD the references to the relevant cryptographic objects, or the reference to the applicable security environments. Through such information, the IFD would be able to determine the entire processing wherever the ICC has been issued, provided the ICC is IAS-enabled.

##### 3.1.2 Open Platform APDU Command subset

To support the model of an IAS Application on a Java Card Open Platform [GPOP], some among the requirements for Open Platform APDU commands may be added to the APDU external interface. The necessary commands that shall be supported by Security Domains are marked with a cross mark (X) in the table below. The role of the Card Manager in the load of Applet protocol is detailed in [LoadApplets] document. The Card Manager being likely to perform the load of Applets and to set up the Secure Messaging for this purpose, some among the commands required by [GPOP] are considered optional for Security Domains on IAS-enabled cards. In other words, a Security Domain is not expected to initialise a Secure Channel on IAS-enabled cards in the course of a transaction between the IAS Application and an off-card system. Regarding the DELETE and PUT KEY commands, IAS being an application it may be subject to deletion if DELETE command is allowed, and subject to key update if PUT KEY is allowed. This decision remains up to the Issuer.

COMMAND	Card Manager	Security Domain with Delegated Management	Security Domain with DAP or Mandated DAP	Security Domain without Delegated Management
DELETE	OPTIONAL	OPTIONAL		
GET DATA	X			
GET STATUS	X			

COMMAND	Card Manager	Security Domain with Delegated Management	Security Domain with DAP or Mandated DAP	Security Domain without Delegated Management
INSTALL	X	OPTIONAL		
LOAD	X	OPTIONAL		
PUT KEY	OPTIONAL	OPTIONAL	OPTIONAL	OPTIONAL
SELECT	X	OPTIONAL	OPTIONAL	OPTIONAL
INITIALIZE UPDATE	X			
EXTERNAL AUTHENTICATE	X			
SET STATUS	X	OPTIONAL	OPTIONAL	OPTIONAL

**Table 2:** Required OP Commands according to OP Card Spec 2.0.1' section 9

Legend of Table 2:

X : command support required according to [GPOP].  
Blank cell: Not supported command.  
OPTIONAL : optional command

### 3.2 Open Platform APIs interface subset

In the section *Interfaces enumeration* of this document an overview was given about the OP API Interface. [GPOP] provide the class *openplatform.OPSystem* and the interface *openplatform.ProviderSecurityDomain* Besides, few of the methods provided by the class *openplatform.OPSystem* are necessary for IAS-enabled cards. In particular, a method to return the life cycle state of the selected application may be useful: *OPSystem.getCardContentState ()*. Those required commands are listed below. In fact, the GP/OP 2.0.1' package called "openplatform" provides the interface "ProviderSecurityDomain" and the class "OPSystem".

#### 3.2.1 Interface *openplatform.ProviderSecurityDomain*

The *ProviderSecurityDomain* should extends the *javacard.framework.Shareable* class so that it may provide shareable interface objects (handle in other words) to client Applets that want to access cryptographic services namely for secure channel initialization and Secure Messaging purposes. But as the Secure Messaging according to GP/OP is based on session keys generated through a symmetric cryptography process, it is not applicable for the EEPOCH transactions that require session keys generated through an asymmetric cryptography process (according to E-SIGN K). The GP/OP Secure Messaging is only acceptable for Load of Applets because this operation is handled in a secure environment (Personalization Bureau for instance) and is intimately linked to OP cards. Therefore, on EEPOCH card, an on-card application is not expected to require the IAS Application, or its associated Security Domain if any, to initialize a secure channel following GP/OP specification. As the Load of Applet operation is carried out by the Card Manager according to [LoadApplets], the related Secure Messaging is established through APDU external interface (INITIALIZE UPDATE & EXTERNAL

AUTHENTICATE commands), and therefore the methods provided by the interface *ProviderSecurityDomain* for Secure Channel management are not necessary anymore for EEPOCH cards.

(When a Secure Messaging is required for a given transaction with EEPOCH card, it should be initialized by the IAS application according to EEPOCH protocol itself based on E-SIGN K asymmetric authentication scheme).

