Government Smart Card Interoperability Specification
Contract Modification

August 29, 2000
Government Smart Cards Interoperability Specification

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modification Overview</td>
<td>iii</td>
</tr>
<tr>
<td>Architecture Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.0 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2.0 Architectural Components</td>
<td>1</td>
</tr>
<tr>
<td>2.1 Service Provider Module</td>
<td>1</td>
</tr>
<tr>
<td>2.2 Basic Services</td>
<td>1</td>
</tr>
<tr>
<td>2.3 Extended Services</td>
<td>1</td>
</tr>
<tr>
<td>3.0 Requirements and Constraints</td>
<td>2</td>
</tr>
<tr>
<td>3.1 Interoperability Requirements</td>
<td>2</td>
</tr>
<tr>
<td>3.2 Application Requirements</td>
<td>3</td>
</tr>
<tr>
<td>3.2.1 Cardholder Identification andAuthentication</td>
<td>3</td>
</tr>
<tr>
<td>3.2.2 Cryptography</td>
<td>4</td>
</tr>
<tr>
<td>3.2.3 Minimal Conformant Implementation</td>
<td>4</td>
</tr>
<tr>
<td>4.0 Interface Definitions</td>
<td>4</td>
</tr>
<tr>
<td>4.1 Basic Services Interface</td>
<td>4</td>
</tr>
<tr>
<td>4.2 Card Edge Presentation</td>
<td>5</td>
</tr>
<tr>
<td>4.3 Card Edge for File System Cards</td>
<td>5</td>
</tr>
<tr>
<td>4.4 Card Edge for Virtual Machine Cards</td>
<td>6</td>
</tr>
<tr>
<td>4.5 SKI for VM Cards</td>
<td>6</td>
</tr>
<tr>
<td>4.6 PKI for VM Cards</td>
<td>6</td>
</tr>
<tr>
<td>5.0 SPM Test Requirements</td>
<td>7</td>
</tr>
<tr>
<td>Appendix A – Basic Services Interface</td>
<td>8</td>
</tr>
<tr>
<td>Appendix B – Card Edge Presentation</td>
<td>47</td>
</tr>
<tr>
<td>Appendix C – Card Edge for File System Cards</td>
<td>63</td>
</tr>
<tr>
<td>Appendix D – Card Edge for Virtual Machine Cards</td>
<td>67</td>
</tr>
<tr>
<td>Appendix E – SKI for VM Cards</td>
<td>80</td>
</tr>
<tr>
<td>Appendix F – PKI for VM Cards</td>
<td>91</td>
</tr>
<tr>
<td>Appendix G – SPM Test Requirements</td>
<td>103</td>
</tr>
</tbody>
</table>
Modification Overview

Overview

This modification contains the interoperability specification as required in GSA contract resulting from Solicitation Number GS-TFF-99-203, Section H.24 “Government Smart Card Interoperability.” The specifications contained in this modification were developed with the assistance of the contract awardees and the Government. They establish technical standards for government smart card interoperability. This modification contains; an architectural model, interface definitions, conformance testing requirements, and the revised Section J.8 (GSA Common Data Model) data elements of the contract. All products and services procured under this contract must comply with these interoperability requirements and technical specifications unless otherwise specifically requested by the customer at the time of task order.

Interoperability Specification Updates and Version Control

GSA considers the interoperability technical specifications included in this modification to be the specification’s first version (version 1.0). As such, GSA anticipates revisions to these technical specifications as improvements and developments occur with smart cards. All products and services procured with this contract are required to conform to the most current specification version at the time of task order award.

All future changes and versions will comply with all government smart card interoperability requirements within Section H.24 of this contract and be backward compatible. Future specifications and changes will be developed with the assistance of the contract awardees and the Government. GSA will publish and amend this contract for all future specification versions.
1.0 Introduction

This section outlines the Interoperability Specification Architecture described in GSA’s Common Access Identification contract. The interoperability specification consists of the architectural components and a series of associated interface specification documents.

There are many different types of cards, readers, and software in the marketplace. This architecture identifies those components necessary for Government smart card interoperability. This is achieved by defining a model for smart card Service Provider Modules that present a standard Basic Services Interface to all client applications.

2.0 Architectural Components

2.1 Service Provider Module

All GSC systems contain a GSC Service Provider Module (SPM) consisting of cards, card readers, and driver software (Fig. 1). The purpose of an SPM is to provide card related services and functions to client applications through a set of standard interfaces. SPMs will be based on the smart card protocol stacks that currently exist on major platforms. The host-side software component of an SPM is referred to as the Service Provider Software (SPS).

2.2 Basic Services

The card related services that support logical access control, physical access control, cryptography, and biometrics must be interoperable. This is accomplished by defining a set of basic services and a corresponding Basic Services Interface (BSI) that are common to all SPMs. The services provided by the BSI are general enough to support a wide range of applications.

2.3 Extended Services

The GSC Architectural Model recognizes that many organizations will require additional card related services beyond those available through the BSI. This is accommodated through Extended Service Interfaces (XSIs). Any imaginable service can be implemented within an SPM and provided to client applications through an XSI. This will typically be done at the task order level, since extended services are designed to meet the application-specific requirements of a given organization.
3.0 Requirements and Constraints

3.1 Interoperability Requirements

All SPMs must implement the same BSI. SPMs are therefore interchangeable at the BSI level: client applications do not have to be rewritten when one SPM is replaced with another since all SPMs present the same service interface. All protocols and data formats for the interoperable card related services of the SPM are completely defined by the BSI specification.

The term “card-edge interface” refers to the set of command and response messages supported by a particular card or family of cards. In some cases, the protocols and associated card edge interfaces used to implement a particular card level service or
function vary widely across card families and manufacturers. If these differences cannot be resolved within the SPS, the GSC interoperability requirement dictates that the corresponding functionality cannot be included in the BSI. To do so would require “tunneling” different card-level protocols through the BSI. Client applications would need to be aware of which card they were communicating with, and would need to directly implement a set of card-level protocols.

For the reasons stated above, a number of card-level functions have been excluded from the BSI to preserve interoperability. Two principal examples are cryptographic key pair generation and the ability to change user PINs. However, many applications will require one or more of these non-interoperable functions. The GSC Architecture accommodates this through the concept of Extended Service Interfaces (XSIs). Application-specific functionality can be included in any SPM implementation by adding one or more XSIs to the SPM, and these interfaces can be defined in any way that suits the requirements of the application. Applications need to be modified to recognize and use specific XSIs, and the XSIs associated with different SPMs will vary. XSIs are at the same level of abstraction as the BSI, but cannot be defined at the level of the GSC-IS.

Even within the domain of interoperability defined for the BSI, services provided by different cards will vary. For example, depending on the card currently available to an SPM, public key cryptographic services might not be available at the card level. When a client application requests services from an SPM that cannot be provided by the card, the SPM will return a “Service Not Available” message.

3.2 Application Requirements

3.2.1 Cardholder Identification and Authentication

The GSC Architecture is specifically designed to provide card services that support logical and physical access control, biometric identification and authentication, and cryptography. Smart card systems typically provide identification and authentication services to logical and physical access control systems. The cryptographic functions of a card system are also associated with cardholder authentication, since challenge-response authentication protocols are based on cryptographic calculations using a secret or private key stored on the card.

From the card’s perspective, access control system clients require an exchange of information needed to identify or authenticate the cardholder. There are many ways to accomplish this exchange, ranging from simple PIN presentation to a cryptographic challenge-response authentication using a private or secret key stored on the card. The GSC-IS supports submission of a PIN to the card, transfer of static authentication data from the card to client applications, and cryptographic challenge-response protocols using either symmetric or asymmetric key algorithms.

Client applications can access the services of an SPM directly through it’s associated BSI. In some cases, applications may prefer to access SPM services through a
preexisting high level service interface (HLSI). In these cases the HLSI is essentially layered over the SPM, and the SPM becomes part of a service provider module associated with the HLSI. This is particularly true in the case of cryptographic services, since the major operating environments include native cryptographic service interfaces. This approach enhances interoperability, since legacy applications will already be configured to use these HLSIs.

3.2.2 Cryptography

The SPM model is designed to work with the Microsoft Crypto API, Sun’s Java Cryptography Architecture, and the Intel/Open Group Common Data Security Architecture. These cryptographic architectures are based on a client-server model, where client applications access Cryptographic Service Provider (CSP) modules through a high level Cryptographic Applications Programming Interface (CAPI). Other cryptographic service interfaces are easily accommodated by writing bridge code between the BSI and the target CAPI.

An SPM does not support all the services required of a typical CSP, and so it cannot act as a standalone CSP for the cryptographic architectures listed above. However, an SPM can be used to “smart card enable” a CSP implementation. This is done by integrating an SPM into a host CSP to provide card-related cryptographic services through a layered protocol stack. Client applications then access the services of the CSP through the high level CAPI, and the CSP communicates with the SPM through its BSI.

4.0 Interface Definitions

The GSC-IS package contains a set of interface definitions, standard data formats, and mechanisms for mapping heterogeneous smart card command sets to a common card edge interface. These specifications are defined in a series of six documents that are summarized below in subsections 4.1 through 4.6.

4.1 GSC: Interface Definition – Basic Services Interface

The BSI is the primary communications interface between client applications and SPMs. It is composed of three provider modules: a utility provider module, a generic container provider module, and a cryptographic provider module.

The utility provider module allows discovery of the reader, establishment of a logical connection with the card, and monitoring card status. Client applications can also communicate directly with cards using a pass-through function in the utility provider. The functions of the utility provider are not protected, and therefore no access control rules are enforced for these functions.

The generic container provider supports secure data storage services. Specifically, applications can create, delete, read, and update collections of {Tag, Length, Value} data items stored in containers on the card using the functions of this provider. A generic
container is a protected storage area on a smart card. It can be implemented as a file or as a buffer managed by a card applet, depending on the card type (filesystem or virtual machine). Access control rules are applied to the functions of the generic container provider, since generic containers are protected objects.

The cryptographic provider module provides random number operations, symmetric key infrastructure operations, and public key infrastructure operations. These operations are primarily used for cardholder authentication via cryptographic challenge-response protocols. Access control rules are applied to cryptographic provider module functions.

Detailed information is provided in Appendix A – Basic Services Interface.

4.2 GSC: Interface Definition – Card Edge Presentation

This specification achieves interoperability between SPS components and smart cards by defining a card capabilities discovery mechanism. Each card has a card capabilities container that maps the function set of the card to a common generic card-edge interface. This mapping mechanism allows an SPS to communicate with both file system cards and virtual machine cards. For file system cards, the card’s APDU set is mapped to a generic APDU-level interface. Virtual machine cards directly implement the predefined generic VM card interface modules described in subsections 4.3 through 4.5, and so the card-edge mapping process for VM cards is simply a matter of selecting the Application Identifier (AID) for the card applet that implements the functionality of the desired module.

The Card-Edge Interface Presentation document defines a card identifier object, a CardURL object to reference card services, an access control rules object, the card capabilities file grammar, and the associated generic implementation of an ISO 7816-4 APDU set for filesystem cards.

Detailed information is provided in Appendix B – Card Edge Presentation.

4.3 GSC: Interface Definition – Card Edge for File System Cards

This specification defines a generic ISO 7816-4 APDU set for file system (non-VM) cards. An SPS uses information from the card capabilities file to map the native APDU set of a file system card to the generic APDU set. This specification presents a common card edge interface for all file system cards to the SPS.

Detailed information is provided in Appendix C – Card Edge for File System Cards

4.4 GSC: Interface Definition – Card Edge for Virtual Machine Cards

This specification defines the APDU set of a card applet that provides card-level services to support the BSI generic container operations.
Detailed information is provided in Appendix D – Card Edge for Virtual Machine Cards.

4.5 GSC: Interface Definition – SKI for VM Cards

This specification defines the APDU set of a card applet that provides card-level services to support the BSI symmetric key cryptographic operations.

Detailed information is provided in Appendix E – SKI for VM Cards.

4.6 GSC: Interface Definition – PKI for VM Cards

This specification defines the APDU set of a card applet that provides card-level services to support the BSI public key cryptographic operations.

Detailed information is provided in Appendix F – PKI for VM Cards.
5.0 SPM Test Requirements and Procedures

This specification describes the requirements and procedures used to test SPM implementations for conformance to the GSC Interoperability Specification. Test procedures are defined for the modules of the BSI and the Card-Edge interfaces.

Detailed information is provided in Appendix G – SPM Test Requirements
Appendix A - GSC: Interface Definition - Basic Services Interface

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

?? A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI API presented in this document and amounts to the “top” part of an SPS.

?? A second level of interoperability is defined to allow smart cards to inter-operate at the card-edge interface. This level allows any SPS provider to inter-operate with any smart card that supports the card edge interface.

The goal of this document is to describe a simple set of Application Programming Interfaces covering the Basic Services for which the smart cards addressed by the GSA are required to inter-operate.

The Basic Services required are:
?? Secure storage and retrieval of the J.8 data set
?? smart card elementary cryptographic services

The focus of the system described is interoperability, as well as simplicity. The provider is stateless. It does not provide key management but focuses on allowing the enforcement of usage policies with their keys. It is the responsibility of the calling application to manage and protect the card access credentials.

The BSI relies on:
?? A Smart Card Utility provider Module.
This module abstracts a simple interface compatible with PC/SC and OCF. It allows also managing the communication with the smart card and the status of the card.
?? A Smart Card Secure Storage Services provider Module: the Generic Container Provider
This module abstracts the storage semantic of the smart card, and provides to Applications, with the concept of generic container, a simple Tag/value API.
?? A Smart Card Cryptographic provider Module
This module provides to applications and specialized middle-ware the essential cryptographic services that must be provided by the smart cards.

These modules constitute the software part of the SPM.

The functional scope of the provider is more generic than required so that the interoperability standard could be easily extended.
Compatibility
The Provider is compatible with any Smart Card in the following list:
?? JavaCard 2.1 or MultOS implementing the Card Edge Interface of the Common Access Card.
?? WpSC with its ISO7816-4 extension.
?? ISO7816-4 smart cards.

Limitations
The proposed BSI provider is an operational API and not a management API. It does not provide services like applet download or applet instantiation, or the creation of a file system. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation. Furthermore, PIN management functions like Change PIN or Unlock PIN are not part of this API.
The smart card is supposed to be already initialized: applets are downloaded, instantiated, and the file system created.
Establishing these limitations is balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:
- Is it a rare operation?
- Is it feasible or very difficult to define an interoperable method for this service?

The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the specification would have the result that the specification would not guaranty that all implementations would inter-operate.
Documents
The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.

The Government Smart Card Interoperability Standard
API Reference

Access Control Management
All smart card services exposed by the BSI API are submitted to operation specific access
control. A set of possible Access Control Rules can be defined for each service, at the
initialization of the smart card.
Each mModule of the provider offers a discovery function call that allows an application
to discover what is the set of Access Control Rules governing the access to the services
exposed by the module.

The Access Control Rules available are:
?? **Always**: the corresponding service can be provided without restrictions
?? **PIN protected**: the corresponding service can be provided only if its associated PIN
code has been already verified
?? **External authenticate**: the corresponding service can be provided only after a
get_challenge APDU.
?? **External Authenticate then PIN**: the two methods must be chained successfully
before access to the service is granted. This allows the authentication of the
Application and of the user.
?? **External Authenticate or PIN**: either one of the two control gives access to the
service. This allows for a CardHolder validation when a PIN pad is available and for
an external authentication when no PIN pad is available. Or, this provides an
Authentication method when the Application cannot be trusted to perform an external
Authentication and to protect the external authentication key.
?? **Secure Channel**: the corresponding service can be provided through a Secure
Channel managed by a card Secure Messaging layer that can be Open Platform, DIN,
the ISO7816-4 Secure Messaging.
?? **Never**: the corresponding service can never be provided.

As a reminder, the External Authentication method should comply with ISO7816-4. The
algorithm mandated is DES3-ECB, with a double length key-size 16 Bytes, and a
challenge of 8 Bytes.

For the External Authentication Access Control Rule, the provider provides two modes of
operation.

?? **The weak mode**: In this mode, the External authentication key is provided to the
provider in clear text, by the calling application, and the provider encapsulates the
external authentication transaction with the card or the applet. The clear text key is
provided in the Key member of the Authenticator structure defined hereafter.

?? **The strong mode**: If the Application has very high operational security requirements,
where it is not acceptable to expose the External Authentication Key [for example the
Application operates a Hardware Security Module (HSM)]. In this case, the following
mechanism is proposed: an explicit, provider level, get challenge functions will ask
the card or the applet to provide a challenge, which is returned to the Application.
The Application can then have the challenge encrypted by its Hardware Security Module and provides the resulting cryptogram in the AuthValue field of the Authenticator structure. This way, the authentication key will never leave the Hardware Security Module.

The Secure channel Access Control rule implies cryptographic operations performed at the APDU level (MAC). A pass-through call is provided to allow applications to create a secure channel and operate inside a secure channel.

Security Context
A function is provided that allows the Application to establish the Security Context required by the card by executing the Access Control Rule specific to a card service. This context must be released. For the Secure Channel Access Control rules, the Provider offers only a transport service, the calling application is entirely responsible for managing the security context.

