# ActivID® ActivClient® SDK
# BSI API Reference Guide

DOCUMENT REFERENCE: AC_SDK_BSI_7.1_RG_07.2016

PRODUCT VERSION: 7.1

JULY 2016



# Copyright

© 2008-2016 HID Global Corporation/ASSA ABLOY AB. All rights reserved. This document may not be reproduced, disseminated or republished in any form without the prior written permission of HID Global Corporation.

# Trademarks

HID, HID Global, the HID Blue Brick logo, the Chain Design and 4TRESS, ActivIdentity and ActivID are trademarks or registered trademarks of HID Global or its licensor(s)/supplier(s) in the US and other countries and may not be used without permission. The absence of a mark, product, service name or logo from this list does not constitute a waiver of the HID Global trademark or other intellectual property rights concerning that name or logo. All other trademarks, service marks, and product or service names are trademarks or registered trademarks of their respective owners.

# Revision History

| Date | Description | Document Version |
|---|---|---|
| July 2016 | Initial release of rebranded document and major technical updates. | 1.0 |

# Contacts


| Americas | | Asia Pacific |
|---|---|---|
| United States: +1 510.574.0100 | U.S. Federal: +1 571.522.1000 | +61 (0) 3.9809.2892 |
| **Europe** | | **Corporate** |
| +33 (0) 1.42.04.84.00 | | +1 800.237.7769 |

**Technical Support**

If you purchased your product from a third party, then please contact that third party for Technical Support. For products purchased directly from HID Global, please go to http://www.hidglobal.com/support

# Typographic and Document Conventions

| Typography | Description |
|---|---|
| blue | Cross-references within the document. |
| blue, underline | References to external web addresses. |
| **bold** | Action steps (paths, buttons, options); field and drop-down list labels; emphasis. |
| *italic* | File names, document titles, and file extensions. |
| `Code snippets` | Highlights `code snippets` within regular content. |
| `Code samples` | `Highlights code samples` |
|  | **WARNING**: This symbol indicates a critical warning. It applies to actions that if taken or not taken will break the system. Read the warning carefully and follow it. |
|  | **Important**: This symbol indicates something very important to the reader. Ignore this symbol at your own risk. |
|  | **Note**: This symbol indicates a note that should be of interest to the reader. It is not critical. Nevertheless, the reader should pay attention. |

# Table of Contents

**1.0 Introduction ..... 7**

1.1 Product Overview ..... 7
1.2 Document Scope and Audience ..... 7
1.3 About the BSI API ..... 8

**2.0 Overview ..... 10**

2.1 Include Files Required ..... 10
2.2 How to Deploy ..... 10
2.3 Access Control Rules ..... 10
2.3.1 Establishing the Security Context ..... 10
2.3.2 External Authentication Access Control Rule Modes ..... 10
2.3.2.1 Access Method Types ..... 11
2.3.2.2 Access Control Rule Types ..... 11
2.3.2.3 Access Control Rule Values ..... 11
2.3.2.4 Secure Channel Access Control Rule ..... 12
2.4 Cryptographic Algorithms ..... 12
2.5 Data Encoding ..... 13
2.6 Determining the Size of Buffer to Allocate ..... 13
2.7 Support for Multi-threading ..... 13
2.8 Support for Concurrency ..... 13

**3.0 Packing List ..... 14**

3.1 Header Files ..... 14
3.2 Static Libraries ..... 14
3.3 Samples ..... 14
3.3.1 BSI C Sample ..... 14
3.3.2 BSI Java Sample ..... 14

**4.0 Constants ..... 15**

4.1 General Constants ..... 15
4.1.1 BSI_AUTHENTICATOR_MAX_LEN ..... 15
4.1.2 BSI_ERROR_TEXT_LEN ..... 15
4.1.3 BSI_KEY_LENGTH ..... 15
4.1.4 BSI_RSA_NO_PAD ..... 15
4.2 Access Method Types ..... 15
4.2.1 BSI_AM_PIN ..... 15
4.2.2 BSI_AM_SECURE_CHANNEL_GP ..... 15
4.2.3 BSI_AM_SECURE_CHANNEL_ISO ..... 16
4.2.4 BSI_AM_XAUTH ..... 16
4.3 Access Control Rule Types (ACRType) ..... 16
4.3.1 BSI_ACR_ALWAYS ..... 16
4.3.2 BSI_ACR_NEVER ..... 16
4.3.3 BSI_ACR_XAUTH ..... 16