NOTE : according to EEPOCH (§ [LoadApplet] deliverable) and to GO/OP “*while the Card Manager is responsible for checking each DAP present in the Load File, it does not have the cryptographic keys or knowledge of the algorithms required to perform the actual verification. The Card Manager therefore sends the required information to the identified Security Domain, and it is this Security Domain’s responsibility to inform the Card Manager whether the DAP is valid or not.*”

This means that in case one rely on the DAP Verification mechanism to secure the Load of Applet operation, one need to store the DAP Verification Key (that may be RSA 1024-bit public key or Triple DES 16-bytes key) into a Security Domain. In effect, the IAS application is the core application that undertakes the security operations but it is not obliged to control and handle the keys dedicated to DAP verification because those keys remain Service Provider-specific and are not involved in Identification, Authentication or eSignature that are the main tasks imparted to the IAS application.

### 3.2.2 Class *openplatform.OPSystem*

The class *OPSystem*, provides methods for the Life Cycle State management. Those methods may be exposed by the IAS to on-card applications that are assumed to be entitled to execute sensitive commands modifying the life cycle state of applets and even the life cycle state of the Card Manager. The "getCardContentState( )" method is less risky than "terminateCardManager( )", "setCardContentState( )", or "lockCardManager ( )" for example. However, the GET STATUS and SET STATUS commands provide the same service at External APDU interface and may replace the *OPSystem* methods dedicated to the life cycle state management. Nevertheless, in case of PIN management, the *OPSystem* methods involved for this purpose ("getTriesRemaining( )", "verifyPin(APDU apdu, short offset)", "setPin( )") may be useful if they are exposed by the IAS application. But as ISO 7816-4 provides of course equivalent commands hereby VERIFY, CHANGE REFERENCE DATA and RESET RETRY COUNTER, one may dispense with the *OPSystem* methods (by the way, the PIN code is a cryptographic object that is to be stored and referred to according to PKCS#15 standard).

Therefore we may consider that all the operations that may be performed through the APDU interface (GP/OP APDU commands) and that are handled by the Card Manager would rather be addressed directly to the Card Manager instead of being required from the IAS through an API of kind *openplatform.OPSystem* or *openplatform.ProviderSecurityDomain*. Thus we maintain the unique entry point "ProcessIasService (APDU apdu)" to access IAS application services.

To execute method below, the Card Manager locates the AID of the selected application in the registry and returns the life cycle state.

Method Summary	
Static byte	<b>getCardContentState()</b> This method returns the Life Cycle State of the current applet context.

### 3.3 IAS interface

As said previously (§ section 2 Interfaces enumeration), the IAS Application is accessible to the off-card application through two channels, of which the first is the APDU external interface, and the second is the inter-application communication channel.

The public IAS interface serves to identify the IAS Application entry point (see the declaration in the *javadoc* (html files) associated with the present document.

This interface extends *javacard.framework.Shareable* interface. An IAS Application (server applet) shall define a class that implements this interface.

The unique method provided by this interface IAS is defined as follows:

**public void processIasService (APDU apdu)**

The advantages of determining a unique entry point for the IAS Application are:

- The client applet (providing the eService) will address directly ISO APDU commands to the IAS Application, those commands being in accordance with E-SIGN K subset specification.
- The client applet will play the role of a “pipe” conveying APDUs between the External APDU interface and the IAS Application as shown in the sample code in Annex A of the present document. In this example the server applet (IAS Application) delivers a handle (sio) to the client applet. Through this handle, the client applet may forward to the IAS Application all the APDUs conducted to it by the Card Manager and received from the off-card system.
- The server applet (IAS Application) may also instantiate three different classes, each of which implementing a dedicated interface IAS, respectively for Identification, Authentication, or eSignature purpose. The server applet may deliver the appropriate SIO (Shareable Interface Object) to the calling client applet, according to the parameter conveyed with the call for SIO (see Figure 2 and Figure 3). Once the client applet disposes of the SIO handle, it may call the associated *processIasService (APDU apdu)* method, and the instance will check the APDU instruction parameter value (INS) to determine the action to be performed to achieve the service. The client applet will therefore call the same method with a sequence of APDUs according to the specifications defined in [IAS /E-SIGN K].