Mandatory Cryptographic Algorithms
?? Algorithm Identifier “0x81”: DES3-ECB, with a double length key-size, 16 Bytes.¹
?? Algorithm Identifier “0xA3”: RSA_NO_PAD, computation on the private key, Chinese Remainder.
?? Algorithm Identifier “0x81”: DES3-CBC, with a double length key-size, 16 Bytes

¹ For detail description see “Open Platform, Card Specification, version 2.01’, April 7,2000”
Return Codes

Following is the list of the BSI API return codes. Codes returned by each function are listed with the function’s description later in this chapter.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSI_ACCESS_DENIED</td>
<td>Access conditions are not fulfilled</td>
</tr>
<tr>
<td>BSI_BAD_AID</td>
<td>The card application of the given AID cannot be found on the card</td>
</tr>
<tr>
<td>BSI_BAD_ALGO_ID</td>
<td>The algorithm ID provided to the cryptographic Provider is not supported.</td>
</tr>
<tr>
<td>BSI_BAD_AUTH</td>
<td>Authenticator value or type is not correct</td>
</tr>
<tr>
<td>BSI_BAD_HANDLE</td>
<td>Card communication handle is unknown</td>
</tr>
<tr>
<td>BSI_BAD_PARAM</td>
<td>Incorrect parameter value</td>
</tr>
<tr>
<td>BSI_CARD_ABSENT</td>
<td>There is no card in the reader</td>
</tr>
<tr>
<td>BSI_CARD_NOT_INIT</td>
<td>The smart card is not yet initialized</td>
</tr>
<tr>
<td>BSI_CARD_PRESENT</td>
<td>There is a card in the reader</td>
</tr>
<tr>
<td>BSI_CARD_REMOVED</td>
<td>The connected smart card has been removed</td>
</tr>
<tr>
<td>BSI_COMM_ERROR</td>
<td>Error during communication with the card</td>
</tr>
<tr>
<td>BSI_CREATE_ERROR</td>
<td>Error creating data in Generic Container</td>
</tr>
<tr>
<td>BSI_DATA_CORRUPTED</td>
<td>The provider has detected that the data read from the smart card is corrupted.</td>
</tr>
<tr>
<td>BSI_DELETE_ERROR</td>
<td>Error deleting data in Generic Container</td>
</tr>
<tr>
<td>BSI_INSUFFICIENT_BUFFER</td>
<td>The buffer provided to retrieve data is too small</td>
</tr>
<tr>
<td>BSI_LOAD_LIB_FAILED</td>
<td>Loading of card communication library failed</td>
</tr>
<tr>
<td>BSI_NO_MORE_DATA</td>
<td>No space available for data creation in container</td>
</tr>
<tr>
<td>BSI_NOT_IMPLEMENTED</td>
<td>The service is not implemented in the API version</td>
</tr>
<tr>
<td>BSI_OK</td>
<td>Execution completed successfully</td>
</tr>
<tr>
<td>BSI_PIN_LOCKED</td>
<td>The card is locked because too many wrong PIN have been entered.</td>
</tr>
<tr>
<td>BSI_READ_ERROR</td>
<td>Error reading data in Generic Container or getting a Certificate</td>
</tr>
<tr>
<td>BSI_SERVICE_NOT_AVAILABLE</td>
<td>The card does not implement a BSI service required by the Application.</td>
</tr>
<tr>
<td>BSI_UNKNOWN_ERROR</td>
<td>An error occurred but the cause is unknown</td>
</tr>
<tr>
<td>BSI_UPDATE_ERROR</td>
<td>Error updating data in Generic Container</td>
</tr>
<tr>
<td>BSI_ACR_NOT_AVAILABLE</td>
<td>The card or applet does not support the access control rule for which the application was attempting to establish a security context.</td>
</tr>
<tr>
<td>BSI_TAG_EXISTS</td>
<td>Attempt to duplicate an existing tag.</td>
</tr>
</tbody>
</table>
Access Control Rules

The following Access Control Rules can be requested for each BSI service. An exact matrix of each Access Control Rule supported within a given provider is defined in the provider specific chapter.

<table>
<thead>
<tr>
<th>Access Control Rules</th>
<th>Number of Authenticators</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSI_ACR_ALWAYS</td>
<td>0</td>
<td>No access control rule is required</td>
</tr>
<tr>
<td>BSI_ACR_NEVER</td>
<td>1</td>
<td>Operation is never possible</td>
</tr>
<tr>
<td>BSI_ACR_PIN</td>
<td>1</td>
<td>PIN code is required</td>
</tr>
<tr>
<td>BSI_ACR_XAUTH</td>
<td>1</td>
<td>External authentication, performed as a challenge/response is required</td>
</tr>
<tr>
<td>BSI_ACR_XAUTH_THEN_PIN</td>
<td>2</td>
<td>External Authentication followed by a PIN presentation</td>
</tr>
<tr>
<td>BSI_ACR_XAUTH_OR_PIN</td>
<td>1</td>
<td>The object method can be accessed either after an External Authentication or after a successful CHV</td>
</tr>
<tr>
<td>BSI_SECURE_CHANNEL_OP</td>
<td>0</td>
<td>The calling Application establishes and operates inside the passthru function</td>
</tr>
<tr>
<td>BSI_SECURE_CHANNEL_DIN</td>
<td>0</td>
<td>The calling Application establishes and operates inside the passthru function</td>
</tr>
<tr>
<td>BSI_SECURE_CHANNEL_ISO</td>
<td>0</td>
<td>The calling Application establishes and operates inside the passthru function</td>
</tr>
</tbody>
</table>

It is the responsibility of the calling application to provide the required Access Control Key or the Access Control Authenticator. The provider handles the implementation of the access control rule.

Authenticator structure

```c
typedef struct strBSI_Authenticator{
    unsigned int  unACRType;
    unsigned char* uszAuthValue[BSI_AUTHENTICATOR_MAX_LEN];
    unsigned int  unAuthLen;
    unsigned char* uszKey[BSI_KEY_LENGTH];
} BSIAuthenticator;
```

`unACRType` : access control rule.
`uszAuthValue` : authenticator value, can be the PIN code or the External Authentication cryptogram.
unAuthLen: authenticator value length.

UszKey: Authentication key. To be used in the provider “weak” mode.
The Authenticator must be provided as a parameter to all access calls of the provider.

?? Card handle
   #typedef int UTILCardHandle
   Card connection handle.

?? Tag
   #typedef unsigned char gcTag
   The Generic Container Provider operates on Tagged data, and thus relies on a
definition of what a TAG is.

?? Container size structure
   typedef struct strGC_ContainerSize{
     unsigned int unMaxNbDataItems,
     unsigned int unMaxValueStorageSize,
   } GCContainerSize;

   MaxNbDataItems: The maximum number of {Tag, Length, Value} data items
this instance of the Generic Container can hold. The provider is responsible for
calculating this value.

   UnMaxValueStorageSize: The size of the value storage area of the Container.

---

2 see document “gsc BSI internal TLV format specification”
GscBsiUtility Provider

Utility Provider Module function list
Allows the discovery of the reader, the establishment of a connection with the card, and
the monitoring of the card.

`gscBsiUtilGetCardStatus()`
Retrieves card presence for a connection handle or a reader.

`gscBsiUtilCardConnect()`
Connect to the card, using the reader name the card is inserted in.

`gscBsiUtilCardDisconnect()`
Disconnect to the card

`gscBsiUtilGetCardProperties()`
Retrieve card dependant data (Serial number)

`gscBsiGcGetContainerProperties()`
Retrieve the properties of the Container(s)

`gscBsiUtilGetVersion()`
Retrieve the version of the CG provider.

`gscBsiUtilPassthru()`
Allows to send an APDU to the card or the applet and get the answer from the card.

`gscBsiUtilAcquireContext()`
Establishes the Security Context required by the command, as discovered using the
appropriate get_properties function.

`gscBsiUtilReleaseContext()`
Releases the Security Context previously established.
Access Control Rules supported
Not Applicable. The operations of the Utility Provider are not protected. Login is not required.

gscBsiUtilGetCardStatus()

Purpose
Retrieves card status knowing whether the card is already connected.

Notes: Connection handle and reader parameters are exclusive.

Prototype

Int gscBsiUtilGetCardStatus (IN UTILCardHandle hCard);

Parameters

Hcard: Input parameter
Card connection handle, from gscBsiUtilCardConnect()

Return Codes

BSI_OK a connected card present for the handle
BSI_CARD_REMOVED Card removed
BSI_CARD_PRESENT A card is in the reader
BSI_CARD_ABSENT There is no card in the reader
gscBsiUtilCardConnect()

Purpose
Connect to the card.

Prototype
Int gscBsiUtilCardConnect(
    OUT UTILCardHandle* phCard);

Parameters
phCard : Output parameter
    Card connection handle.

Return Codes
BSI_OK                  Connection successful
BSI_BAD_PARAM           Bad parameter
gscBsiUtilCardDisconnect()

**Purpose**
Disconnected to the card.

**Prototype**
```c
Int gscBsiUtilCardDisconnect(
    IN UTILCardHandle hCard);
```

**Parameters**
- **HCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`.

**Return Codes**
- **BSI_OK**: Successful function
- **BSI_BAD_HANDLE**: Unknown card handle
gscBsiUtilGetCardProperties()

**Purpose**
Retrieves the version of the Card as well as the capability of the card. Usage is to use the call once with a NULL pointer to get the size required and then, to allocate the buffer to the proper size.

**Prototype**
```c
Int gscBsiUtilGetCardProperties(
    IN UTILCardHandle hCard,
    OUT unsigned char* puszCardUniqueID,
    IN/OUT unsigned int* punUniqueIDLength,
    OUT unsigned int unCardCapability
);
```

**Parameters**
- **puszCardUniqueID**: Output parameter
  - Pre-allocated buffer.
  - Version of the Provider: “major,minor,revision,build_number”

- **punUniqueIDLength**: Input / Output parameter
  - Length of the Card Unique ID string expected is provided as input.
  - Length of the exact length of the returned string is provided as output.

- **unCardCapability**: Output parameter
  - Bitwise mask defining the provider supported by the card:
    ```
    #define BSI_GCJ8 0x00000001
    #define BSI_GCJ8X 0x00000002
    #define BSI_SKI 0x00000004
    #define BSI_PKI 0x00000008
    ```

**Notes:**
- The version is a string ended by a ‘\0’ character

**Return Codes**
- **BSI_OK**: Function successful
- **BSI_BAD_PARAM**
- **BSI_INSUFFICIENT_BUFFER**
gscBsiGcGetContainerProperties()

**Purpose**
Retrieve the properties of the Container(s): Access Control Rules or read Tag list, read value and value operations on the container.
The Access Control Rules returned are common for all data items managed by the selected container.

**Prototype**
```c
int gscBsiGcGetContainerProperties(
    IN UTILCardHandle   hCard,
    IN unsigned char*  usAID,
    IN unsigned int    unAIDLen,
    OUT GCacr*         pstrACR,
    OUT GCContainerSize* pstrContainerSizes,
    OUT unsigned char* pszContainerVersion
    IN OUT unsigned int* pszContainerVersionLength
);```

**Parameters**
- **hCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`

- **usAID**: Input parameter
  GC AID value.

- **unAIDLen**: Input parameter
  GC AID value length.

- **pstrACR**: Output parameter
  Structure indicating all access conditions for operations. The caller must allocate the structure.

- **pstrContainerSizes**: Output parameter
  Structure indicating the sizes of the two storage areas of the container. The structure must be allocated by the caller.

- **pszContainerVersion**: Output parameter
  The version of the Generic Container.

- **pszContainerVersionLength**: Input/Output parameter
  The length of version of the Generic Container.

**Return Codes**
- **BSI_OK**: Successful function
- **BSI_BAD_AID**: There is no GC for this AID
- **BSI_BAD_PARAM**: One of the output access right pointer is NULL
gscBsiUtilGetVersion()

Purpose
Retrieves the version of the Provider

Prototype
Int gscBsiUtilGetVersion (  
    OUT unsigned char* puszVersion,  
    OUT unsigned int* punVersionLength);

Parameters
PuszVersion : Output parameter
Pre-allocated buffer.
Version of the Provider:
“major,minor,revision,build_number”

PunVersionLength : Input / Output parameter
Length of the version string expected is provided as input.
Length of the exact length of the returned string.

Notes:
The version is a string ended by a ‘\0’ character

Return Codes
BSI_OK Function successful
BSI_BAD_PARAM
BSI_INSUFFICIENT_BUFFER


gscBsiUtilPassthru()

**Purpose**
Allows to send an APDU to the card or the applet and get the answer from the card.

**Prototype**
```c
Int  gscBsiUtilPassthru(
    IN UTILCardHandle  hCard,
    IN unsigned char*  puszCardCommand,
    IN unsigned int    unCardCommandLen,
    OUT unsigned char* puszCardResponse,
    OUT unsigned int*  punCardResponseLen);
```

**Parameters**
- **hCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`

- **puszCardCommand**: Input parameter
  An APDU sent to the card by an Application. This allows an application to enforce APDU level security like a secure channel.

- **unCardCommandLen**: Input parameter
  Length of the APDU command sent to the card by an Application.

- **puszCardResponse**: Output parameter
  An APDU sent to the card by an Application. This allows an application to enforce APDU level security like a secure channel. The buffer must be pre-allocated by the caller. The function can be called twice to discover the exact size of the buffer to allocate. The first time it can be called with NULL, and will return the size of the buffer needed.

- **punCardResponseLen**: Output parameter
  Length of the APDU command returned to Application by the Card or applet. Must be allocated by the calling Application.

**Return Codes**
- **BSI_OK**: Function successful
BSI_BAD_PARAM
BSI_INSUFFICIENT_BUFFER the size of the allocated buffer is insufficient to contain the returned APDU.
gscBsiUtilAcquireContext()

In the so-called weak mode, the application will NULL the "uszAuthValue" and put the key in uszKey". This will tell the provider that the weak mode has been chosen and that it should perform the XAUTH transaction itself.

In strong mode, the acquire_context call was preceeded by a get_challenge, so the application computes the authenticator itself and puts it in "uszAuthValue", and NULLs "uszKey".

The principle is that the application knows the environment and the level of security it needs to apply much better than the middleware.

**Purpose**

Login to the Card using the appropriate Authenticator.

**Prototype**

Int gscBsiUtilAcquireContext (  
IN UTILCardHandle hCard,  
IN unsigned char* usAID,  
IN unsigned int unAIDLen,  
IN BSIAuthenticator* pstrAuthenticator,  
IN unsigned int unAuthNb);  

**Parameters**

- **hCard**: Input parameter
  Card connection handle, from gscBsiUtilCardConnect()

- **usAID**: Input parameter
  GCA AID value.

- **unAIDLen**: Input parameter
  GCA AID value length.

- **pstrAuthenticator**: Input parameter
  Array of structures containing the authenticator(s) specified by the Access Control Rule required to update a value in the container [obtained with gcGetContainerProperties()]. The caller of the provider must allocate this structure.

- **UnAuthNb**: Input parameter
  Size of *pstrAuthenticator* array. This allows to chain authenticators. If two authentication methods are chained, like with XAUTH_THEN_PIN, the size must be put to two. The Provider encapsulates the chaining.

**Return Codes**

- **BSI_OK**: Function successful
- **BSI_BAD_PARAM**
- **BSI_ACCESS_DENIED**
gscBsiUtilReleaseContext()

Purpose  Logout of the Card using the appropriate Authenticator.

Prototype  Int  gscBsiUtilReleaseContext (  
            IN UTILCardHandle  hCard,  
            IN unsigned char*  usAID,  
            IN unsigned int  unAIDLen);  

Parameters  HCard:  Input parameter  
            Card connection handle, from  
            gscBsiUtilCardConnect ()  

            UsAID:  Input parameter  
            GCA AID value.  

            UnAIDLen:  Input parameter  
            GCA AID value length.  

Return Codes  BSI_OK  Function successful  
BSI_BAD_PARAM

Generic Container Provider

The scope of the GC provider is to allow the creation, deletion, read and update of collections of {Tag, Length, value} data-items on the card, in a container, with the appropriate Access Control Management.

A Generic Container is a protected storage area on a smart card. It can be implemented as a file, on a file system card or on a WpSC card, or as a buffer managed by an applet on a VM card.

This provider API has been designed to provide an interface compatible with the Common Access Card Demographic information and the GSA J.8 identity information, as well as data-based extended services such as health data storage, certification and training data, roistering, property management, including the storage and retrieval of a biometric template.

The provider offers a number of “discovery” functions to the applications:
?? Discover the list of data items managed by the Container  
?? Discover the properties of the Container in terms of size and Access Control Rules

A container is created with a set of Access Control Rules for create, delete, read and update operations. In order to discriminate between data sets requiring different level of
security, different containers with a different set of Access Control Rules must be created. It is assumed that the keys required for read access and the keys required for update access can be different.

The available size of the **container** limits the number and size of data items within the card.

The provider identifies data-items with their **Tag**. This Tag enables the provider to access the data-item **Value** stored on the card. The provider offers also the service of listing the Tags managed by a Container, allowing thereby to an application to discover what the container manages, to acquire the security context, and to read the values.

**Generic Container Provider Module function list**

**gscBsiGcDataCreate()**
Create a new data item in the selected container. This will store a value and a Tag.

**gscBsiGcDataDelete()**
Delete a data in the selected container.

**gscBsiGcReadTagList()**
Read the list of Tags in the selected container.

**gscBsiGcReadValue()**
Retrieve the current value of a given Tag in the selected container.

**gscBsiGcUpdateValue()**
Update the current value of a given Tag with the provided value.
Access Control

**gscBsiGcGetContainerProperties()**
Retrieves the access conditions enabling to read and update Tags and their values for a specific container. Access conditions returned are common for all data items present in the selected container.

**Access Control Rules supported**
Access Control Rules (ACR) on operations:

```c
typedef struct strGC_ACR{
    unsigned int unCreateACR;
    unsigned int unDeleteACR;
    unsigned int unReadTagListACR;
    unsigned int unReadValueACR;
    unsigned int unUpdateValueACR;
} GCacr;
```

<table>
<thead>
<tr>
<th>Service</th>
<th>ACR supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>gscBsiGcDataCreate()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGcDataDelete()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGcReadTagList()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGcReadValue()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGcUpdateValue()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGcGetContainerProperties()</td>
<td>N/A</td>
</tr>
</tbody>
</table>
gscBsiGcGetContainerProperties()

Purpose
Retrieve the properties of the Container(s): Access Control Rules or read Tag list, read value and value operations on the container. The Access Control Rules returned are common for all data items managed by the selected container.

Prototype
```
int gscBsiGcGetContainerProperties(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    OUT GCacr* pstrACR,
    OUT GCContainerSize* pstrContainerSizes,
    OUT unsigned char* pszContainerVersion
    IN OUT unsigned int* punContainerVersionLength
);
```

Parameters
- **hCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`
- **usAID**: Input parameter
  GC AID value.
- **unAIDLen**: Input parameter
  GC AID value length.
- **pstrACR**: Output parameter
  Structure indicating all access conditions for operations. The caller must allocate the structure.
- **pstrContainerSizes**: Output parameter
  Structure indicating the sizes of the two storage areas of the container. The structure must be allocated by the caller.
- **pszContainerVersion**: Output parameter
  The version of the Generic Container.
- **pszContainerVersionLength**: Input/Output parameter
  The length of version of the Generic Container.

Return Codes
- **BSI_OK**: Successful function
- **BSI_BAD_AID**: There is no GC for this AID
- **BSI_BAD_PARAM**: One of the output access right pointer is NULL
gscBsiGcDataCreate()

**Purpose**
Create a new data item \{Tag, Length, Value\} in the selected container \{AID\}.
The creation of a data is performed in one atomic step.
Tag and Value of the data are provided to the provider.

To perform the creation of a data, the authenticator associated with the update value access condition to the container must be
set in the `pstrAuthenticator` structure. The access condition for updating values is returned by the
`gscBsiGcGetContainerProperties()` function.

**Prototype**

```c
Int gscBsiGcDataCreate(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN gcTag usTag,
    IN unsigned char* usValue,
    IN unsigned int unValueLen);
```

**Parameters**

- **hCard**: Input parameter
  Card connection handle, from
  `gscBsiUtilCardConnect()`

- **usAID**: Input parameter
  GCA AID value.

- **unAIDLen**: Input parameter
  GCA AID value length.

- **usTag**: Input parameter
  Tag of Data item to store.

- **usValue**: Input parameter
  Data value to store.

- **unValueLen**: Input parameter
  Data value length, in bytes.

**Return Codes**

- **BSI_OK**: Container successfully created.
- **BSI_BAD_HANDLE**: Unknown card handle
- **BSI_BAD_AID**: There is no GCA for this AID
BSI_ACCESS_DENIED  The Access Control Rule was not fulfilled. The access to the data item is rejected.
BSI_BAD_PARAM     Bad input parameter.
BSI_NO_MORE_SPACE No more space available in the GCA
BSI_TAG_EXISTS    Attempt to duplicate an existing tag.
gscBsiGcDataDelete()

**Purpose**
Delete all data item information in the selected container. Tag and value are lost.
To perform the deletion of a data, the corresponding Access Control Rule to the card Container must be fulfilled in the `pstrAuthenticator` structure accordingly to the requested *update* access rights returned by the `gscBsiGcGetContainerProperties()` function.

**Prototype**
```
int gscBsiGcDataDelete(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN gcTag usTag)
```

**Parameters**
- **hCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`
- **usAID**: Input parameter
  GC AID value.
- **unAIDLen**: Input parameter
  GC AID value length.
- **UsTag**: Input parameter
  Tag of data item to delete.

**Return Codes**
- **BSI_OK**: Deletion successful
- **BSI_BAD_HANDLE**: Unknown card handle
- **BSI_ACCESS_DENIED**: The Access Control Rule have not been fulfilled. The access to the data item is rejected.
- **BSI_BAD_AID**: There is no Generic Container for this AID
- **BSI_BAD_TAG**: No data with this tag was found in the container.
- **BSI_BAD_PARAM**: Bad input parameter.
**Purpose**

Read of the list of Tags of the selected container

To read the list of Tags, the access conditions to the read Tags operation must be set in the `pstrAuthenticator` structure. The access condition can be obtained with the `gscBsiGcGetContainerProperties()` function.

**Prototype**

```c
int gscBsiGcReadTagList(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    OUT gcTag* pTagArray,
    OUT unsigned int* pNbTags)
```

**Parameters**

- `hCard`: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`

- `usAID`: Input parameter
  GC AID value.

- `unAIDLen`: Input parameter
  GC AID value length.

- `pTagArray`: Output parameter
  Array of tags. The array of structures must be pre-allocated by the caller. The function can be called twice to discover the exact number of tags to allocate. The first time it can be called with NULL, and will return the number of tags.

- `pNbTags`: Output parameter
  Number of returned tags.

**Note:** `pNbTags` is an IN/Out Parameter where the passed in length is the length of the allocated buffer.

**Return Codes**

- `BSI_OK`: Read successful
- `BSI_BAD_HANDLE`: Unknown card handle
- `BSI_BAD_AID`: There is no container for this AID
- `BSI_BAD_AUTH`: Provided authentication value (or type) is incorrect
- `BSI_ACCESS_DENIED`: The Access Control Rule was not fulfilled
- `BSI_INSUFFICIENT_BUFFER`: The buffer provided to retrieve data is too small
- `BSI_BAD_PARAM`: Bad input parameter.
gscBsiGcReadValue()

**Purpose**
Retrieves the current Value of the given Tag in the selected container.

The Authenticator(s) specified by the Access Control Rule for the ReadValue operation must be set in the `pstrAuthenticator` structure. The Access Control Rule can be obtained with the `gscBsiGcGetContainerProperties()` function.

**Prototype**

```c
int gscBsiGcReadValue(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN gcTag usTag,
    OUT unsigned char* usValue,
    IN/OUT unsigned int* punValueLen)
```

**Parameters**

- `hCard` : Input parameter  
  Card connection handle, from `gscBsiUtilCardConnect()`

- `usAID` : Input parameter  
  GC AID value.

- `unAIDLen` : Input parameter  
  GC AID value length.

- `usTag` : Input parameter  
  Tag of the data item to read.

- `usValue` : Output parameter  
  Value read returned by the function. must be allocated by the caller

- `punValueLen` : Input/Output parameter  
  Value length.  
  IN : size of the allocated buffer  
  OUT: size of the value returned by the function.
### Return Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSI_OK</td>
<td>Read successful.</td>
</tr>
<tr>
<td>BSI_BAD_HANDLE</td>
<td>Unknown card handle.</td>
</tr>
<tr>
<td>BSI_BAD_AID</td>
<td>There is no container for this ID.</td>
</tr>
<tr>
<td>BSI_BAD_TAG</td>
<td>This GC instance does not manage a data item of this Tag.</td>
</tr>
<tr>
<td>BSI_BAD_AUTH</td>
<td>Provided authentication value (or type) is incorrect</td>
</tr>
<tr>
<td>BSI_ACCESS_DENIED</td>
<td>The Access Control Rule was not fulfilled. The access to the data item is rejected.</td>
</tr>
<tr>
<td>BSI_BAD_PARAM</td>
<td>punValueLen is NULL.</td>
</tr>
<tr>
<td>BSI_INSUFFICIENT_BUFFER</td>
<td>The buffer provided to retrieve data is too small</td>
</tr>
</tbody>
</table>
gscBsiGcUpdateValue()

Purpose
Updates the current Value of the given tag with the provided value.

The Authenticator(s) specified by the Access Control Rule for the UpdateValue operation must be set in the pstrAuthenticator structure. The Access Control Rule can be obtained with the gscBsiGcGetContainerProperties() function.

Prototype
int gscBsiGcUpdateValue(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN gcTag usTag,
    IN unsigned char* usValue,
    IN unsigned int unValueLen
);  

Parameters
hCard: Input parameter
Card connection handle, from gscBsiUtilCardConnect()

usAID: Input parameter
GC AID value.

unAIDLen: Input parameter
GC AID value length.

UsTag: Input parameter
Tag of data item to update the value of.

usValue: Input parameter
Value for the data-item to be updated to.

unValueLen: Input parameter
Value Length.

Return Codes
BSI_OK Data value successfully updated.
BSI_BAD_HANDLE Unknown card handle
BSI_BAD_AID There is no container for this ID
BSI_BAD_TAG This GC instance does not manage a data item of this Tag.
BSI_BAD_AUTH Provided authenticator value or type
is incorrect

BSI_ACCESS_DENIED       The Access Control Rule was not fulfilled. The access to the data item is rejected.

BSI_NO_MORE_SPACE      No more space available in this GC

BSI_BAD_PARAM          Bad input parameter.
Cryptographic Provider

Cryptographic Provider Module function list

RANDOM NUMBER operations
gscBsiGetChallenge()
Retrieve a challenge from the card.

SKI operations
gscBsiSkiInternalAuthenticate()
Compute a symmetric key cryptography authenticator in response to a challenge.

PKI operations
gscBsiPkiCompute()
Compute the private key encrypt/decrypt. The mandatory PKI algorithm of the BSI is RSA_NO_PAD.

gscBsiPkiReadCertificate()
Read the certificate

gscBsiPkiGetProperties()
Retrieve status information on the PKI instance.

Access Control
gscBsiGetCryptoProperties()
Retrieve access conditions enabling to read and update Tags and values for a specific container.
Access conditions returned are common for all data items present in the selected container.

Access Control Rules supported
Access Control Rules (ACR) on operations cryptographic operations:
typedef struct strCrypto_ACR{
    unsigned int unGetChallengeACR;
    unsigned int unInternalAuthenticateACR;
    unsigned int unPkiComputeACR;
    unsigned int unReadCertificateACR;
} CRYPTOacr;
<table>
<thead>
<tr>
<th>Service</th>
<th>ACR supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>gscBsiGetChallenge()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiSkiInternalAuthenticate()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiPkiCompute()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiPkiReadCertificate()</td>
<td>BSI_ACR_ALWAYS</td>
</tr>
<tr>
<td>gscBsiGetCryptoProperties()</td>
<td>N/A</td>
</tr>
</tbody>
</table>

gscBsiGetCryptoProperties()

**Purpose**
Retrieves card dependant data (card serial number).

**Synopsis**
```c
int gscBsiGetCryptoProperties(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    OUT CRYPTOacr* pstrACR,
    OUT unsigned int* punKeyLength);
```

**Parameters**
- **hCard**: Input parameter
  Card connection handle, from
  `gscBsiUtilCardConnect()`

- **usAID**: Input parameter
  PKI AID value.

- **unAIDLen**: Input parameter
  PKI AID value length.

- **pstrACR**: Output parameter
  The caller must allocate structure indicating all access conditions for operations. The structure.

- **punKeyLength**: Output parameter
  RSA key length managed by the instance.

**Return Codes**
- **PKI_OK** card present
- **PKI_BAD_HANDLE** Unknown card handle
gscBsiGetChallenge()

Purpose
Retrieves a Challenge from the selected AID. This function is to be used only in for strong key management requirements when the External Authentication key should not be exposed in clear text. It is the first step for a strong authentication of the application to the card.

Prototype
```
int gscBsiGetChallenge(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    OUT unsigned char* uszChallenge,
    IN unsigned int unChallengeLen)
```

Parameters

*hCard*: Input parameter
Card connection handle, from
```
gscBsiUtilCardConnect()
```

*usAID*: Input parameter
GC AID value.

*unAIDLen*: Input parameter
GC AID value length.

*uszChallenge*: Output parameter
very good quality random number returned by the GC applet.

*punChallengeLen*: Output parameter
8 Bytes.

Return Codes
- **BSI_OK**: Read successful.
- **BSI_BAD_HANDLE**: Unknown card handle
- **BSI_BAD_AID**: There is no container for this ID.
- **BSI_BAD_TAG**: This GC instance does not manage a data item of this Tag.
- **BSI_BAD_AUTH**: Provided authentication value (or type) is incorrect
- **BSI_BAD_PARAM**: *punValueLen* is NULL
gscBsiPkiCompute()

**Purpose**
Computes RSA operations for a given challenge.

**Synopsis**
```c
int gscBsiPkiCompute(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN unsigned char ucAlgoID,
    IN unsigned char* pMessage,
    IN unsigned int unMessageLen,
    IN/OUT unsigned char* pResult,
    IN/OUT unsigned int* punResultLen);
```

**Parameters**
- **hCard**: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`
- **usAID**: Input parameter
  PKI AID value
- **unAIDLen**: Input parameter
  PKI AID length
- **ucAlgoID**: Input parameter
  Algorithm Identifier. In this version of the interoperability specification, the only algorithm supported is RSA_NO_PAD, with an ID of “0xA3”.
- **pMessage**: Input parameter
  Message to sign.
- **unMessageLen**: Input parameter
  Message length to sign.
- **pResult**: Output parameter
  Buffer containing the signature.
- **punResultLen**: Output parameter
  Length of the signature buffer.

**Note:**
PunResultLen must indicate the size of the pre-allocated memory. The function may be called with pResult = NULL, in this case the function returns the signature buffer length in punResultLen.
Note: This call assumes the card uses RSA_NO_PAD (offers maximum application interoperability) so the operation of encryption (signature) is the same as decryption.

<table>
<thead>
<tr>
<th>Return Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKI_OK</td>
<td>Function successful</td>
</tr>
<tr>
<td>PKI_BAD_HANDLE</td>
<td>Unknown card handle</td>
</tr>
<tr>
<td>PKI_BAD_PARAM</td>
<td>Bad parameter</td>
</tr>
<tr>
<td>PKI_ACCESS_DENIED</td>
<td>Wrong PIN code.</td>
</tr>
<tr>
<td>PKI_INSUFFICIENT_BUFFER</td>
<td>The pre-allocated buffer was too small to contain the result.</td>
</tr>
</tbody>
</table>
gscBsiPkiReadCertificate()

Purpose

Reads the certificate from the card

Synopsis

```c
int gscBsiPkiReadCertificate(
    IN IDCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    OUT unsigned char* pCertificate,
    OUT unsigned int* punCertificateLen);
```

Parameters

- `hCard`: Input parameter
  Card connection handle, from `gscBsiUtilCardConnect()`
- `usAID`: Input parameter
  PKI AID value
- `unAIDLen`: Input parameter
  PKI AID length
- `pCertificate`: Output parameter
  Buffer containing the signature.
- `punCertificateLen`: Output parameter
  Length of the signature buffer.

Return Codes

- `PKI_OK`: Function successful
- `PKI_BAD_HANDLE`: Unknown card handle
- `PKI_BAD_PARAM`
- `PKI_ACCESS_DENIED`: Wrong PIN code.

PunResultLen must indicate the size of the pre-allocated memory. The function may be called with `pResult = NULL`, in this case the function returns the signature buffer length in `punResultLen`. 
gscBsiSkiInternalAuthenticate ()

Purpose
Compute a symmetric key cryptography authenticator in response to a challenge.

Synopsis
int gscBsiSkiInternalAuthenticate(
    IN UTILCardHandle hCard,
    IN unsigned char* usAID,
    IN unsigned int unAIDLen,
    IN unsigned char ucAlgoID,
    IN unsigned char* uszChallenge,
    IN unsigned int unChallengeLen,
    OUT unsigned char* uszCryptogram,
    OUT unsigned int* punCryptogramLength);

Parameters
hCard: Input parameter
Card connection handle, from gscBsiUtilCardConnect()

usAID: Input parameter
SKI AID value.

unAIDLen: Input parameter
SKI AID value length.

ucAlgoID: Input parameter
Algorithm Identifier. In this version of the interoperability specification, the only algorithm supported for this operation is DES3_ECB, double key length, with an ID of “0x81”.

uszChallenge: Input parameter
Challenge generated by the Application and submitted to the card.

unChallengeLen: Input parameter
The length of the Challenge. In this version of the Interoperability Specification the challenge length is always 8 bytes.

uszCryptogram: Output parameter
The cryptogram computed by the card.

punCryptogramLength: Output parameter
The length of the cryptogram computed by the card. The result of a DES computation is always 8 Bytes, BUT, some cards truncate the cryptogram so that skiInternalAuthenticate cannot be used to encrypt data. Also, this truncation makes the authentication protocol stronger.

Return Codes
PKI_OK card present
PKI_BAD_HANDLE  Unknown card handle
Appendix B - GSC: Interface Definition – Card Edge Presentation

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI api and amounts to the “top” part of an SPS.

A second level of interoperability is defined to allow smart cards to interoperate: the card-edge interface. This level, allows any SPS provider to interoperate with any smart card that supports the card edge interface.

This document presents the Card-Edge interface of the Government Smart Card.

Limitations
The proposed card-edge interface is an operational API and not a management API. It does not provide services like applet download or applet instantiation. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation.

PIN management functions like Change PIN or Unlock PIN are not part of this API.

The smart card is supposed to be already initialized: applets are downloaded and instantiated and the file system is created.

Establishing these limitations is a balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder, or the generation of a key pair. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:
- Is it a rare operation?
- Is it feasible or very difficult to define an interoperable method for this service?

The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the specification would have the result that the specification would not guaranty that all implementations would inter-operate.

Compatibility
This Card-Edge specification can be implemented on reasonbably well behaved File System Cards (with respect to ISO7816-4), JAVACARD2.1, Smart Card for Windows 1.1, and MULTOS.

Documents
The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.
The Government Smart Card Interoperability Standard

Documents describing the GSC Card-Edge Interface v1.0
- The Government Smart Card Card-Edge Interface Presentation v1.0
- The Employee Identification (ID) and Physical Access Interoperability Specification v1.0
- The Common Access Card card-edge interface documents:
  - Government Smart Card Generic Container card-edge Application Programming Interface for VM cards V1.0
  - Government Smart Card PKI card-edge Application Programming Interface for VM cards V1.0
  - Government Smart Card SKI card-edge Application Programming Interface for VM cards V1.0
The GSC Card-Edge Interface v1.0

Definition of the GSC Card-Edge Interface
The GSC cards-edge interface provides the basis allowing the cards to interoperate for
the BSI services.
The GSC card-edge interface is fully unambiguously described by:

1. A Basic Data Model for the J.8 data, that specifies,
   ?? the Internal TLV storage format of the J.8 data,
   ?? the Tags assigned to each J.8 data item,
   ?? a grouping of the data-item into a set of Containers
   ?? fixed Access Control Rule to access these containers,
   ?? the naming of those containers,
   ?? on file system cards the containers are implemented as Transparent files under a GSA
   Dedicated File,
   ?? on VM cards, the containers are implemented as instances of the Generic Container
   Applet.

2. A basic set of cryptographic services, with the naming of the PKI and SKI
   cryptographic capabilities required for the BSI.

3. A functional interface
   ?? On file system cards, an arbitrary default set of ISO7816-4 and cryptographic
   APDUs.
   ?? On VM cards, the APDU set of the Common Access Card.

4. The concept of Capability Container to describe the way the implementation on the
   card differs from the model described above.

Access Control Rules compact coding proposal
The Access Control Rules that have been exposed so far in the BSI provider API are:
?? **Always**: the corresponding service can be provided without restrictions
?? **PIN protected**: the corresponding service can be provided only if its associated PIN
   code has been already verified
?? **External authenticate**: the corresponding service can be provided only after a
   get_challenge APDU.
?? **External Authenticate then PIN**: the two methods must be chained successfully
   before access to the service is granted. This allows the authentication of the
   Application AND of the user.
?? **External Authenticate or PIN**: either one of the two control gives access to the
   service. This allows for a CardHolder validation when a PIN PAD is available and for
   an external authentication when no PINPAD is available. Or, this provides an
   Authentication method when the Application cannot be trusted to perform an external
   Authentication and to protect the external authentication key.
Secure Channel: the corresponding service can be provided through a Secure Channel managed by a card Secure Messaging layer that can be Open Platform, DIN, and the ISO7816-4 Secure Messaging.

Never: the corresponding service can never be provided.

This acr can be represented in the following manner:

```
acr ::= SEQUENCE{
    readAcr BIT STRING SIZE(4) -- The read access control rule, a nibble
    writeAcr BIT STRING SIZE(4) -- The write access control rule, a nibble
}
```

Proposed values:

<table>
<thead>
<tr>
<th>Access Control Rule</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PIN protected</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>External authenticate</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>External Authenticate then PIN</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>External Authenticate or PIN</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Secure Channel</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Never</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Internal TLV format

Each SIMPLE-TLV data object shall consist of 2 or 3 consecutive fields:

The tag field T consists of a single byte encoding only a number from 1 to 254. No class or construction types are coded.

The length field consists of 1 or 3 consecutive bytes. If the leading byte of the length field is in the range from ‘00’ to ‘FE’, then the length field consists of a single byte encoding an integer L valued from 0 to 254. If the leading byte is equal to ‘FF’, then the length field continues on the two subsequent bytes which encode an integer L with a value from 0 to 65,535.

If L is not null, then the value field V consists of L consecutive bytes. If L is null or if a tag is omitted from its file/buffer, then the data object is empty: there is no value field for that tag.

Card Identifier compact naming proposal

Any card compliant with this specification must receive a unique identifier, attributed by the GSA.

The Identifier must at least provide the type of the card:

CardUniqueIdentifier ::= SEQUENCE
{  
gsa-rid       OCTET STRING SIZE (5) -- 5 bytes OBJECT
IDENTIFIER,
ManufacturerID BIT STRING SIZE(8)– a unique manufacturer identifier
cardType   CHOICE {
    fscsystemCard    BIT STRING SIZE(8) ::= 0
    javaCard         BIT STRING SIZE(8) ::= 1
    scpW              BIT STRING SIZE(8) ::= 2
    Multos            BIT STRING SIZE(8) ::= 3
    ...
}  
cardID      STRING
}

PIN and CHV
?? The PIN is defined as a system Personal Identification Number for applications external to the card.

?? The CHV or CardHolder Verification, is the designation for the PIN used within the card to permit access to a particular file or buffer.

Common Access ID Card Data Structure and Model

Data Structure and Model
The GSA Common Data Model will be contained within a single directory (DF) and designated by a single Application Identifier (AID). For cards supporting AID, it is the preferred method for selecting the application however a reference to the directory’s file identifier (FID) must be included in the card capabilities file for those cards that do not support AID.

The Common Data Model is organized as 7 containers (files or buffers) holding a collection of SIMPLE-TLV data objects. The read access rules are defined in the table below however it is up to each agency to define the write/modify rules.

Container compact naming proposal
The standard method to access a service on a smart card is to select it with an AID. This is actually the only method that works with a JavaCard, and unfortunately it is not supported by most file system cards.
Therefore, we introduce the concept of CardURL, which can be used to uniquely reference a service offered by the smart card.
- In the context of a JavaCard the CardURL is an AID.
- In the context of a File System card the Card URL must be interpreted in the following manner:

GSAcurl ::= SEQUENCE   -- This is considered as an AID on a VM card and
    -- must be interpreted for a file system card.
{  
    gsa-rid = OCTET STRING SIZE(5) -- 5 bytes OBJECT IDENTIFIER,  
    cardApplicationType ::= CHOICE {  
        J8data BIT STRING SIZE(8) ::= 0  
        SKI BIT STRING SIZE(8) ::= 1  
        PKI BIT STRING SIZE(8) ::= 2  
        SecureDataExtended BIT STRING SIZE(8) ::= 3 -- secure storage application other than J.8  
        CryptoSKIExtended BIT STRING SIZE(8) ::= 4  
        CryptoPKIExtended BIT STRING SIZE(8) ::= 5  
        ...  
    }  
    gsaDF = BYTE[2] -- CHOICE {  
        KeyID BIT STRING SIZE(16)  
        FileID BIT STRING SIZE(16)  
    }  
    acr ::= SEQUENCE{  
        read readAcr -- see definition above  
        write writeAcr  
    }  
}  

The following matrix contains the proposed FIDs for the J.8 containers for file system cards:

<table>
<thead>
<tr>
<th>File/Buffer Description</th>
<th>FID</th>
<th>Maximum Length (Bytes)</th>
<th>Read Access Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td>DB00</td>
<td>508</td>
<td>Always</td>
</tr>
<tr>
<td>General Information</td>
<td>2000</td>
<td>508</td>
<td>Always</td>
</tr>
<tr>
<td>Protected Personal Information</td>
<td>2100</td>
<td>18</td>
<td>After Verify CHV</td>
</tr>
<tr>
<td>Access Control</td>
<td>3000</td>
<td>48</td>
<td>Always</td>
</tr>
<tr>
<td>Login</td>
<td>4300</td>
<td>140</td>
<td>After Verify CHV</td>
</tr>
<tr>
<td>Card Information</td>
<td>5000</td>
<td>164</td>
<td>Always</td>
</tr>
<tr>
<td>Biometrics – X.509 Certificate</td>
<td>6000</td>
<td>2012</td>
<td>Always</td>
</tr>
<tr>
<td>PKI – Digital Signature Certificate</td>
<td>7000</td>
<td>3016</td>
<td>After Verify CHV</td>
</tr>
</tbody>
</table>

Once the GSA-RID is known and the GSA-DF has been decided it becomes possible to attribute an unambiguous universal name to the containers, files and applets.

**Error Detection**

The trailing Data Object of each file/buffer will consist of an Error Detection Code (EDC) Object for the file. Either a LRC or CRC may be used.
1. **LRC:** The longitudinal redundancy check consists of one byte. Its value shall be such that the exclusive-oring of all the bytes of the block is null. (ISO7816-3). The tag for an LRC Object is FE.

2. **CRC** The cyclic redundancy check consists of two bytes. For its value see ISO3309. The tag for an CRC Object is FD

### File/Buffer Formats & Tag Definitions

<table>
<thead>
<tr>
<th>General Information File / Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>First Name</td>
</tr>
<tr>
<td>Middle Name</td>
</tr>
<tr>
<td>Last Name</td>
</tr>
<tr>
<td>Suffix</td>
</tr>
<tr>
<td>Government Agency</td>
</tr>
<tr>
<td>Bureau Name</td>
</tr>
<tr>
<td>Agency Bureau Code</td>
</tr>
<tr>
<td>Department Code</td>
</tr>
<tr>
<td>Position/Title</td>
</tr>
<tr>
<td>Building Name</td>
</tr>
<tr>
<td>Office Address 1</td>
</tr>
<tr>
<td>Office Address 2</td>
</tr>
<tr>
<td>Office City</td>
</tr>
<tr>
<td>Office State</td>
</tr>
<tr>
<td>Office ZIP</td>
</tr>
<tr>
<td>Office Country</td>
</tr>
<tr>
<td>Office Phone</td>
</tr>
<tr>
<td>Office Extension</td>
</tr>
<tr>
<td>Office Fax</td>
</tr>
<tr>
<td>Office Email</td>
</tr>
<tr>
<td>Office Room Number</td>
</tr>
<tr>
<td>Non-Government Agency</td>
</tr>
<tr>
<td>SSN Designator</td>
</tr>
<tr>
<td>Error Detection Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protected Personal Information File / Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td>Social Security Number</td>
</tr>
<tr>
<td>Date of Birth</td>
</tr>
<tr>
<td>Gender</td>
</tr>
<tr>
<td>Error Detection Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Control File / Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
</tr>
<tr>
<td>--------------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Contract No.: GS00T00ALD0208  PS02  © 2000
<table>
<thead>
<tr>
<th>Data Element (TLV)</th>
<th>Tag</th>
<th>Type</th>
<th>Max. Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEIWG Data</td>
<td>30</td>
<td>Fixed</td>
<td>40</td>
</tr>
<tr>
<td>PIN</td>
<td>31</td>
<td>Fixed Numeric</td>
<td>10</td>
</tr>
<tr>
<td>Domain (Facility/System ID)</td>
<td>32</td>
<td>Variable</td>
<td>8</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE</td>
<td>LRC</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Login Information File / Buffer</th>
<th>EF 4000</th>
<th>CHV Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
<td>Tag</td>
<td>Type</td>
</tr>
<tr>
<td>User ID</td>
<td>40</td>
<td>Variable</td>
</tr>
<tr>
<td>Domain</td>
<td>41</td>
<td>Variable</td>
</tr>
<tr>
<td>Password</td>
<td>42</td>
<td>Variable</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE</td>
<td>LRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Card Information File / Buffer</th>
<th>EF 5000</th>
<th>Always Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
<td>Tag</td>
<td>Type</td>
</tr>
<tr>
<td>Issuer ID</td>
<td>50</td>
<td>Variable</td>
</tr>
<tr>
<td>Issuance Counter</td>
<td>51</td>
<td>Variable</td>
</tr>
<tr>
<td>Issue Date</td>
<td>52</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Expiration Date</td>
<td>53</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Card Type</td>
<td>54</td>
<td>Variable</td>
</tr>
<tr>
<td>Demographic Data Load Date</td>
<td>55</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Demographic Data Expiration Date</td>
<td>56</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Card Security Code</td>
<td>57</td>
<td>Fixed Text</td>
</tr>
<tr>
<td>Card ID AID</td>
<td>58</td>
<td>Variable</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE</td>
<td>LRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Biometrics – X.509 Certificate File / Buffer</th>
<th>EF6000</th>
<th>Always Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element (TLV)</td>
<td>Tag</td>
<td>Type</td>
</tr>
<tr>
<td>Template</td>
<td>60</td>
<td>Variable</td>
</tr>
<tr>
<td>Certificate</td>
<td>61</td>
<td>Variable</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE/FD</td>
<td>LRC / CRC</td>
</tr>
<tr>
<td>PKI – Digital Signature Certificates File / Buffer</td>
<td>EF 7000</td>
<td>CHV Verify</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>Data Element (TLV)</td>
<td>Tag</td>
<td>Type</td>
</tr>
<tr>
<td>Certificate</td>
<td>70</td>
<td>Variable</td>
</tr>
<tr>
<td>Issue Date</td>
<td>71</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Expiration Date</td>
<td>72</td>
<td>Date (YYYYMMDD)</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE/FD</td>
<td>LRC / CRC (TBD)</td>
</tr>
</tbody>
</table>

**Reserved Container Tags**

Tags within the range of 00 through 9F are reserved for the Common Access ID Card data structure. Agencies are free to utilize tags within the range of A0 through EF.

**Character Sets**

The Security Enterprise Integration Working Group (SEIWG) data element within the Access Control File container as required by the SEIWG specification will be of the Binary Coded Decimal (BCD) character set. All other file containers and data elements will use the American Standards Code for Information Interchange (ASCII) character set.

**Card Capability**

The Architecture Subcommittee of the Government Smart Card Interoperability Committee has completed an analysis of the APDU-level interoperability issue. The background information and opinions provided by the five primes are included in the Subcommittee's report dated June 23. Based on this information, the Subcommittee has developed a model for the Government Smart Card Service Provider Module (GSC-SPM).

APDU A Card Capability Container is included for each card that contains a compact description of the differences between the card’s APDU set and usage of the J.8 model, and the default APDU set and the J.8 model. Once the Card Capability Container is processed, the SPM can configure itself to interface with the card and execute the most important commands to achieve a minimum level of interoperability.

The Card capability Container is implemented as a Transparent File on a file system card and as an instance of the Generic Container applet on a VM card.

The absence of a Capability Container on a card means complete compliance with the default APDU set and with the J.8 model.

The Card Capability Container allows interoperability between a broad range of cards without the problems and costs associated with configuration management techniques used in the past. As an interesting side-effect, the proposed framework can be used for extended services and make them,

1> easier to implement,
2> easier to make interoperable when it becomes a need.

The Capability model offers two level of flexibility to handle variance with nominal J.8.
Container Level Flexibility
An Agency might decide that the Container as defined by nominal J.8 are suitable for the Agency’s requirements, except for the Access Control Rules (for example, replacement of the ALWAYS acr by an EXTERNAL AUTHENTICATE acr for read access to a container).
In such a case, the CardURL of the container is indicated in the Capability Container with its specific ACR.
And the BSI provider can provide the information to the application so that the application can establish the Security Context required by the Container, and this without breaking the J.8 compatibility.

Tag Level Flexibility: redirection Tag
In the case an Agency decides that a specific subset of Tags need a particular Security Context and that a specific access control rule should be enforced, it is possible to create a Container for this set of Tags.
A special Tag is added to J.8, the redirection Tag. This Tag can be used to indicate to the BSI Provider which J.8 Tags are being “redirected” to the Container.
The “value” part of the TLV for this redirection Tag can be described as follow:

Redirection_value ::= SEQUENCE {
   Container GSAcurl
   SEQUENCE OF {
      Tags BYTE -- The “redirected Tags”
   }
   ...
}
A J8 Container can have any number of “redirection flags” to handle Tag level exceptions to the J8 nominal model.

Accessing the capability file
The Capability Container is the main instrument of interoperability provided by the specification. There should therefore be a agreed upon method to access it. On VM card the accessibility is immediate once the AID of the container is known. On file system cards however a heuristic needs to be agreed upon.
The provider implementing Card-Edge will attempt different APDUs until it eventually completes the reading of the capability file.
The sequence is composed of:
1>- the select MF
2>- the select capability file EF
3>- the read_binary of the file

Hereafter, an interoperable procedure that should be applied to the capability file:

**Universal Procedure for Selecting EF file under a DF.**
1. Send command APDU as follows to select MF:

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>FIDH</th>
<th>FIDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST CLA</td>
<td>A4</td>
<td>00</td>
<td>00</td>
<td>02</td>
<td>3F</td>
<td>00</td>
</tr>
</tbody>
</table>

The TEST CLA byte we use for our subset of cards are: 00, C0,F0,80,BC,01.
(Additional test values for CLA are: 0X,8X,9X,AX,B0-CF.)

2. Wait for Status bytes. If Status Bytes are “6E00”, Class is not supported. Loop back and attempt another CLA.

3. If Status Bytes are “9000” or “61XX”, correct command structured and CLA

4. If Status Bytes are none of the above, set P2=0C and repeat steps 1 through 3.
5. Once CLA has been determined, select DF under MF

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>FIDH</th>
<th>FIDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determined CLA</td>
<td>A4</td>
<td>00</td>
<td>P2</td>
<td>02</td>
<td>DFID H</td>
<td>DFID L</td>
</tr>
</tbody>
</table>

6. If Status Bytes return error codes (values other than “9000” or “61XX”, set P1=01.

7. To select a EF under a selected DF:

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>FIDH</th>
<th>FIDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determined CLA</td>
<td>A4</td>
<td>00</td>
<td>P2</td>
<td>02</td>
<td>EFID H</td>
<td>EFID L</td>
</tr>
</tbody>
</table>

8. If Status Bytes return error codes (values other than “9000” or “61XX”, set P1=02.

9. To Read a binary file with no secure messaging, use the following APDU:

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determined CLA</td>
<td>B0</td>
<td>Off/H</td>
<td>Off/L</td>
<td>L</td>
</tr>
</tbody>
</table>

Note P1 and P2 are offset to start reading from. L is length of data to read.
Card Capability File Grammar

The GSA Card Capability File will be contained in the Master Directory (3F00) and designated by the Capabilities Application Identifier (AID: GSA-RID||DB00) as well as the FID: DB 00

The Card Capabilities File consists of a collection of SIMPLE-TLV data objects. It will be configured for ALWAYS READ however it is up to each agency to define the write/modify rules.

File/Buffer Format & Tag Definitions

<table>
<thead>
<tr>
<th>Data Capabilities File</th>
<th>EF DB00</th>
<th>Always Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card Identifier</td>
<td>F0</td>
<td>Fixed-tbd</td>
</tr>
<tr>
<td>Capability File version</td>
<td>F1</td>
<td>Fixed-tbd</td>
</tr>
<tr>
<td>Capability Grammar version</td>
<td>F2</td>
<td>Fixed-tbd</td>
</tr>
<tr>
<td>Applications CardURL</td>
<td>F3</td>
<td>Variable – Multiple Objects</td>
</tr>
<tr>
<td>Redirection Tag</td>
<td>FA</td>
<td>Variable</td>
</tr>
<tr>
<td>Capability Tuples (CT’s)</td>
<td>FB</td>
<td>Variable: Collection of 2 Byte Tuples</td>
</tr>
<tr>
<td>Status Tuples (ST’s)</td>
<td>FC</td>
<td>Variable: Collection of 2 Byte Tuples</td>
</tr>
<tr>
<td>Optional Issuer Objects</td>
<td>Issuer Def</td>
<td>Variable</td>
</tr>
<tr>
<td>Error Detection Code</td>
<td>FE</td>
<td>LRC</td>
</tr>
</tbody>
</table>

The Card Identifier will be a code specified by the GSA for each card type. It should include critical card parameters such as max buffer size and supported encryption algorithms and key lengths.

The Card Capabilities File may contain multiple instances of the Application AID tags. They can be assembled into a list of the applications, including FID’s and paths, Key Identifiers and Access Control Methods, which are supported by the card.

The Card Capabilities File will contain a single Capability Tuple (CT) object, which consists of a collection of two byte tuples, which define the capabilities, formats and procedures supported by the card. The GSA will define a default set of capability tuples that represent a generic implementation of the ISO 7816 standard. It will only be necessary to include CT’s to indicate a variance between the cards capabilities and the default set.

The Card Capabilities File may contain a single Status Tuple (ST) object, which consists of a collection of three byte tuples, which define the possible status codes for each function. It will only be necessary to include ST’s which deviate between the cards status codes and the status codes defines in ISO 7816-4.
The card issuer may include additional TLV objects in the Card Capabilities File for application specific purposes. These are not needed for interoperability but may be used to facilitate extended applications. They may be ignored by any implementation without affecting interoperability. Any optional objects that are not recognized will be ignored.

**Capability Tuple Construction**

Capability Tuples will consist of two bytes, labeled C and V, which describe the details of a particular function. They are formatted as follows:

*APDU*

<table>
<thead>
<tr>
<th>C</th>
<th>1.1.1.1.1.1 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>0= Const</td>
<td>Parameter</td>
</tr>
<tr>
<td>1= Desc</td>
<td>Function Code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>If C bit 7 = 0 Then V contains a constant value</td>
<td></td>
</tr>
<tr>
<td>If C bit 7 = 1 Then V contains a Descriptor code</td>
<td></td>
</tr>
<tr>
<td>Function Codes</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>Reserved</td>
</tr>
<tr>
<td>01</td>
<td>Select File</td>
</tr>
<tr>
<td>02</td>
<td>Transparent Read (Binary)</td>
</tr>
<tr>
<td>03</td>
<td>Update Binary File</td>
</tr>
<tr>
<td>04</td>
<td>Update Binary File (Secure Messaging)</td>
</tr>
<tr>
<td>05</td>
<td>RSA Compute</td>
</tr>
<tr>
<td>06</td>
<td>Get Challenge</td>
</tr>
<tr>
<td>07</td>
<td>Get Response</td>
</tr>
<tr>
<td>08</td>
<td>Verify PIN (CHV)</td>
</tr>
<tr>
<td>09</td>
<td>Internal Authenticate</td>
</tr>
<tr>
<td>0A</td>
<td>External Authenticate</td>
</tr>
<tr>
<td>0B</td>
<td>RFU</td>
</tr>
<tr>
<td>0C</td>
<td>Select KEY</td>
</tr>
<tr>
<td>0D</td>
<td>Card Specific 1</td>
</tr>
<tr>
<td>0E</td>
<td>Card Specific 2</td>
</tr>
<tr>
<td>0F</td>
<td>Card Specific 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
<tr>
<td>03</td>
</tr>
<tr>
<td>04</td>
</tr>
<tr>
<td>05</td>
</tr>
<tr>
<td>06</td>
</tr>
<tr>
<td>07</td>
</tr>
<tr>
<td>Descriptor Codes</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>00 – 0F</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>1A</td>
</tr>
<tr>
<td>1B</td>
</tr>
<tr>
<td>1C</td>
</tr>
<tr>
<td>1D</td>
</tr>
<tr>
<td>1E</td>
</tr>
<tr>
<td>1F</td>
</tr>
</tbody>
</table>
Status Tuple Construction

Status Tuples will consist of three bytes, labeled S, SW1 and SW2, which describe the possible status conditions for each function. Multiple sets of SW1 and SW2 may translate into a single Status Condition.

The purpose of the Status Tuples is to map a card’s non-standard status response SW1 & SW2 into a common set of status conditions for a given function. It is not mandatory to list any status conditions which conform with ISO-7816.

<table>
<thead>
<tr>
<th>S</th>
<th>1.1.1.1.2 SW</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Status Condition</td>
<td>Function Code</td>
</tr>
</tbody>
</table>

Status Condition

- 00 Successful Completion
- 01 Successful Completion – Warning 1
- 02 Successful Completion – Warning 2
- 03 Reserved
- 04 Reserved
- 05 Reserved
- 06 Reserved
- 07 Reserved
- 08 Access Condition not Satisfied
- 09 Function not Allowed
- 0A Inconsistent Parameter
- 0B Data Error
- 0C Wrong Length
- 0D Function not compatible with file structure
- 0E File/Record not Found