4.3.4 BSI_ACR_XAUTH_OR_PIN ..... 17
4.3.5 BSI_SECURE_CHANNEL_GP ..... 17
4.3.6 BSI_ACR_PIN_ALWAYS ..... 17
4.3.7 BSI_ACR_PIN ..... 17
4.3.8 BSI_ACR_XAUTH_THEN_PIN ..... 17
4.3.9 BSI_ACR_UPDATE_ONCE ..... 17
4.3.10 BSI_ACR_PIN_THEN_XAUTH ..... 18
4.3.11 BSI_SECURE_CHANNEL_OP ..... 18
4.3.12 BSI_SECURE_CHANNEL_GP ..... 18
4.3.13 BSI_SECURE_CHANNEL_ISO ..... 18
4.3.14 BSI_ACR_XAUTH_AND_PIN ..... 18

**5.0 Data Types and Structures ..... 19**

5.1 Basic Types ..... 19
5.2 BSI Return Type ..... 19
5.3 Card Connection Types ..... 19
5.4 Generic Container Types ..... 20
5.4.1 GCContainerSize ..... 20
5.4.2 Generic Container Data Identification Types ..... 20
5.5 Access Control Rule Structures ..... 21
5.5.1 BSIAuthenticator ..... 21
5.5.2 BSIAcr ..... 22
5.5.3 GCAcr ..... 23
5.5.4 CRYPTOAcr ..... 24

**6.0 Utility Provider ..... 25**

6.1 Access Control Rules ..... 25
6.2 Return Codes ..... 25
6.3 Function Summary ..... 25
6.4 Function Details ..... 26
6.4.1 gscBsiUtilAcquireContext() ..... 26
6.4.2 gscBsiUtilBeginTransaction() ..... 27
6.4.3 gscBsiUtilConnect() ..... 28
6.4.4 gscBsiUtilDisconnect() ..... 29
6.4.5 gscBsiUtilEndTransaction() ..... 29
6.4.6 gscBsiUtilGetCardProperties() ..... 30
6.4.7 gscBsiUtilGetCardStatus() ..... 31
6.4.8 gscBsiUtilGetExtendedErrorText() ..... 31
6.4.9 gscBsiUtilGetReaderList() ..... 32
6.4.10 gscBsiUtilGetVersion() ..... 33
6.4.11 gscBsiUtilPassthru() ..... 33
6.4.12 gscBsiUtilReleaseContext() ..... 34
6.5 Extended Functionality ..... 35
6.5.1 gscXsiUtilGetCardATR() ..... 35
6.5.2 gscXsiUtilGetCuid() ..... 36
6.5.3 gscXsiUtilGetIdentifier() ..... 37
6.5.4 gscXsiUtilChangePIN() ..... 38
6.5.5 gscXsiUtilIsPINVerified() ..... 39

6.5.6 gscXsiNotifySynchronizationStart() ..... 40
6.5.7 gscXsiNotifySynchronizationEnd() ..... 40
6.5.8 gscXsiUtilGetCardManagerState() ..... 41
6.5.9 gscXsiUtilGetForceChangePIN() ..... 41

**7.0 Generic Container Provider ..... 43**

7.1 Discovery Functions ..... 43
7.2 Differentiating Between Data Sets ..... 43
7.3 Access Control Rules ..... 43
7.3.1 Compatibility with the Card Edge Interface ..... 44
7.3.2 SIMPLE-TLV Data Object ..... 44
7.3.3 Byte Order ..... 44
7.4 Generic Container Data Types and Structures ..... 45
7.5 Function Summary ..... 45
7.5.1 Access Control ..... 45
7.5.2 Data Management Functions ..... 45
7.6 Function Details ..... 46
7.6.1 gscBsiGcDataCreate() ..... 46
7.6.2 gscBsiGcDataDelete() ..... 47
7.6.3 gscBsiGcGetContainerProperties() ..... 48
7.6.4 gscBsiGcReadTagList() ..... 49
7.6.5 gscBsiGcReadValue() ..... 50
7.6.6 gscBsiGcUpdateValue() ..... 51

**8.0 Cryptographic Provider ..... 53**

8.1 About the Cryptographic Provider ..... 53
8.2 Access Control Rules ..... 53
8.2.1 Function Summary ..... 53
8.2.1.1 Access Control ..... 53
8.2.1.2 Challenge ..... 53
8.2.1.3 SKI ..... 53
8.2.1.4 PKI ..... 54
8.3 Function Details ..... 54
8.3.1 gscBsiGetChallenge() ..... 54
8.3.2 gscBsiGetCryptoProperties() ..... 55
8.3.3 gscBsiPkiCompute() ..... 56
8.3.4 gscBsiPkiGetCertificate() ..... 58
8.3.5 gscBsiSkiInternalAuthenticate() ..... 59

**9.0 Return Codes ..... 61**

**Appendix A: Terms and Acronyms ..... 63**

A.1. Terms ..... 63
A.2. Acronyms ..... 65

# 1.0 Introduction




**ActivID® ActivClient® guards against an ever-changing threat landscape by providing organizations with risk-appropriate and secure access to corporate IT assets.**

## 1.1 Product Overview

HID Global's ActivID ActivClient is the latest smart card and USB token middleware from that allows enterprise and government customers to easily use smart cards and USB tokens to protect workstations and networks.