NOTE : when an Applet embodies a service on card and relies on the IAS Application, it may need to get back data from the IAS Application. Nevertheless, the IAS is expected to return a shareable interface object to the client applet so that the applet may execute

allowed methods, but the IAS application is not expected to pass objects themselves for the sake of firewall protection between applications. Nevertheless, in Java Card, passing objects as parameters or return values in a shareable interface method is allowed in a very limited way through JCRE entry points: JCRE temporary entry point like APDU buffer may be accessed from any context and used to return values from the server applet to the client applet.

### 3.4 Java Card API

Hereafter is listed an API subset that is necessary to support the commands addressing the IAS Application (the method between bracket is part of GP/OP API). Commands details are provided in [JCAPI].

#### 3.4.1.1 Commands dedicated to IDENTIFICATION

For purposes of PIN verification:

- ❑ APDU.getBuffer()
- ❑ APDU.setIncommingAndReceive()
- ❑ [ [OPSystem.getCardContentState\(\)](#) ]
- ❑ Util.getShort()
- ❑ Util.setShort()
- ❑ OwnerPIN.check()
- ❑ OwnerPIN.getTriesRemaining()
- ❑ ISOException.throwIt()

#### 3.4.1.2 Commands dedicated to AUTHENTICATION

For purposes of External Authentication for instance:

- ❑ APDU.getBuffer()
- ❑ APDU.setIncommingAndReceive()
- ❑ [ [OPSystem.getCardContentState\(\)](#) ]
- ❑ Util.getShort()
- ❑ Util.setShort()
- ❑ OwnerPIN.isValidated()
- ❑ MessageDigest.reset()
- ❑ MessageDigest.update()
- ❑ MessageDigest.doFinal()
- ❑ Cipher.init()
- ❑ Cipher.doFinal()
- ❑ DESKey.setKey()
- ❑ ISOException.throwIt()

#### 3.4.1.3 Commands dedicated to eSIGNATURE

For purposes of the PSO: Compute Digital Signature for instance

- ❑ APDU.getBuffer()
- ❑ APDU.setIncommingAndReceive()
- ❑ [ [OPSystem.getCardContentState\(\)](#) ]
- ❑ Util.getShort()
- ❑ Util.setShort()

- ❑ Cipher.doFinal()
- ❑ Cipher.init() [used in relation with MSE:SET]
- ❑ Signature.sign
- ❑ Signature.init() [used in relation with MSE:SET]
- ❑ ISOException.throwIt()

## ANNEX A

### Implementation sample

Client applet:

```
#####
```

```
// An administration is assumed to be the Service Provider  
package gov.admin.eVote;
```

```
import javacard.framework.*;  
import gov.eservice.ias.IAS;
```

```
//this client applet is dedicated to eVote service
```

```
public class VoteApp extends Applet {
```

```
    //declare the handle
```

```
    IAS sio;
```

```
    //value of the IAS_AID as defined for ESIGN application [A0 00 00 01 67]
```

```
    private byte[] aid_of_ias = SERVER_AID;
```

```
    //optional secret value that may be exploited to indicate to the IAS server
```

```
    //application the kind of action to perform.
```

```
    final byte id_param = SECRET_VALUE;
```

```
public void process(APDU apdu) {  
    byte[] buffer = apdu.getBuffer();
```

```
    //return if the APDU is the Applet SELECT command
```

```
    if (selectingApplet())
```

```
        return;
```

```
    //call the IAS application
```

```
    sio.processIasService(apdu)
```

```
}
```

```
public static void install() {
```

```
    //not detailed here
```

```
}
```

```
private VoteApp() {
```

```
    //constructor not detailed here
```

```
    ...
```

```
    register();
```

```
}
```

```
public void select(){
```

```
//gets the server AID object
    AID ias_aid = JCSystem.lookupAID(aid_of_ias, (short)0, (byte)aid_of_ias.length);
    if(ias_aid == null) ISOException.throwIt (SW_IAS_APP_NOT_EXIST);

    //requests the shareable interface object from the IAS application, the "param"
    //being the action indicator for example
    sio = (IAS)(JCSystem.getAppletShareableInterfaceObject(ias_aid, id_param));

    if( sio == null) ISOException.throwIt (SW_NO_SIO_AVAILABLE);
}

public void deselect(){
//not detailed here
}

} //VoteApp
```