- 0F Function Not Supported
Appendix C - GSC: Interface Definition – Card Edge for File System Cards

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI api and amounts to the “top” part of an SPS.

A second level of interoperability is defined to allow smart cards to interoperate: the card-edge interface. This level, allows any SPS provider to interoperate with any smart card that supports the card edge interface.

This document presents the Card-Edge interface of the Government Smart Card.

Limitations
The proposed card-edge interface is an operational API and not a management API. It does not provide services like applet download or applet instantiation. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation.

PIN management functions like Change PIN or Unlock PIN are not part of this API. The smart card is supposed to be already initialized: applets are downloaded and instantiated, file system is created, etc….

Establishing these limitations is a balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder, or the generation of a key pair. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:
- Is it a rare operation?
- Is it feasible or very difficult to define an interoperable method for this service?

The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the specification would have the result that the specification would not guaranty that all implementations would inter-operate.
The Government Smart Card Interoperability Standard

Introduction
The present document defines the default functional interface of File System cards: an arbitrary default set of ISO7816-4 and cryptographic APDUs

The Capability file described in other Card-edge specification documents provides a formalism that allows to describe how the interface of a specific file system card differs from this default.
## Proposed GSA Default APDU Set

<table>
<thead>
<tr>
<th>Card Type</th>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select DF File under MF</td>
<td>00</td>
<td>A4</td>
<td>01</td>
<td>00</td>
<td>L (02)</td>
<td>File ID (2 Bytes)</td>
<td></td>
</tr>
<tr>
<td>Select EF File under selected DF</td>
<td>00</td>
<td>A4</td>
<td>02</td>
<td>00</td>
<td>L (02)</td>
<td>File ID (2 Bytes)</td>
<td>See Note 1</td>
</tr>
<tr>
<td>Read Binary (Transparent)</td>
<td>00</td>
<td>B0</td>
<td>Off/H</td>
<td>Off/L</td>
<td>L</td>
<td>Response</td>
<td>AC must be fulfilled</td>
</tr>
<tr>
<td>Read Binary (Transparent)- Secure Messagi</td>
<td>04</td>
<td>B0</td>
<td>Off/H</td>
<td>Off/L</td>
<td>L+03</td>
<td>Plain Data + Crypto</td>
<td>See Note 2 (pre – APDU SelFk)</td>
</tr>
<tr>
<td>Update Binary (Normal)</td>
<td>00</td>
<td>D6</td>
<td>Off/H</td>
<td>Off/L</td>
<td>L</td>
<td>Data to Update</td>
<td>AC must be fulfilled</td>
</tr>
<tr>
<td>Update Binary (Secure Msg)</td>
<td>04</td>
<td>D6</td>
<td>Off/H</td>
<td>Off/L</td>
<td>L+03</td>
<td>Plain Data + Cryptogram</td>
<td>See Note 3</td>
</tr>
<tr>
<td>Internal Authenticate</td>
<td>00</td>
<td>88</td>
<td>AI</td>
<td>KN</td>
<td>L (08)</td>
<td>8 byte challenge</td>
<td>See Note 4 (pre-Verify CHV) (post-Get Response)</td>
</tr>
<tr>
<td>Get Response</td>
<td>00</td>
<td>C0</td>
<td>00</td>
<td>00</td>
<td>L</td>
<td>Data to retrieve</td>
<td>See Note 5</td>
</tr>
<tr>
<td>Get Challenge</td>
<td>00</td>
<td>84</td>
<td>00</td>
<td>00</td>
<td>L (08)</td>
<td>8 byte challenge from card</td>
<td>See Note 6</td>
</tr>
<tr>
<td>Verify CHV</td>
<td>00</td>
<td>20</td>
<td>00</td>
<td>CHV</td>
<td>L (08)</td>
<td></td>
<td>See Note 7</td>
</tr>
<tr>
<td>Change CHV/PIN</td>
<td>00</td>
<td>24</td>
<td>00</td>
<td>CHV</td>
<td>L (10)</td>
<td></td>
<td>See Note 8</td>
</tr>
<tr>
<td>External Authenticate</td>
<td>00</td>
<td>82</td>
<td>00</td>
<td>Key #</td>
<td>L (06)</td>
<td></td>
<td>See Note 9</td>
</tr>
<tr>
<td>RSA Compute</td>
<td>80</td>
<td>42</td>
<td>00</td>
<td>Key #</td>
<td>L</td>
<td>Message to sign/decrypt</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. P1 specifies selection type: EF with short file identifier (=00), select DF (=01), EF under currently select DF (=02), parent DF (03), DF by its name. P2 specifies response required (=00) or no response required (=C0). L is data field length and depends on selection type. For short file selection, L=2. Data is short file identifier or name.
2. For secure messaging CLA=04. Data is read in plaintext but a cryptogram (3 Most Significant Bytes) is appended to it. Data is padded with 0’s. Encryption is 3DES with temporary administration key. SelFk command must be executed before.
3. X consists in the 3 Least significant Bytes of the Cryptogram. Use SelFk to get temporary administration key before command and GetResponse after command to obtain 3 Most Significant Bytes of Cryptogram to optionally verify card checksum. Data is padded with 0’s so that it is a multiple of 8. Encryption is 3DES based.
4. AI (Algorithm ID) selects DES (=00) or 3DES (=02), 512 RSA (=C5), 768 RSA (=C7), or 1024 RSA (=C9). Command is preceded by Verify CHV and followed by Get Response. No info about size of returned cryptogram.
5. Length parameter depends on previous command issued to card.
6. Number lost if card reset, unsuccessful Get Challenge, External Authenticate.
7. CHV specifies 01 to 0F, if key. Use 10 if CHV
8. CHV/PUB/Key number: 01 to 0F if Key; 10 if CHV; 11 if PUK
Key # is from 01 to 0F. Data is 6 MSB of cryptogram from standard DES/3DES encryption of random challenge.
Appendix D - GSC: Interface Definition – Card Edge for Virtual Machine Cards

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI api and amounts to the “top” part of an SPS.

A second level of interoperability is defined to allow smart cards to interoperate: the card-edge interface. This level, allows any SPS provider to interoperate with any smart card that supports the card edge interface.

This document is a Card-Edge Interface document and describes the services and the interfaces of the Generic Container Applet at the APDU level. The Generic Container Applet is already used by the Common Access Card. This card edge interface will allow application developers:
?? To make use of the existing instances of the GCA, to exploit, for example, the demographics data,
?? To operate their own instance of the GCA.

This applet card-edge specification represents an attempt to minimize the size of the code on the card while providing the required flexibility. Furthermore, the specification represents an attempt to define an interoperable feature set.

Finally it is to be noted that this specification describes a feature set that goes beyond what is required for the GSC.

Limitations
The proposed card-edge interface is an operational API and not a management API. It does not provide services like applet download or applet instantiation. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation. Furthermore, PIN management functions like Change PIN or Unlock PIN are not part of this API.
The smart card is supposed to be already initialized: applets are downloaded and instantiated, file system is created, etc.…

Establishing these limitations is balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:
-Is it a rare operation?
-Is it feasible or very difficult to define an interoperable method for this service?
The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the specification would have the result that the specification would not guaranty that all implementations would inter-operate.

**Compatibility**

This Card-Edge specification can be implemented on JAVACARD2.1, Smart Card for Windows 1.1, and MULTOS.

**Documents**

The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.

![Diagram of the document structure]

The Government Smart Card Interoperability Standard

**Presentation of the Generic Container Applet**

The Generic Container Applet offers a basic size-effective platform to provide protected data services to applications. It allows the management of 2 buffers and a rich access control to these buffers.
The intent is to allow the proxy application on the terminal to proceed to a discovery of the Access Control Rules for accessing the two buffers with a GET PROPERTIES APDU.

Then, in general after fulfilling a weak Access Control Rule, the application reads the first buffer containing a list of {Tag, Length} data items.

At this point the application knows whether the data it operates on, is managed by this Applet instance and can then proceed to access the second buffer, that contains a list of {Value} data items, to read it or update it, after fulfilling the required Access Control Rule.

In the rest of the document, the buffers are referred to as:
TAG, or T-buffer providing secure storage for the Tag and value’s length
VALUE, or V-buffer providing secure storage for the values in sequence.

To get the value associated to the tag, it is necessary to read all the {tag, Length} in the T-buffer and to compute the length of the associated value. The result gives the offset in the V-buffer to read the associated value.

This applet specification has been designed to provide an interface compatible with the Common Access Card Demographics information, the GSA J.8 data, the data needed by physical access control systems, as well as data-based extended services such as health data storage, certification and training data, rostering, property management, including the storage and retrieval of biometric templates.

The Access Control configuration options include PIN verification, external authentication, external authentication then PIN verification, external authentication or PIN, for read and update.

Two distinct Key sets are used to protect the read access and the update access.

**Next versions**
The basic platform described in this document can be enriched considerably, but this will cost space on the card. Very useful features to add would be:
Update and remove commands atomicity.
Internal TLV parsing.

Unfortunately, these “active” applet services would break the basic compatibility with file system cards, that the applet provides in the current state of the specification.

3 To start a transaction with the applet, a get_challenge APDU is sent to the applet. The applet returns a challenge. A cryptogram is computed with the challenge and the applet can then authenticate the host by verifying the cryptogram sent by the External Authenticate.
Security
The design relies heavily on a subset of Open Platform 2.01 security mechanisms, addressed by security domains, for the loading, the instantiation, and the initialization of the security parameters of the applet.
These services are not exposed by the present specification, which focuses exclusively on usage.
A function offered by an applet is called a service (for example: read the T-Buffer or update the V-buffer, or, signature, decryption for a PKI applet).
The granularity of control is the service control. That means that there is one control level by object operated by the applet and by service.
This control level is defined at the instantiation of the applet. It remains the same for all the credential life.

The different control levels are the following:
?? **Always**: the corresponding service can be provided without restrictions
?? **PIN protected**: the corresponding service can be provided only if its associated PIN code has been already verified
?? **External authenticate**: the corresponding service can be provided only after a get_challenge APDU.
?? APDU9898.
?? **External Authenticate then PIN**: the two methods must be chained successfully before access to the service is granted. This allows the authentication of the Application AND of the user.
?? **External Authenticate or PIN**: either one of the two control gives access to the service. This allows for a CardHolder validation when a PIN PAD is available and for an external authentication when no PINPAD is available. Or, this provides an Authentication method when the Application cannot be trusted to perform an external Authentication and to protect the external authentication key.
?? **Secure Channel**: the corresponding service can be provided through a Secure Channel managed by the card Open Platform layer. This control is used only for administrative operations that are not covered by this specification.
?? **Never**: the corresponding service can never be provided.

For all services, three levels of access control are defined:
?? **Level1**,
?? **Level2**.
?? **Level3**.

At applet instantiation phase, for any services provided by the applet, a level has to be chosen.

Cryptographic requirements
This applet requires from the Card Operating System the availability of the following algorithms:
DES3 ECB, double length key (16 Bytes).

**Supported APDUs**
UPDATE BUFFER (to update a buffer)
READ BUFFER (to read a buffer)
GET PROPERTIES (to retrieve applet instance properties from the card)
GET CHALLENGE (to retrieve a challenge from the card to perform a host authentication)
EXTERNAL AUTHENTICATE (to send to the card a cryptogram identifying host)
PIN VERIFY (to perform PIN verification and to check if the PIN has been already verified).

**General Error Conditions**
The following error conditions may be returned by any of the commands hereafter described:

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6200h</td>
<td>Applet or instance logically deleted</td>
</tr>
<tr>
<td>6581h</td>
<td>Memory failure</td>
</tr>
<tr>
<td>6700h</td>
<td>Incorrect parameter Lc</td>
</tr>
<tr>
<td>67LLh</td>
<td>Wrong length in Le parameter, the ‘LL’ value is expected</td>
</tr>
<tr>
<td>6982h</td>
<td>Security status not satisfied</td>
</tr>
<tr>
<td>6985h</td>
<td>Conditions of use not satisfied</td>
</tr>
<tr>
<td>6A80h</td>
<td>Invalid data in command Data Field</td>
</tr>
<tr>
<td>6A84h</td>
<td>Insufficient memory space to complete command</td>
</tr>
<tr>
<td>6A86h</td>
<td>Incorrect P1 or P2 parameter</td>
</tr>
<tr>
<td>6A88h</td>
<td>Referenced data not found</td>
</tr>
<tr>
<td>6D00h</td>
<td>Unknown instruction given in the command</td>
</tr>
<tr>
<td>6E00h</td>
<td>Wrong class given in the command</td>
</tr>
<tr>
<td>6F00h</td>
<td>Technical problem with no diagnostic given</td>
</tr>
<tr>
<td>9000h</td>
<td>Normal ending of the command</td>
</tr>
</tbody>
</table>

**Access Control Configurations**
Several security parameters can be defined according to the following table. These parameters are defined at the instantiation phase as described in the next chapter.

<table>
<thead>
<tr>
<th>Global Service</th>
<th>Always</th>
<th>PIN</th>
<th>Extern auth.</th>
<th>Extern auth. Then PIN</th>
<th>Extern Auth. or PIN</th>
<th>Secure Channel</th>
<th>Never</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Level2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
The association between the APDUs (described later in the document) and the services is accorded to this table:

<table>
<thead>
<tr>
<th>Service</th>
<th>Global service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update T-buffer</td>
<td>Level1</td>
</tr>
<tr>
<td>Read T-buffer</td>
<td>Level2</td>
</tr>
<tr>
<td>Update V-buffer</td>
<td>Level1</td>
</tr>
<tr>
<td>Read V-buffer</td>
<td>Level3</td>
</tr>
</tbody>
</table>

Only level 1 and level 3 operations can be protected by an external authentication. The authentication key as to be stored in a key set (see VOP definition of a key set). The associated key for controlling the access to level 1 operation must have a key-set version equal to 1. Also, to protect level 3 operations, the associated key needs to be stored in a key-set that has a key-set version set to 3.

For information, other APDUs have a fixed access control set to:

<table>
<thead>
<tr>
<th>Service</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Properties</td>
<td>Always</td>
</tr>
<tr>
<td>Put Key</td>
<td>Secure channel</td>
</tr>
<tr>
<td>Get challenge</td>
<td>Always</td>
</tr>
<tr>
<td>External authenticate</td>
<td>Always</td>
</tr>
<tr>
<td>Pin Verify</td>
<td>Always</td>
</tr>
</tbody>
</table>

Security levels are coded as follows in applets:

<table>
<thead>
<tr>
<th>Security level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>0</td>
</tr>
<tr>
<td>Never</td>
<td>1</td>
</tr>
<tr>
<td>External authenticate</td>
<td>2</td>
</tr>
<tr>
<td>External authenticate then PIN</td>
<td>3</td>
</tr>
<tr>
<td>Secure channel</td>
<td>4</td>
</tr>
<tr>
<td>Update Once</td>
<td>5</td>
</tr>
<tr>
<td>PIN protected</td>
<td>6</td>
</tr>
<tr>
<td>External authenticate or PIN</td>
<td>7</td>
</tr>
</tbody>
</table>
Generic Container Applet APDU interface

UPDATE BUFFER
This command allows updating a part of, or the totality of a buffer.

Command message
The Update static buffer command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>90h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>58h</td>
</tr>
<tr>
<td>P1</td>
<td>Reference Control Parameter P1</td>
</tr>
<tr>
<td>P2</td>
<td>Reference Control Parameter P2</td>
</tr>
<tr>
<td>Lc</td>
<td>Length of data + 1</td>
</tr>
<tr>
<td>Data Field</td>
<td>Buffer type + data to be updated</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Reference control parameter P1/P2
The reference control parameters P1 and P2 are used to store the offset from which data are to be written.

Data field sent in the command message
The first byte of the data field is used to indicate which buffer is to be updated. The possible values are:

?? 01h
  T-buffer
?? 02h
  V-buffer

The other bytes correspond to the data to be updated.

Response message
Data field returned in the response message
The data field in the response message is always empty.

Processing state returned in the response message

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6981h</td>
<td>No corresponding buffer</td>
</tr>
</tbody>
</table>
READ BUFFER

Introduction
This command allows reading a part of, or the totality of a buffer.

Command message
The Read buffer command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>90h (whatever the access conditions are)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>52h</td>
</tr>
<tr>
<td>P1</td>
<td>Reference Control Parameter P1</td>
</tr>
<tr>
<td>P2</td>
<td>Reference Control Parameter P2</td>
</tr>
<tr>
<td>Lc</td>
<td>01h + 01h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Buffer type + data length to read</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Reference control parameter P1/P2
The reference control parameters P1 and P2 are used to store the offset from which data are to be read.

Data field sent in the command message
The data field is used to indicate which buffer is to be read.
The possible values are:

**01h**
- T-buffer

**02h**
- V-buffer

Response message

Data field returned in the response message
The data field in the response message corresponds to the data read from the card, according to the P1 parameter.

Processing state returned in the response message

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6981h</td>
<td>No corresponding buffer</td>
</tr>
</tbody>
</table>
GET PROPERTIES

Introduction
This command is used to retrieve applet instance properties.

Command message

The Static Get properties command message is coded according the following table:

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>80h</td>
</tr>
<tr>
<td>INS</td>
<td>56h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>Expected applet instance properties length</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message
The Data fields returned in the response message contain the following properties with their current value:

?? Applet family (1 byte)
?? Applet version (4 bytes)
?? Level1 access control/ Level2 access control (1 byte)
?? Level3 access control/ RFU (1 byte)
?? RFU byte
?? RFU byte
?? ID-applet AID length (1 byte)
?? ID-applet AID (16 bytes padded with 0)
?? Key Set Version (1 byte)
?? Key Set Id (1 byte)
?? T-Buffer length (2 bytes)
?? V-Buffer length (2 bytes)
?? X bytes RFU to complement to 46 bytes

Processing state returned in the response message

See General Error Conditions in this section.
GET CHALLENGE

Command description
Each GCA applet instance can receive a Get challenge command, in order to perform a host authentication in the following sent command.
This command has to be sent by host just before a External authenticate command, and corresponds to the first step of host authentication before sending a command having an external authenticate security level.
The computed challenge is valid only in the APDU following the GET CHALLENGE APDU.

Command message
The Get challenge command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>84h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>Challenge length (has to be 8 bytes)</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message
The response message contains the challenge used later for authentication.

This challenge has to be memorized inside the applet instance, in order to calculate the corresponding response. If the response is not sent (using External Authenticate command) in the command following the Get Challenge command, the calculated challenge is then lost.

Processing state returned in the response message

See General Error Conditions in this section.
EXTERNAL AUTHENTICATE

Command description
Just after receiving a card challenge from Get challenge command, a cryptogram is computed by the host using the 8-Bytes card challenge, and a special 16-Bytes 3DES key known by the host and the static applet (with the PUTKEY command). The cryptogram is the result of the 3DES algorithm with the appropriate key and the challenge as the input.

The key used to protect the level 1 operation has a key set version set to 1. The key used to protect the level 3 operation has a key set version set to 3.

The version of the key set must be specified in the command so the applet can know which keyset to use.