ActivClient enables the use of PKI certificates and keys and one-time passwords and static password credentials on a smart card or USB token to secure:

- Desktop applications
- Network logon
- Remote access
- Web logon
- E-mail
- Electronic transactions

ActivClient provides the following range of services:

- PKI services
- Remote access and One-Time Password (OTP) services
- Remote session services
- Management services (for end users and administrators)
- Development services with a Software Development Kit (SDK)

## 1.2 Document Scope and Audience

This document describes the Basic Services Interface (BSI) API, a generic API that provides access to cryptographic and non-cryptographic services on smart cards. The BSI API v2.1 complies with the Government Smart Card Interoperability Specification GSC-IS v2.1 (http://csrc.nist.gov/publications/nistir/nistir-6887.pdf), which was developed by NIST.

Readers of this document are assumed to be experienced programmers who are accustomed to reading and writing in the C programming language.

## 1.3 About the BSI API

The BSI API is used to access the services provided by applets stored on the card. Although released as a single DLL, BSI API functionality is divided into three providers, each of which provides access to a set of services implemented by the applets stored on the card.

- Utility Provider – Provides utility services for obtaining a list of available card readers and establishing and terminating logical connections with a smart card. For further information, see section 6.0 Utility Provider on page 25.
- Generic Container Provider – Abstracts the storage semantics of the smart card applications with a simple interface for managing Generic Containers of data elements in tag/length/value format. For further information, see section 7.0 Generic Container Provider on page 43.
- Cryptographic Provider – Provides fundamental cryptographic services such as random number generation, authentication, and digital signature generation. For further information, see section 8.0 Cryptographic Provider on page 53.



**Note**: BSI is recommended only for very basic needs such as authentication to a card or access and usage of data stored in a dedicated applet. BSI may also be used to provide PKI services, but in order to use BSI in this way, the developer must have knowledge of the card data model.

For developing PKI-enabled applications, high-level cryptographic providers such as PKCS#11 or Microsoft Cryptography API (provided in the ActivClient Mini Driver) are recommended. Unlike the PKCS#11 API and ActivClient Mini Driver, which provide a high level of abstraction, the BSI API requires the application developer to know the smart card logical data model. Specifically, in order to make a call using the BSI API, the caller must know the AID of the applet instances.

**Prerequisite**: The BSI API is supported on platforms where ActivClient has been installed. Refer to the *ActivID ActivClient for Windows Installation Guide* for pre-requisites for ActivClient installation.

Use the BSI API if you are developing a multi-credential application.

**Supports**: PKI and static data such as personal information data. The BSI API and the PKCS#11 API are similar in functionality and scope. They differ in that PKCS#11 is a standard developed by RSA whereas BSI is a standard developed by the US government/NIST. PKCS#11 implements a higher level of abstraction of card objects and services than BSI, which is much lower level than PKCS#11. In addition, BSI supports SKI.

**Languages**: C, Java; HID Global provides samples in C and Java.

**Description**: ActivClient SDK's BSI API component is an implementation of the Basic Services Interface (BSI) included in the U.S. Government Smart Card - Interoperability Specifications (GSC-IS). The library included in ActivClient SDK implements a subset of BSI API v2.1. This API provides cryptographic, data storage, and utility services.

The BSI API:

- Provides support for:
  - Smart card cryptographic and data storage operations.
  - Smart card state and reader state management.

  - Data storage.
  - PIN management.

- Is recommended for developers who want to perform smart card cryptographic operations and/or some data storage. While the BSI API hides most of the complexity of working with smart cards, it requires more knowledge of smart cards than other ActivClient APIs require.

# 2.0 Overview

## 2.1 Include Files Required

The necessary include files are located in the **BSI/C/Headers** folder of the distribution.

## 2.2 How to Deploy

BSI API applications can be linked statically with the BSI API import library. The static libraries are located in the **BSI/C/Libraries/x86** and **BSI/C/Libraries/x64** folders of the distribution.

## 2.3 Access Control Rules

An access method is an authentication method that must be verified by the smart card before access to card services is allowed. An access control rule is a set of access methods that are logically ORed and/or ANDed together.

Each smart card service is protected by specific access control rules. The set of access control rules is defined for each service during card issuance when the smart card is initialized. Each Provider Module offers a discovery function call that allows an application to discover which access control rules govern access to the services available in the module. The smart card will not provide services if the access control rules are not validated.