#####

Server Applet (IAS Application) :

#####

*//interface*

```
package gov.eservices.ias;
import javacard.framework.Shareable;
```

```
//this interface encompasses all the instruction parameters (INS) that are likely to be
//used within the commands addressed to the IAS application. Those commands are
//shared into two categories depending on their CLA value: either standard or
//proprietary/GPOP commands.
//The list of instruction codes within the shareable interface is likely to provide a better
//visibility of the set of commands that may be passed to the IAS Application.
```

**public interface IAS extends Shareable {**

*//the instructions*

```
public final static byte INS_VERIFY = (byte)0x20;
public final static byte INS_CHG_REFDATA = (byte)0x24;
public final static byte INS_RETRY_COUNTER = (byte)0x2C;
public final static byte INS_GET_DATA = (byte)0xCA;
public final static byte INS_PSO = (byte)0x2A;
public final static byte INS_MSE = (byte)0x22;
public final static byte INS_READ_BINARY = (byte)0xB0;
public final static byte INS_INT_AUTH = (byte)0x88;
public final static byte INS_GET_CHAL = (byte)0x84;
public final static byte INS_EXT_AUTH = (byte)0x82;
public final static byte INS_SELECT = (byte)0XA4;
public final static byte INS_GEN_AKP = (byte)0X46;
```

```
public final static byte INS_GEN_AKP_ = (byte)0X47;
public final static byte INS_WRITE_CERTIF = (byte)0XD0;
public final static byte INS_EXPORT_KEY = (byte)0xCB;
public final static byte INS_PSO_HASH_FILE = (byte)0x2A;

//instructions for GPOP commands hereafter are not part of
//the commands used by the Client Applet to provide the service.
//These commands are reserved for Applet Loading and personalization
//procedures. Therefore the corresponding instructions are not included
//in the IAS interface definition.
/*
public final static byte INS_OP_INSTALL = (byte)0xE6;