The corresponding response is then sent to the card using External Authenticate command.

This command has to be sent by host just before a command with an External authentication security level, and correspond to the second step of host authentication. Successful cryptogram verification is mandatory to perform the following command.

If the command is not sent just after the Get challenge command, card challenge is then lost.

The External authentication is only valid while still accessing the same security level. If the applet receives an APDU with a different security level, the access level is lost. This means that if the card is removed or if the applet is unselected, the access level is lost.

Command message
The external authenticate command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>42h</td>
</tr>
<tr>
<td>P1</td>
<td>Keyset version</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Host cryptogram</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
Host cryptogram.
Response message
Success: 0x9000
Access denied: 0x6982

Data field returned in the response message

The data field returned is always empty

Processing state returned in the response message

See General Error Conditions in this section.

PIN VERIFY

Commands description
Each GC applet instance can receive a PIN Verify command, in order to verify a PIN code, or to check if a PIN code has been already verified or not. When receiving those commands, the selected instance forwards the information to the ID applet instance containing the PIN number specified in the command, through its shared functions.

Command message
The PIN Verify command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>20h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h or 08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>PIN code to be verified</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
If the data field sent in the command message does not include a PIN code, the command is corresponding to a PIN verify check command, in order to know if the PIN code has been already verified or not.

If the data field includes the PIN code to be verified, the PIN code value is corresponding to the 8 first bytes of the data field, coded as described in the 3.1.2.4 chapter.

- If the verification fails, the left PIN tries flag is decremented, and the PIN verified flag value does not change. The PIN always flag value is set to 00h. If the left PIN tries flag value is 00h, the PIN code is considered as locked.
- If the verification succeeds, the PIN verified flag value, and the PIN always flag value are set to 01h.
If \( P2 = 01h \), the AID of the ID applet instance containing the PIN code to be verified or checked is stored just after the PIN code to verify, or at the beginning of the data field if the command corresponds to a **PIN verified check** command.

**Response message**

**Data field returned in the response message**

The data field in the response message is always empty.

**Processing state returned in the response message**

If PIN verification or PIN verified checking fails, the status code returned is \( SW1 = 63h \), \( SW2 = \) left PIN tries.

\( SW2 = FFh \) means infinite tries.

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63LLh</td>
<td>PIN verify rejected and left PIN tries are ‘LL’</td>
</tr>
<tr>
<td>6981h</td>
<td>No PIN code defined</td>
</tr>
<tr>
<td>6983h</td>
<td>PIN code blocked</td>
</tr>
</tbody>
</table>
Appendix E - GSC: Interface Definition - SKI for VM Cards

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI api and amounts to the “top” part of an SPS.

A second level of interoperability is defined to allow smart cards to interoperate: the card-edge interface. This level, allows any SPS provider to interoperate with any smart card that supports the card edge interface.

This document is a Card-Edge Interface document and describes the services and the interfaces of the SKI Applet at the APDU level.

This applet card-edge specification represents an attempt to minimize the size of the code on the card while providing the required flexibility. Furthermore, the specification is voluntarily reduced to the kind of features that can be also expected from a card without a VM, and represents an attempt to define a feature set interoperable with this type of cards, while not sacrificing the efficiency of the VM card.

Limitations
The proposed card-edge interface is an operational API and not a management API. It does not provide services like applet download or applet instantiation. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation.
PIN management functions like Change PIN or Unlock PIN are not part of this API. The smart card is supposed to be already initialized: applets are downloaded and instantiated, and the file system is created.

Establishing these limitations is a balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder, or the generation of a key pair. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:
- Is it a rare operation?
- Is it feasible or very difficult to define an interoperable method for this service?

The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the specification would have the result that the specification would not guaranty that all implementations would inter-operate.
Compatibility
This Card-Edge specification can be implemented on JAVACARD2.1, Smart Card for Windows 1.1, and MULTOS.

Documents
The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.

The Government Smart Card Interoperability Standard

Presentation of the SKI Applet
This applet is responsible for DES and 3DES based operations. It can be used to perform user authentication based on a shared secret key.
Security
The design relies heavily on a subset of Open Platform 2.0 1.b security mechanisms, addressed by security domains, for the loading, the instantiation, and the initialization of the security parameters of the applet. These services are not exposed by the present specification, which focuses exclusively on usage.

A function offered by an applet is called a service (for example: signing a message). The granularity of control is the service control. That means that there is one control level by object operated by the applet and by service.

This control level is defined at the instantiation of the applet. It remains the same for all the credential life.

The different control levels are the following:
- **Always**: the corresponding service can be provided without restrictions
- **PIN protected**: the corresponding service can be provided only if its associated PIN code has been already verified
- **External authenticate**: the corresponding service can be provided only after a get_challenge APDU.
- **Never**: the corresponding service can never be provided.

For all services, three levels of access control are defined:
- Level1,
- Level2,
- Level3.

At applet instantiation phase, for any services provided by the applet, a level has to be chosen.

**Cryptographic requirements**
This applet requires from the Card Operating System the availability of the following algorithms:
- DES ECB, 8 bytes key.
- DES3 ECB, double length key (16 Bytes).

**Supported APDUs**
- INTERNAL AUTHENTICATE (to sign a challenge with the authentication key)
- GET PROPERTIES (to retrieve applet instance properties from the card)
- GET CHALLENGE (to retrieve a challenge from the card to perform a host authentication)
- EXTERNAL AUTHENTICATE (to send to the card a cryptogram authenticating the host)
- PIN VERIFY (to perform PIN verification and to check if the PIN has been already verified).
General Error Conditions
The following error conditions may be returned by any of the commands hereafter described:

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6200h</td>
<td>Applet or instance logically deleted</td>
</tr>
<tr>
<td>6581h</td>
<td>Memory failure</td>
</tr>
<tr>
<td>6700h</td>
<td>Incorrect parameter Lc</td>
</tr>
<tr>
<td>67LLh</td>
<td>Wrong length in Le parameter, the ‘LL’ value is expected</td>
</tr>
<tr>
<td>6982h</td>
<td>Security status not satisfied</td>
</tr>
<tr>
<td>6985h</td>
<td>Conditions of use not satisfied</td>
</tr>
<tr>
<td>6A80h</td>
<td>Invalid data in command Data Field</td>
</tr>
<tr>
<td>6A84h</td>
<td>Insufficient memory space to complete command</td>
</tr>
<tr>
<td>6A86h</td>
<td>Incorrect P1 or P2 parameter</td>
</tr>
<tr>
<td>6A88h</td>
<td>Referenced data not found</td>
</tr>
<tr>
<td>6D00h</td>
<td>Unknown instruction given in the command</td>
</tr>
<tr>
<td>6E00h</td>
<td>Wrong class given in the command</td>
</tr>
<tr>
<td>6F00h</td>
<td>Technical problem with no diagnostic given</td>
</tr>
<tr>
<td>9000h</td>
<td>Normal ending of the command</td>
</tr>
</tbody>
</table>

Access control configurations
Several security parameters can be defined according to the following table. These parameters are defined at the instantiation phase as described in the next chapter.

<table>
<thead>
<tr>
<th>Global Service</th>
<th>Alway s</th>
<th>PIN</th>
<th>Extern auth.</th>
<th>Never</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Level2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The association between the APDU (s described later in the document) and the services is accorded to this table:

<table>
<thead>
<tr>
<th>Service</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Authenticate</td>
<td>Level 1</td>
</tr>
<tr>
<td>RFU</td>
<td>Level 2</td>
</tr>
<tr>
<td>RFU</td>
<td>Level 3</td>
</tr>
</tbody>
</table>

For information, other APDU have a fixed access control set to:
<table>
<thead>
<tr>
<th>Service</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Properties</td>
<td>Always</td>
</tr>
<tr>
<td>Put Key</td>
<td>Secure channel</td>
</tr>
<tr>
<td>Get challenge</td>
<td>Always</td>
</tr>
<tr>
<td>External authenticate</td>
<td>Always</td>
</tr>
<tr>
<td>Pin Verify</td>
<td>Always</td>
</tr>
</tbody>
</table>

Security levels are coded as follows in applets:

<table>
<thead>
<tr>
<th>Security level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>0</td>
</tr>
<tr>
<td>Never</td>
<td>1</td>
</tr>
<tr>
<td>External authenticate</td>
<td>2</td>
</tr>
<tr>
<td>PIN protected</td>
<td>6</td>
</tr>
</tbody>
</table>
SKI Applet APDU interface

INTERNAL AUTHENTICATE

Introduction
This command is used to perform a RSA signature or data decryption.

Command message
The Internal authenticate command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h (whatever the access control rules are)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>42h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>Challenge length (retrieved from a Get properties)</td>
</tr>
<tr>
<td>Data Field</td>
<td>Challenge</td>
</tr>
<tr>
<td>Le</td>
<td>00h</td>
</tr>
</tbody>
</table>

Data field sent in the command message
The data field contains the data to be signed using the selected key.

Response message

Data field returned in the response message
The data field in the response message contains the data signed. The length of the response may vary and depends on the configuration of the applet.

Processing state returned in the response message
See General Error Conditions in this section.
GET PROPERTIES

Introduction
This command is used to retrieve applet instance properties.

Command message

The Static Get properties command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>56h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Le</td>
<td>00h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>Expected applet instance properties length</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message
The Data fields returned in the response message contain the following properties with their current value:

?? Applet family (1 byte)
?? Applet version (4 bytes)
?? Level1 access control/ Level2 access control (1 byte)
?? Level3 access control/ RFU (1 byte)
?? RFU byte
?? RFU byte
?? ID-applet AID length
?? ID-applet AID (always on 16 bytes, padded with 0 if necessary)
?? Key Set (1 byte)
?? Key Id (1 byte)
?? Algo ID (1 byte)
?? Challenge length in bytes (1 byte)
?? Response length in bytes (1 byte)
?? RFU (x bytes to complement to 46)

Processing state returned in the response message

See General Error Conditions in this section.
GET CHALLENGE

Command description
Each applet instance can receive a **Get challenge** command, in order to perform a host authentication in the following sent command. This command has to be sent by host just before a **External authenticate** command, and corresponds to the first step of host authentication before sending a command having an external authenticate security level. The computed challenge is valid only in the APDU following the GET CHALLENGE APDU.

Command message
The **Get challenge** command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>84h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Le</td>
<td>08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>challenge length (has to be 8 bytes)</td>
</tr>
</tbody>
</table>

Response message

*Data field returned in the response message*

This challenge has to be memorized inside the applet instance, in order to calculate the corresponding response. If the response is not sent (using **External Authenticate** command) in the command following the **Get Challenge** command, the calculated challenge is then **lost**.

*Processing state returned in the response message*

*See General Error Conditions in this section.*
EXTERNAL AUTHENTICATE

Command description
Just after receiving a card challenge from Get challenge command, a cryptogram is computed by the host using the 8-Bytes card challenge, and a special 16-Bytes 3DES key known by the host and the static applet (with the PUTKEY command). The cryptogram is the result of the 3DES algorithm with the appropriate key and the challenge as the input.

The key used to protect the level 1 operation has a key set version set to 1. The key used to protect the level 3 operation has a key set version set to 3.

The corresponding response is then sent to the card using External Authenticate command.

This command has to be sent by host just before a command with an External authentication security level, and correspond to the second step of host authentication. Successful cryptogram verification is mandatory to perform the following command.

If the command is not sent just after the Get challenge command, card challenge is then lost.

The External authentication is only valid while still accessing the same security level. If the applet receives an APDU with a different security level, the access level is lost. This means that if the card is removed or if the applet is unselected, the access level is lost.

Command message
The external authenticate command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>42h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Host cryptogram</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
Host cryptogram.
Response message
Success: 0x9000
Access denied: 0x6982

Data field returned in the response message
The data field returned is always empty

Processing state returned in the response message

See General Error Conditions in this section.

PIN VERIFY

Commands description
Each GC applet instance can receive a PIN Verify command, in order to verify a PIN code, or to check if a PIN code has been already verified or not.
When receiving those commands, the selected instance forwards the information to the ID applet instance containing the PIN number specified in the command, through its shared functions.

Command message
The PIN Verify command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>20h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h or 08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>PIN code to be verified</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
If the data field sent in the command message does not include a PIN code, the command is corresponding to a PIN verify check command, in order to know if the PIN code has been already verified or not.
If the data field includes the PIN code to be verified, the PIN code value is corresponding to the 8 first bytes of the data field, coded as described in the 3.1.2.4 chapter.

- If the verification fails, the left PIN tries flag is decremented, and the PIN verified flag value does not change. The PIN always flag value is set to 00h. If the left PIN tries flag value is 00h, the PIN code is considered as locked.
- If the verification succeeds, the PIN verified flag value, and the PIN always flag value are set to 01h.
If P2 = 01h, the AID of the ID applet instance containing the PIN code to be verified or checked is stored just after the PIN code to verify, or at the beginning of the data field if the command corresponds to a **PIN verified check** command.

**Response message**

*Data field returned in the response message*

The data field in the response message is always empty.

*Processing state returned in the response message*

If PIN verification or PIN verified checking fails, the status code returned is SW1 = 63h, SW2 = left PIN tries.

SW2 = FFh means infinite tries.

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63LLh</td>
<td>PIN verify rejected and left PIN tries are ‘LL’</td>
</tr>
<tr>
<td>6981h</td>
<td>No PIN code defined</td>
</tr>
<tr>
<td>6983h</td>
<td>PIN code blocked</td>
</tr>
</tbody>
</table>
Appendix F - GSC: Interface Definition – PKI for VM Cards

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

A first level of interoperability is defined to protect the Application consuming smart cards from the need to know about any specific smart card. This level is achieved by the BSI api and amounts to the “top” part of an SPS.

A second level of interoperability is defined to allow smart cards to interoperate: the card-edge interface. This level, allows any SPS provider to interoperate with any smart card that supports the card edge interface.

This document, a Card-Edge interface specification, describes the services and the interfaces of the PKI Applet at the APDU level. The PKI Applet is already used by the Common Access Card. This card edge interface will allow application developers:

?? To make use of the existing instances of the PKI applet.
?? To operate their own instance of a PKI applet.

This applet card-edge specification represents an attempt to minimize the size of the code on the card while providing the required flexibility. Furthermore, the specification is voluntarily reduced to the kind of features that can be also expected from a card without a VM, and represents an attempt to define a feature set interoperable with this type of cards, while not sacrificing the efficiency of the VM card.

Limitations
The proposed card-edge interface is an operational API and not a management API. It does not provide services like applet download or applet instantiation. It does not allow the changing of access conditions associated with each instance or each file, since access conditions are defined during the container creation.

PIN management functions like Change PIN or Unlock PIN are not part of this API.

The key generation command is also not part of this version of the specification, and this is also true for the key injection.

The smart card is supposed to be already initialized: applets are downloaded and instantiated, and the file system is created.

Establishing these limitations is a balancing act since some services are in the gray area between usage and management, like PIN change by the cardholder, or the generation of a key pair. Two criteria help to make a decision whether the service should be excluded from the interoperability specification or not:

-Is it a rare operation?
-Is it feasible or very difficult to define an interoperable method for this service?

The fact that a service is excluded from the specification does not mean it is not required from a smart card system, it means only that for this service interoperability is not a requirement, or that it cannot be technically achieved, so that putting it in the
specification would have the result that the specification would not guaranty that all implementations would inter-operate.

**Compatibility**
This Card-Edge specification can be implemented on **JAVACARD2.1, Smart Card for Windows 1.1, and MULTOS.**

**Documents**
The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.

---

**The Government Smart Card Interoperability Standard**

**Presentation of the PKI Applet**
This applet is responsible for RSA based operations. It can be used to perform RSA signature as well as to retrieve the certificate associated to the key instance.
The RSA key can be whether internally generated in the card or securely injected using the secure channel provided by the VOP ‘Security Domain’.

RSA operations are performed in the ‘raw’ mode. Therefore, the signature than decryption operations do not differ.

**Security**
The design relies heavily on a subset of Open Platform 2.0 1.b security mechanisms, addressed by security domains, for the loading, the instantiation, and the initialization of the security parameters of the applet. These services are not exposed by the present specification, which focuses exclusively on usage.

A function offered by an applet is called a service (for example: singing a message). The granularity of control is the service control. That means that there is one control level by object operated by the applet and by service. This control level is defined at the instantiation of the applet. It remains the same for all the credential life.

The different control levels are the following:
- **Always**: the corresponding service can be provided without restrictions
- **PIN protected**: the corresponding service can be provided only if its associated PIN code has been already verified