public final static byte INS_OP_LOAD = (byte)0xE8;
public final static byte INS_OP_PUT_KEY = (byte)0xD8;
public final static byte INS_OP_INIT_UPDATE = (byte)0x50;
public final static byte INS_OP_EXT_AUTH = (byte)0x82;
public final static byte INS_OP_SET_STATUS = (byte)0xF0;
*/

//the entry point
public void processIasService(APDU apdu);

}

//server applet IAS
package gov.eservice.ias;
import javacard.framework.*;

public class IASApp extends Applet implements IAS {

//it's assumed that the IAS knows AIDs of the Applets allowed to access the IAS
//Application. Those Applet AID values are initialized during personalization.
//through those AIDs, the Issuer may have a further secure way to control the IAS
//service access.

final private byte[] Vote = aid_of_clientApplet;
final private byte SECRET_VALUE = secret_value;

public Shareable getShareableInterfaceObject (AID client_aid, byte param)
{
//may optionally identify the client Applet. In fact, the Applet has been already
//authenticated during the Loading process. But this control may prevent a rogue applet
//to require the IAS. The scheme of control will depend on the business model of course.

if (client_aid.equals(aid_of_clientApplet, (short)0, (byte)(aid_of_clientApplet.length)) ==
false)
return null;

//examine the matching with the secret value to further authenticate the client Applet,
```

```
//this optional one-byte parameter may serve to indicate a specific action to do.

/*
    if (param != SECRET_VALUE)
        return null;
*/

    // then provide the SIO
    return this;

} //getShareableInterfaceObject

// the entry point
public void processIasService(APDU apdu) {
    byte[] bAPDUBuffer;

    //get the Client Applet's AID in order to find out the actual caller
    // and to check whether the SIO has been shared illegally with a third non authorized
    //Applet.

    AID client = JCSysystem.getPreviousContextAID();

    if (client.equals(aid_of_clientApplet, (short)0, (byte)aid_of_clientApplet.length)) == false)
        ISOException.throwIt (SW_UNKNOWN_CLIENT);

    bAPDUBuffer = apdu.getBuffer();

    //check optionally the CLA byte
    //we can rather check directly the INS value
    if (bAPDUBuffer [ ISO7816.OFFSET_CLA ] == (byte)0x00)
    {
        //check the INS byte for standard commands
        switch(bAPDUBuffer [ ISO7816.OFFSET_INS ])
        {
            case INS_VERIFY:
                { //execute the corresponding procedure
                    ...
                    break;
                }
            ...
            case INS_MSE:
                { processMSE(apdu)
                    break;
                }
            ...
            case INS_WRITE_CERTIF:
                {
                    break;
                }
        }
    }
}
```

```
        Default:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }

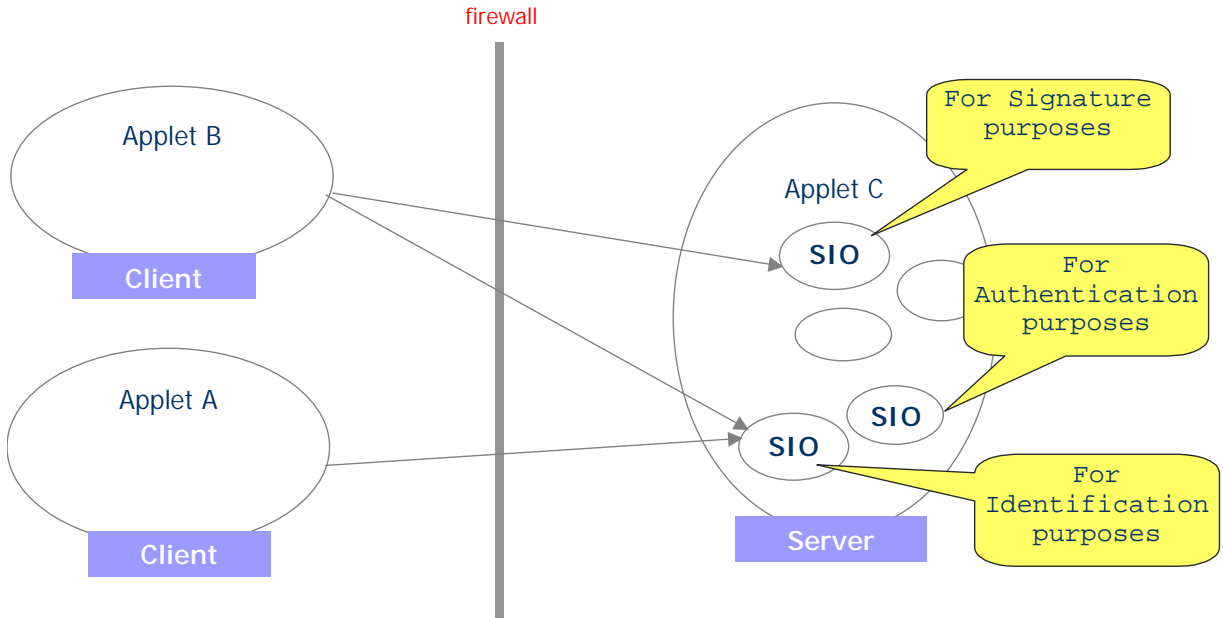
}else if (bAPDUBuffer [ ISO7816.OFFSET_CLA ] == (byte)0x80 || bAPDUBuffer [
ISO7816.OFFSET_CLA ] == (byte)0x84)
{
    //check the INS byte for GOP or proprietary commands
    switch(bAPDUBuffer [ ISO7816.OFFSET_INS ])
    {
        case INS_OP_INSTALL:
        { //execute the corresponding procedure
        ...
        break;
        }
        case INS_OP_INIT_UPDATE:
        { //execute the corresponding procedure
        ...
        break;
        }
        ...
        case INS_PSO_HASH_FILE:
        { processPSOHashFile(apdu)
        ...
        break;
        }
        Default:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}

}else{
    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
}
}

} //processIasService

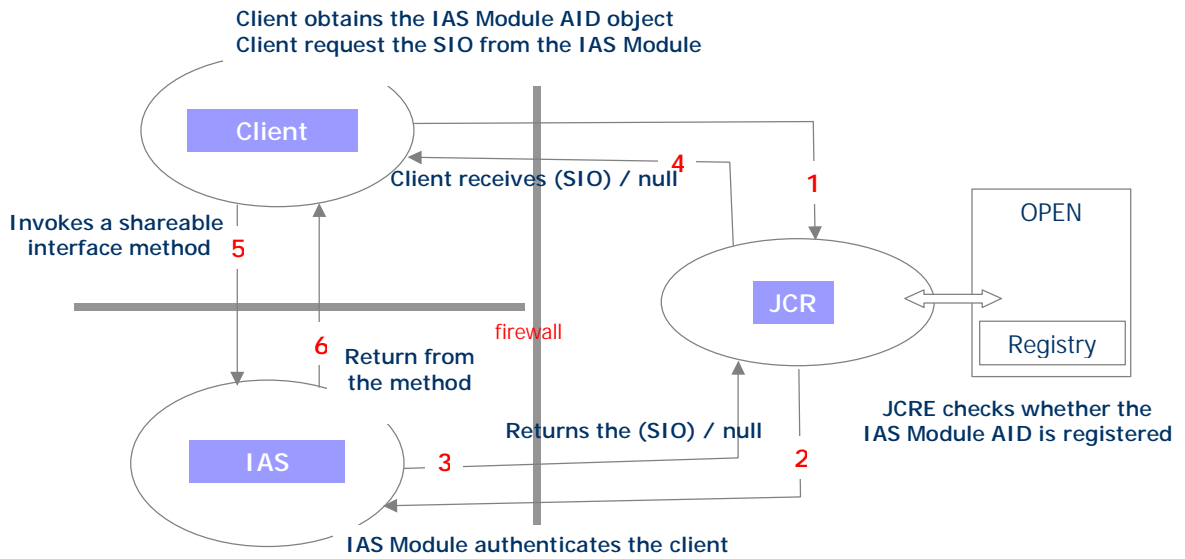
//procedures corresponding respectively to each INS
processMSE(APDU apdu)
{
...
}
...
processPSOHashFile(APDU apdu)
{
...
}
...
} //IASApp
```