- **External authenticate**: the corresponding service can be provided only after a get_challenge APDU.
- **Never**: the corresponding service can never be provided.

For all services, three levels of access control are defined:
- Level1,
- Level2,
- Level3.

At applet instantiation phase, for any services provided by the applet, a level has to be chosen.

**Cryptographic requirements**
This applet requires from the Card Operating System the availability of the following algorithms:
- DES3 ECB, double length key (16 Bytes).
- DES3 CBC,
- RSA key generation (length = 512b, 768b, 1024b)
- RSA algorithm without padding (length = 512b, 768b, 1024b)

For RSA signature and decryption, it is assumed that the padding will be performed outside the card.
Supported APDUs
PRIVATE SIGN/DECYPRT (to sign or decrypt data using RSA private key)
GET PROPERTIES (to retrieve applet instance properties from the card)
GET CHALLENGE (to retrieve a challenge from the card to perform a host authentication)
EXTERNAL AUTHENTICATE (to send to the card a cryptogram authenticating the host)
PIN VERIFY (to perform PIN verification and to check if the PIN has been already verified).

General Error Conditions
The following error conditions may be returned by any of the commands hereafter described:

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6200h</td>
<td>Applet or instance logically deleted</td>
</tr>
<tr>
<td>6581h</td>
<td>Memory failure</td>
</tr>
<tr>
<td>6700h</td>
<td>Incorrect parameter Lc</td>
</tr>
<tr>
<td>67LLh</td>
<td>Wrong length in Le parameter, the ‘LL’ value is expected</td>
</tr>
<tr>
<td>6982h</td>
<td>Security status not satisfied</td>
</tr>
<tr>
<td>6985h</td>
<td>Conditions of use not satisfied</td>
</tr>
<tr>
<td>6A80h</td>
<td>Invalid data in command Data Field</td>
</tr>
<tr>
<td>6A84h</td>
<td>Insufficient memory space to complete command</td>
</tr>
<tr>
<td>6A86h</td>
<td>Incorrect P1 or P2 parameter</td>
</tr>
<tr>
<td>6A88h</td>
<td>Referenced data not found</td>
</tr>
<tr>
<td>6D00h</td>
<td>Unknown instruction given in the command</td>
</tr>
<tr>
<td>6E00h</td>
<td>Wrong class given in the command</td>
</tr>
<tr>
<td>6F00h</td>
<td>Technical problem with no diagnostic given</td>
</tr>
<tr>
<td>9000h</td>
<td>Normal ending of the command</td>
</tr>
</tbody>
</table>

Access Control Configurations
Several security parameters can be defined according to the following table. These parameters are defined at the instantiation phase as described in the next chapter.

<table>
<thead>
<tr>
<th>Global Service</th>
<th>Always</th>
<th>PIN</th>
<th>Extern auth.</th>
<th>Never</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Level2</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Level3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The association between the APDUs (described later in the document) and the services is accorded to this table:
<table>
<thead>
<tr>
<th>Operation</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate key</td>
<td>Level 1</td>
</tr>
<tr>
<td>Private sign &amp; decrypt</td>
<td>Level 2</td>
</tr>
<tr>
<td>Read certificate</td>
<td>Level 3</td>
</tr>
</tbody>
</table>

Only level 1 and level 3 operations can be protected by an external authentication. The authentication key as to be stored in a key set (see VOP definition of a key set). The associated key for controlling the access to level 1 operation must have a key-set version equal to 1. Also, to protect level 3 operations, the associated key needs to be stored in a key-set that has a key-set version set to 3.

For information, other APDUs have a fixed access control set to:

<table>
<thead>
<tr>
<th>Service</th>
<th>Security level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Properties</td>
<td>Always</td>
</tr>
<tr>
<td>Put Key</td>
<td>Secure channel</td>
</tr>
<tr>
<td>Get challenge</td>
<td>Always</td>
</tr>
<tr>
<td>External authenticate</td>
<td>Always</td>
</tr>
<tr>
<td>Pin Verify</td>
<td>Always</td>
</tr>
</tbody>
</table>

Security levels are coded as follows in applets:

<table>
<thead>
<tr>
<th>Security level</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>0</td>
</tr>
<tr>
<td>Never</td>
<td>1</td>
</tr>
<tr>
<td>External authenticate</td>
<td>2</td>
</tr>
<tr>
<td>PIN protected</td>
<td>6</td>
</tr>
</tbody>
</table>
PKI Applet APDU Interface

PRIVATE SIGN/DECRYPT

Introduction
This command is used to perform a RSA signature or data decryption.

Command message
The Private sign/decrypt command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h  (whatever the access control rules are)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>42h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>Data Field length (modulus length)</td>
</tr>
<tr>
<td>Data Field</td>
<td>Data to sign</td>
</tr>
<tr>
<td>Le</td>
<td>Expected length of the signature/decryption</td>
</tr>
</tbody>
</table>

Data field sent in the command message
The data field contains the data to be signed using the selected RSA key pair.

The data have to be already padded according to the standard used (PKCS#1 for example), before the message is sent. The message is never padded in the card.

Response message

Data field returned in the response message
The data field in the response message contains the data signed or decrypted.

Processing state returned in the response message

See General Error Conditions in this section.
GET CERTIFICATE

Introduction
This command is used to retrieve the certificate associated to a RSA key pair.

This is operational if the certificate has been stored in a GCA instance where the AID is 0xA00000007901FEh.

Command message
The Get Certificate command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h (whatever the access control rules are)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>36h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Le</td>
<td>00</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>00h</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message

The data field in the response message contains 255 bytes of certificate (or less). The host application determines if there is more data to read depending on the Status word filled by the applet. 9000h means the complete certificate has been read, 6310h means that more data is available.

Processing state returned in the response message

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6981h</td>
<td>No corresponding certificate</td>
</tr>
<tr>
<td>6310h</td>
<td>More data is available</td>
</tr>
</tbody>
</table>
GET PROPERTIES

Introduction
This command is used to retrieve applet instance properties.

Command message

The Static Get properties command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>56h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>Expected applet instance properties length</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message

The Data fields returned in the response message contain the following properties with their current value:

?? Applet family (1 byte)
?? Applet version (4 bytes)
?? Level1 access control/ Level2 access control (1 byte)
?? Level3 access control/ RFU (1 byte)
?? RFU byte
?? RFU byte
?? ID-applet AID length
?? ID-applet AID (always on 16 bytes, padded with 0 if necessary)
?? Key Set (1 byte)
?? Key Id (1 byte)
?? Key length (1 bytes)
?? RFU (x bytes to complement to 46)

Processing state returned in the response message

See General Error Conditions in this section.
GET CHALLENGE

Command description
Each applet instance can receive a Get challenge command, in order to perform a host authentication in the following sent command. This command has to be sent by host just before a External authenticate command, and corresponds to the first step of host authentication before sending a command having an external authenticate security level. The computed challenge is valid only in the APDU following the GET CHALLENGE APDU.

Command message
The Get challenge command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>84h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Le</td>
<td>08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Empty</td>
</tr>
<tr>
<td>Le</td>
<td>challenge length (has to be 8 bytes)</td>
</tr>
</tbody>
</table>

Response message

Data field returned in the response message
This challenge has to be memorized inside the applet instance, in order to calculate the corresponding response. If the response is not sent (using External Authenticate command) in the command following the Get Challenge command, the calculated challenge is then lost.

Processing state returned in the response message

See General Error Conditions in this section.
EXTERNAL AUTHENTICATE

Command description
Just after receiving a card challenge from Get challenge command, a cryptogram is computed by the host using the 8-Bytes card challenge, and a special 16-Bytes 3DES key known by the host and the static applet (with the PUTKEY command). The cryptogram is the result of the 3DES algorithm with the appropriate key and the challenge as the input.

The key used to protect the level 1 operation has a key set version set to 1.
The key used to protect the level 3 operation has a key set version set to 3.

The corresponding response is then sent to the card using External Authenticate command.

This command has to be sent by host just before a command with an External authentication security level, and correspond to the second step of host authentication. Successful cryptogram verification is mandatory to perform the following command.

If the command is not sent just after the Get challenge command, card challenge is then lost.

The External authentication is only valid while still accessing the same security level. If the applet receives an APDU with a different security level, the access level is lost. This means that if the card is removed or if the applet is unselected, the access level is lost.

Command message
The external authenticate command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>42h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>Host cryptogram</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
Host cryptogram.
Response message
Success: 0x9000
Access denied: 0x6982

Data field returned in the response message
The data field returned is always empty

Processing state returned in the response message

See General Error Conditions in this section.

PIN VERIFY

Commands description
Each GC applet instance can receive a PIN Verify command, in order to verify a PIN code, or to check if a PIN code has been already verified or not.
When receiving those commands, the selected instance forwards the information to the ID applet instance containing the PIN number specified in the command, through its shared functions.

Command message
The PIN Verify command message is coded according the following table:

<table>
<thead>
<tr>
<th>CLA</th>
<th>80h</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>20h</td>
</tr>
<tr>
<td>P1</td>
<td>00h</td>
</tr>
<tr>
<td>P2</td>
<td>00h</td>
</tr>
<tr>
<td>Lc</td>
<td>00h or 08h</td>
</tr>
<tr>
<td>Data Field</td>
<td>PIN code to be verified</td>
</tr>
<tr>
<td>Le</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Data field sent in the command message
If the data field sent in the command message does not include a PIN code, the command is corresponding to a PIN verify check command, in order to know if the PIN code has been already verified or not.

If the data field includes the PIN code to be verified, the PIN code value is corresponding to the 8 first bytes of the data field, coded as described in the 3.1.2.4 chapter.

- If the verification fails, the left PIN tries flag is decremented, and the PIN verified flag value does not change. The PIN always flag value is set to 00h. If the left PIN tries flag value is 00h, the PIN code is considered as locked.
- If the verification succeeds, the PIN verified flag value, and the PIN always flag value are set to 01h.
If \( P2 = 01h \), the AID of the ID applet instance containing the PIN code to be verified or checked is stored just after the PIN code to verify, or at the beginning of the data field if the command corresponds to a **PIN verified check** command.

**Response message**

*Data field returned in the response message*

The data field in the response message is always empty.

*Processing state returned in the response message*

If PIN verification or PIN verified checking fails, the status code returned is \( SW1 = 63h, \) \( SW2 = \) left PIN tries. 
\( SW2 = FFh \) means infinite tries.

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63LLh</td>
<td>PIN verify rejected and left PIN tries are ‘LL’</td>
</tr>
<tr>
<td>6981h</td>
<td>No PIN code defined</td>
</tr>
<tr>
<td>6983h</td>
<td>PIN code blocked</td>
</tr>
</tbody>
</table>
Appendix G - GSC: SPM Test Requirements & Procedures

Scope
This document is part of the effort led by the GSA to achieve the definition of an interoperability standard for Basic Services offered by the Government Smart Cards.

Limitations
The goal of this specification is to achieve the interoperability of smart card subsystems for the following Basic Services:
Secure Storage and retrieval of J.8 data
Basic cryptographic services
Two main levels of specification have been addressed:
A provider level with the BSI API: this API protects the applications from the specifics of the card subsystems (SPM).
A Card-edge level: this interface provides the interoperability at the card level.

The present requirement document shall focus on the interoperability features of these two levels of specification. An interesting side effect of the testing procedures is the de-facto validation of the lower layers.

Documents
The following picture presents the tree of documents that define the standard. The outlined square defines the present document and its position within this tree.
Objectives & Expected Outcomes of the Interoperability testing
A smart card subsystem is constituted of a number of loosely coupled software and hardware components which interoperability must be proven before they can be deployed list as 5 bullets

?? BSI provider with a BSI API
?? a card edge SPI
?? a smart card
?? a reader driver
?? a reader

The purpose of the testing effort is not general Quality Insurance, it is interoperability: bugs are not explicitly looked for. Nevertheless, bugs prevent the system to “operate”, ergo, to “interoperate” and shall therefore constitute a cause for failing a system to pass the interoperability testing.

Requirements
Reader/Card interface
It has been an observation that on occasion, some smart card readers (IFDs) fail in communications with some smart card devices, despite the putative compliance of both with the ISO 7816 standards. It is possible for two separate smart card solution systems, designed and successfully tested under the GSA Government Smart Cards interoperability standards, to have internal card-reader compatibility and still fail in intersystem interoperability because the problems between the cards of one system and the readers of the other. It has therefore been suggested that card-reader compatibility be included in the GSA Interoperability Test Plan.

It has been the experience of some that smart card incompatibility with smart card readers has revealed problems in two critical areas:

?? Inability of the reader to process the card ATR Interface Characters, TA(1) to TD(n), to establish a communication protocol.

?? Timing issues, especially reader timeouts in command/response transmissions between the reader and the card.

One proposed test plan for card-reader compatibility is as follows:

Test Objectives
The purpose of this test is to screen for failures in smart card communications with smart card IFDs, focusing on the initial communication protocol and card-reader timing, then any additional designated critical areas.

Test Method

?? Test Case 1: Card-reader communication protocol in response to the card ATR at card power-up.

Step 1: When a card is initially powered up the response should be checked for the correct ATR, and any error codes returned by the reader should be checked (if the reader supports this feature)
Expected Result: The card ATR is returned correctly, in its entirety, and the reader returns no error codes.

Step 2: One or more standard, non-destructive command, such as the Select File command, should be sent to the card. These commands can be selected from the card-specific APDU information used in card interoperability testing.
Expected Result: For each command, the card returns a successful response, and the reader returns no error codes.

Success Measure: The results for all steps in the test case are successful.

?? Test Case 2: Timeout failures during card-reader command-response exchange.

Step 1: One or more time-intensive command, such as Generate Key Pair for cards supporting PKI, or Read Binary for a large block, should be sent to the card.
These commands can be selected from the card-specific APDU information used in card interoperability testing.

**Expected Result:** For each command, the card returns a successful response, and the reader returns no error codes.

**Success Measure:** The results for all steps in the test case are successful.

**Test Setup**

All smart cards currently under consideration for testing should be tested in the designated central testing facility against the battery of readers selected by the Primes for evaluation for compatibility. A fairly simple test tool designed to use APDU scripts, derived from the card-specific APDU information used in card interoperability testing, along with the PCSC support designated for use with the reader. It is important that Java cards be loaded with the same applets to be used in the actual smart card solution, as timeout issues can be highly applet-specific. If embedded systems wish to be evaluated for card-reader compatibility, further modification of the test platform will be necessary.

**Card Edge testing**

The objective is to test the compliance of the card to the card-edge interoperability specification.

**Card Edge Testing platform: provided by the GSA**

Windows2000 workstations with the interfaces required for reader connectivity:
- Serial, USB, PC-card, parallel.
It should be noted that readers are required to interoperate with one another on the same workstation. Therefore, once reader has been validated, its drivers must be uninstalled.

**Card Edge Test Kit: provided by the primes**

5 cards per card type must be provided. These cards must have been personalized, which means:

- The PIN has been set
- The Capability container has been initialized on the card according to the specification (the capability container is only needed for cards that do not support the default APDU set)
- A simple APDU script allowing to read the whole capability file is provided.
- The J.8 containers have been initialized according to the specification
- The SKI keys have been initialized and are provided.
- The PKI key pair has been initialized.
- The certificate is present on the card.

**Card Edge Test requirements for File system cards**

A testing sample that satisfies the following requirements must be provided by the GSA:

- Read the Capability file
- Parse the capability file and verify its internal format: TLV
- Use capability directives to access to the J.8 containers.
- Verify CHV possibly using a corresponding capability
Card Edge Test requirements for JavaCards

A testing sample that satisfies the following requirements must be provided by the GSA:

- Read the Capability Container if it exists. (In the case of a VM card, the capability is only needed for addressing the differences in the way to implement the access control rules to J.8).
- Parse the capability container and verify its internal format: TLV
- Read the J.8 containers, possibly using the information extracted from the capability container
- Verify CHV
- Read the J.8 containers: verify internal TLV format, as well as CRC and LRC.
- Proceed to an SKI internal authentication using challenge/response
- Proceed to an SKI external authentication using challenge/response
- Get certificate and extract the public key
- Do an RSA compute: for example sign a message and verify the cryptogram with the public key.

BSI Testing

The objective is to test the compliance of the BSI to the interoperability specification. The BSI is the most complex part of the SPM to test since it is composed of 2 interfaces:

- The BSI API, which is the interface the provider exposes to applications.
- The Card-Edge, which the SPI the provider exposes to cards.

Unfortunately it is hardly possible the testing of the 2 interfaces.

BSI testing platform: provided by the GSA

This version of the test requirement focuses on WINTEL platforms. The BSI provider shall be tested on the following platforms and be certified on the platform where it succeeded:

- Windows95/98 with the PC/SC components
- NT4 SP6 with the PC/SC components
- Windows2000

The testing procedure must be run on each platform.

A collection of cards with a tested and validated card-edge are required [these cards are personalized cards]. There should be at least of File System Card and one VM card.

A collection of readers with validate drivers and operation validated with the cards above must be provided.

BSI Test Kit: provided by the primes
BSI Test Requirements
A testing sample that exercises the calls of the BSI must be provided by the GSA. For any provider to be validated, the sample must be run on at least to different types of cards (Javacard and file system cards), preferably on the collection of cards that have passed card-edge. If a service is not provided by a card, the provider must gracefully return “service unavailable”.