## ANNEX B



A server applet can define multiple interfaces, each interface declaring

**Figure 2:** sharing mechanism, multi-interface principle



**Figure 3:** sharing mechanism, SIO requesting process

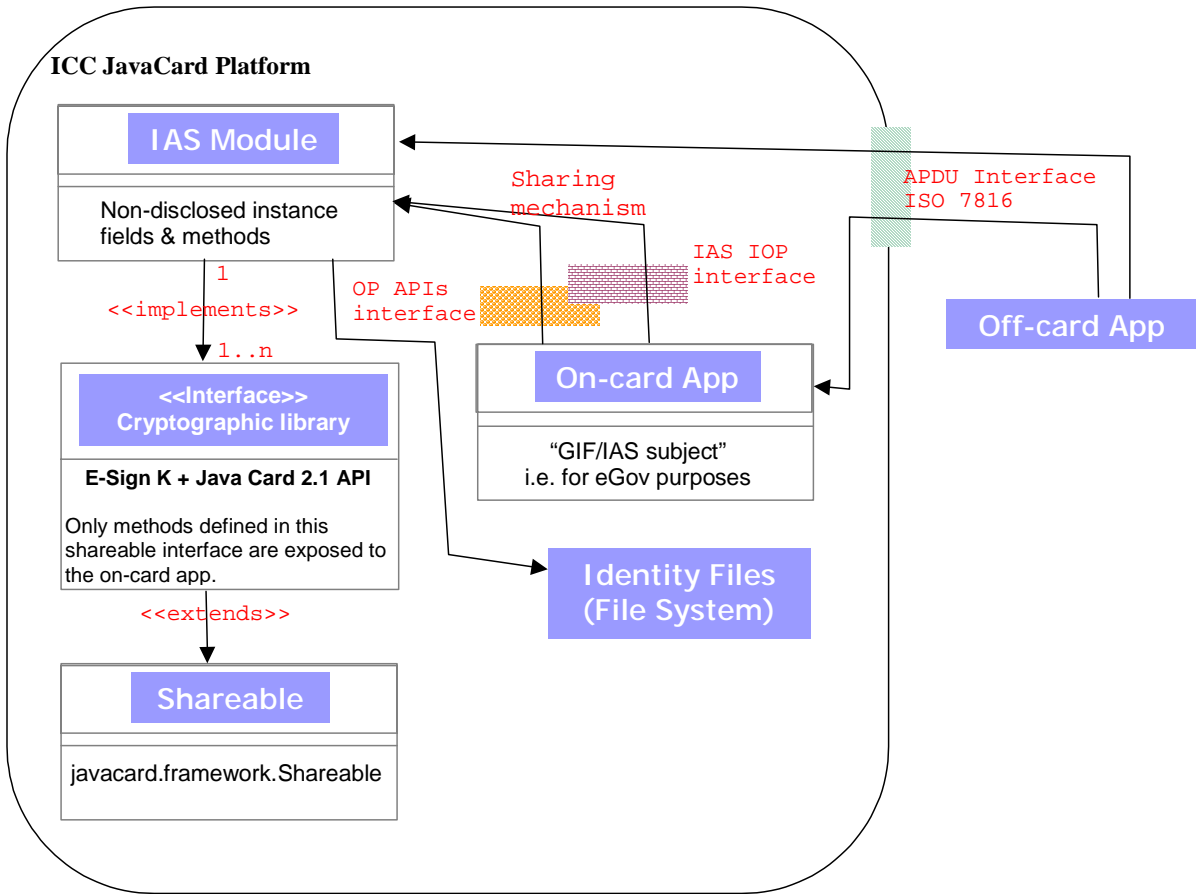


Figure 4: mapping of IAS IOP on Java Card - implementation approach

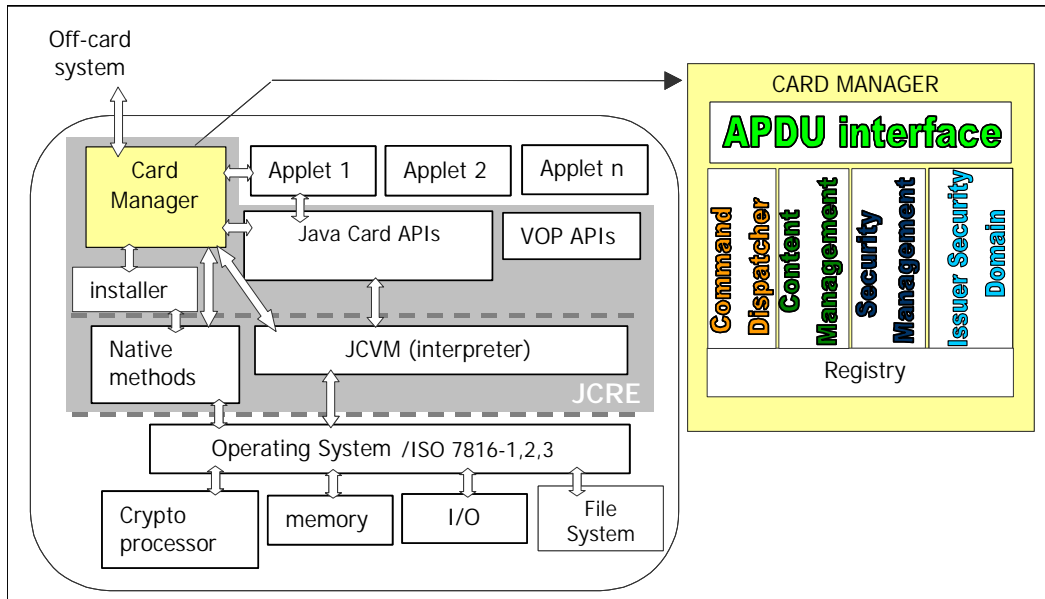


Figure 5: Overview of Open Platform Card Architecture