```c
void gscBsiUtilGetReaderList()
```
Retrieves the list of configured readers. The support of only one reader is required for the tests.

```c
void gscBsiUtilGetCardStatus()
```
Retrieves card presence for a connection handle or a reader. Test extraction and insertion of a card.

```c
void gscBsiUtilCardConnect()
```
Connect to the card, using the reader name the card is inserted in.

```c
void gscBsiUtilCardDisconnect()
```
Disconnect to the card

```c
void gscBsiUtilGetCardProperties()
```
Retrieve card dependant data (Serial number)

```c
void gscBsiUtilGetVersion()
```
Retrieve the version of the provider.

```c
void gscBsiUtilPassthru()
```
Allows to send an APDU to the card or the applet and get the answer from the card. Use it to read the Capability file.

```c
void gscBsiUtilAcquireContext()
```
Establishes the Security Context required by the command, as discovered using the appropriate get_properties function. **Only PIN verification (CHV) is required for testing.**

```c
void gscBsiUtilReleaseContext()
```
Releases the Security Context previously established.
gscBsiGcDataCreate()
Create a new data item in the selected container. This will store a value and a Tag.

gscBsiGcDataDelete()
Delete a data in the selected container.

gscBsiGcReadTagList()
Read the list of Tags in the selected container.


gscBsiGcReadValue()
Retrieve the current value of a given Tag in the selected container.


gscBsiGcUpdateValue()
Update the current value of a given Tag with the provided value.


gscBsiGetChallenge()
Retrieve a challenge from the card.


gscBsiSkiInternalAuthenticate()
Compute a symmetric key cryptography authenticator in response to a challenge.


gscBsiPkiCompute()
Compute the private key encrypt/decrypt. The mandatory PKI algorithm of the BSI is RSA_NO_PAD.


gscBsiPkiReadCertificate()
Read the certificate
## Appendix H - Revised Section J.8, GSA Common Data Model

<table>
<thead>
<tr>
<th>General Information</th>
<th>Type</th>
<th>Max. Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>Variable</td>
<td>20</td>
</tr>
<tr>
<td>Middle Name</td>
<td>Variable</td>
<td>20</td>
</tr>
<tr>
<td>Last Name</td>
<td>Variable</td>
<td>20</td>
</tr>
<tr>
<td>Suffix</td>
<td>Variable</td>
<td>4</td>
</tr>
<tr>
<td>Government Agency</td>
<td>Variable</td>
<td>30</td>
</tr>
<tr>
<td>Bureau Name</td>
<td>Variable</td>
<td>30</td>
</tr>
<tr>
<td>Agency Bureau Code</td>
<td>Variable</td>
<td>4</td>
</tr>
<tr>
<td>Department Code</td>
<td>Variable</td>
<td>4</td>
</tr>
<tr>
<td>Position/Title</td>
<td>Variable</td>
<td>30</td>
</tr>
<tr>
<td>Building Name</td>
<td>Variable</td>
<td>30</td>
</tr>
<tr>
<td>Office Address 1</td>
<td>Variable</td>
<td>60</td>
</tr>
<tr>
<td>Office Address 2</td>
<td>Variable</td>
<td>60</td>
</tr>
<tr>
<td>Office City</td>
<td>Variable</td>
<td>50</td>
</tr>
<tr>
<td>Office State</td>
<td>Fixed Text</td>
<td>20</td>
</tr>
<tr>
<td>Office ZIP</td>
<td>Variable</td>
<td>15</td>
</tr>
<tr>
<td>Office Country</td>
<td>Fixed Text</td>
<td>4</td>
</tr>
<tr>
<td>Office Phone</td>
<td>Variable</td>
<td>15</td>
</tr>
<tr>
<td>Office Extension</td>
<td>Variable</td>
<td>4</td>
</tr>
<tr>
<td>Office Fax</td>
<td>Variable</td>
<td>15</td>
</tr>
<tr>
<td>Office Email</td>
<td>Variable</td>
<td>60</td>
</tr>
<tr>
<td>Office Room Number</td>
<td>Variable</td>
<td>6</td>
</tr>
<tr>
<td>SSN Designator</td>
<td>Variable</td>
<td>6</td>
</tr>
<tr>
<td>Non-Government Agency</td>
<td>Fixed Text</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protected Personal Information</th>
<th>Type</th>
<th>Max. Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social Security Number</td>
<td>Fixed Text</td>
<td>9</td>
</tr>
<tr>
<td>Date of Birth</td>
<td>Date</td>
<td>8</td>
</tr>
<tr>
<td>Gender</td>
<td>Fixed Text</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Control</th>
<th>Type</th>
<th>Max. Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEIWG Data</td>
<td>Fixed Text</td>
<td>40</td>
</tr>
<tr>
<td>PIN</td>
<td>Fixed Numeric</td>
<td>10</td>
</tr>
<tr>
<td>Domain ( Facility / System ID )</td>
<td>Variable</td>
<td>8</td>
</tr>
<tr>
<td><strong>Login Information</strong></td>
<td><strong>Type</strong></td>
<td><strong>Max. Length (Bytes)</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>User ID</td>
<td>Variable</td>
<td>60</td>
</tr>
<tr>
<td>Domain</td>
<td>Variable</td>
<td>60</td>
</tr>
<tr>
<td>Password</td>
<td>Variable</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Card Information</strong></th>
<th><strong>Type</strong></th>
<th><strong>Max. Length (Bytes)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuer ID</td>
<td>Variable</td>
<td>32</td>
</tr>
<tr>
<td>Issuance Counter</td>
<td>Variable</td>
<td>4</td>
</tr>
<tr>
<td>Issue Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
<tr>
<td>Expiration Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
<tr>
<td>Card Type</td>
<td>Variable</td>
<td>32</td>
</tr>
<tr>
<td>Demographic Data Load Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
<tr>
<td>Demographic Data Expiration Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
<tr>
<td>Card Security Code</td>
<td>Fixed Text</td>
<td>32</td>
</tr>
<tr>
<td>Card ID/AID</td>
<td>Variable</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Biometrics – X.509 Certificate</strong></th>
<th><strong>Type</strong></th>
<th><strong>Max. Length (Bytes)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>Variable</td>
<td>512</td>
</tr>
<tr>
<td>Certificate</td>
<td>Variable</td>
<td>1500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PKI – Digital Signature Certificates</strong></th>
<th><strong>Type</strong></th>
<th><strong>Max. Length (Bytes)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>Variable</td>
<td>3000</td>
</tr>
<tr>
<td>Issue Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
<tr>
<td>Expiration Date</td>
<td>Date (YYYYMMDD)</td>
<td>8</td>
</tr>
</tbody>
</table>
# GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>AID</td>
<td>Application Identifier</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>APDU</td>
<td>Application protocol data unit</td>
</tr>
<tr>
<td>API</td>
<td>Applications Programming Interface</td>
</tr>
<tr>
<td>ATR</td>
<td>Answer-to-Reset</td>
</tr>
<tr>
<td>BSI</td>
<td>Basic Services Interface</td>
</tr>
<tr>
<td>C-APDU</td>
<td>Command APDU</td>
</tr>
<tr>
<td>CAPI</td>
<td>Cryptographic Applications Programming Interface</td>
</tr>
<tr>
<td>CHV</td>
<td>Card Holder Verification</td>
</tr>
<tr>
<td>CLA</td>
<td>Class Byte of the Command Message</td>
</tr>
<tr>
<td>CLK</td>
<td>Clock</td>
</tr>
<tr>
<td>Cold reset</td>
<td>The reset of an ICC that occurs when the supply voltage (VCC) and other</td>
</tr>
<tr>
<td></td>
<td>signals to the ICC are raised from the inactive state and the reset (RST)</td>
</tr>
<tr>
<td></td>
<td>signal is applied.</td>
</tr>
<tr>
<td>Command</td>
<td>A message sent by the terminal to the ICC that initiates an action and</td>
</tr>
<tr>
<td></td>
<td>solicits a response from the ICC.</td>
</tr>
<tr>
<td>Contact</td>
<td>A conducting element ensuring galvanic continuity between integrated</td>
</tr>
<tr>
<td></td>
<td>circuit(s) and the external interfacing equipment.</td>
</tr>
<tr>
<td>CSP</td>
<td>Cryptographic Service Provider</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CWI</td>
<td>Character Waiting Time Integer</td>
</tr>
<tr>
<td>CWT</td>
<td>Character Waiting Time</td>
</tr>
<tr>
<td>DAD</td>
<td>Destination Node Address</td>
</tr>
<tr>
<td>Data unit</td>
<td>The smallest set of bits that can be unambiguously referenced.</td>
</tr>
<tr>
<td>EDC</td>
<td>Error Detection Code</td>
</tr>
<tr>
<td>ETU</td>
<td>Elementary Time Unit</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>GSC</td>
<td>Government Smart Card, as defined in the Smart Access Common</td>
</tr>
<tr>
<td></td>
<td>Identification Card Solicitation.</td>
</tr>
<tr>
<td>Guardtime</td>
<td>The minimum time between the trailing edge of the parity bit of a character</td>
</tr>
<tr>
<td></td>
<td>and the leading edge of the start bit of the following character sent in</td>
</tr>
<tr>
<td></td>
<td>the same direction.</td>
</tr>
<tr>
<td>Half-duplex transmission:</td>
<td>Two-way electronic communication that takes place in only one direction</td>
</tr>
<tr>
<td></td>
<td>at a time.</td>
</tr>
<tr>
<td>HLSI</td>
<td>High Level Service Interface</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>I-block</td>
<td>Information block associated with the T=1 protocol.</td>
</tr>
<tr>
<td>ICC</td>
<td>Integrated Circuit Card</td>
</tr>
</tbody>
</table>
IEC International Electrotechnical Commission.
IFD Interface Device
IFS Information Field Size associated with the T=1 protocol.
IFSC Information Field Size for the ICC associated with the T=1 protocol.
IFSD Information Field Size for the terminal associated with the T=1 protocol.
IFSI Information Field Size Integer associated with the T=1 protocol.
Inactive The supply voltage (VCC) and other signals to the ICC are in the inactive state when they are at a potential of 0.4 V or less with respect to ground (GND).
INF Information field associated with the T=1 protocol.
INS Instruction Byte of Command Message associated with the T=0 and T=1 protocol.
ISO International Organization for Standardization
LEN Length
LRC Longitudinal Redundancy Check associated with the T=1 protocol.
NAD Node address associated with the T=1 protocol.
NAK Negative ACK
OCF Open Card Framework
P1(2) Parameters used in the T=0 and T=1 protocol.
PCB Protocol Control Byte
PC/SC Personal Computer/Smart Card
PIN Personal Identification Number
PTS Protocol Type Selection
PKCS Public Key Cryptography System
R-APDU Response APDU
R-block Receive Ready Block
Response A message returned by the ICC to the terminal after the processing of a command message received by the ICC.RFU Reserved for Future Use.
RST Reset
R-TPDU Response TPDU
SAD Source Node address associated with the T=1 protocol.
SEIWG Security Enterprise Integration Working Group
SPI Service Provider Interface
SPM (Also GSC-SPM):
   Service Provider Module
SPS Service Provider Software
S-block Supervisory Block
State A Space (as defined in ISO 1177)
State H High state logic level
State L Low state logic level
State Z Mark (as defined in ISO 1177)
SW1 (2) Status Byte 1 (2)
T=0 Character-oriented asynchronous half duplex transmission protocol
T=1 Block-oriented asynchronous half duplex transmission protocol
TAL Terminal Application Layer
TCK Check Character
TLV  Tag-Length-Value
TPDU  Transport Protocol Data Unit
TTL  Terminal Transport Layer
VCC  Supply Voltage
VPP  Programming Voltage
Warm reset  The reset of an ICC that occurs when the supply voltage (VCC) and the clock (CLK) lines are maintained in their active state and the reset (RST) signal is applied.
WI  Waiting Time Integer
WTX  Waiting Time Extension
XSI  Extended Service Interface(s)