### 2.3.1 Establishing the Security Context

The Utility Provider Module of the BSI API provides the access control functions. The calling application must establish the security context by invoking the `gscBsiUtilAcquireContext()` function. The calling application is then responsible for releasing the security context using the `gscBsiUtilReleaseContext()` function.

### 2.3.2 External Authentication Access Control Rule Modes

- Strong
- Weak

| Mode | Description |
|---|---|
| Strong | From a security perspective, the Strong mode is preferred. Use this 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)). Before calling the authentication function, the calling application should call the `gscBsiGetChallenge()` function provided by the Cryptographic Provider Module in order to get a challenge from the card.<br>To ensure that the authentication key never leaves the HSM, the calling application should |

| Mode | Description |
|---|---|
| | perform the challenge encryption required by its HSM and provide the resulting cryptogram in the `AuthValue` member of the `BSIAuthenticator` structure. |
| **Weak** | Do not use this mode, it is documented only for backward compatibility.<br>In this mode, the external authentication key is passed to the BSI Provider in clear text by the calling application, then the BSI Provider encapsulates the external authentication transaction with the card or the applet.<br>The clear text key is provided in the Key member of the Authenticator structure (for further information, see section 5.5.1 BSIAuthenticator on page 21). |

#### 2.3.2.1 Access Method Types

See section 4.2 Access Method Types on page 15 for a list of access method types.

#### 2.3.2.2 Access Control Rule Types

See section 4.3 Access Control Rule Types (ACRType) on page 16 for a list of access control rule types. The GCAcr structure specifies the access control rules supported by the Generic Container Provider. The CRYPTOAcr structure specifies the ACR supported by the Cryptographic Provider.

#### 2.3.2.3 Access Control Rule Values

The following table lists access control rule values and explains how those values affect the corresponding service.

| Access Control Rule | Service Behavior |
|---|---|
| BSI_ACR_ALWAYS | Always. The service is provided without restrictions. No access control rule is required. |
| BSI_ACR_NEVER | Never. The service can never be provided. |
| BSI_ACR_PIN | PIN Protected. The service is provided only if its associated PIN code has already been verified in the current card session. |
| BSI_ACR_PIN_ALWAYS | Always PIN Protected. The service is provided only if its associated PIN code is verified in the current card session immediately prior to service request. |
| BSI_ACR_PIN_THEN_XAUTH | PIN presentation followed by External Authentication. |
| BSI_ACR_UPDATE_ONCE | Not implemented. |

| Access Control Rule | Service Behavior |
|---|---|
| BSI_ACR_XAUTH | External Authenticate. The service is provided only after a challenge/response exchange has been performed between the client application and the card. For possible operation modes, see section 2.3 Access Control Rules on page 10. |
| BSI_ACR_XAUTH_AND_PIN | External Authenticate and PIN. Either one of the two controls gives access to the corresponding service. This allows a validation of the card holder when a PIN PAD is available and for an external authentication when no PIN PAD is available. |
| BSI_ACR_XAUTH_OR_PIN | External Authenticate or PIN. Either one of the two controls gives access to the corresponding service. This allows a validation of the card holder when a PIN PAD is available and for an external authentication when no PIN PAD is available. |
| BSI_ACR_XAUTH_THEN_PIN | External Authenticate then PIN. You must chain External Authenticate method first, then the PIN protected method, both successfully, in order that access to the service be granted. This makes it possible to authenticate the application and the user of the application. |
| BSI_SECURE_CHANNEL_GP | Secure Channel Global Platform. The service is provided through a Secure Channel managed by the card's Secure Messaging layer. The calling application must operate inside the `gscBsiUtilPassthru()` function. |
| BSI_SECURE_CHANNEL_ISO | Not implemented. |
| BSI_SECURE_CHANNEL_OP | Legacy ACR: use BSI_SECURE_CHANNEL_GP instead. |

#### 2.3.2.4 Secure Channel Access Control Rule

The Secure Channel Access Control Rule requires that cryptographic operations be performed at the Application Protocol Data Unit (APDU) level. The BSI API offers a pass-through service that allows the calling application to send the correct APDU to the card to open the Secure Channel. As with other access controls, this rule is defined during card issuance.

## 2.4 Cryptographic Algorithms

The BSI API supports the following cryptographic algorithms:

- DES ECB with a single length key size, 8 bytes.
- Triple DES ECB with a double length key size, 16 bytes.
- RSA_NO_PAD computation on the private key, Chinese Remainder Theorem.

## 2.5 Data Encoding

All NULL-terminated strings (parameters with prefix usz) contain only printable ASCII characters. You must encode all binary data in ASCII hexadecimal format. The length parameter for data should include the length of the encoded string plus one byte for the NULL terminator.

- Example 1

  For example, the raw hexadecimal data 0x01, 0xFE, and 0x32 would be encoded as the string 01FE32. The length parameter associated with this buffer equals 7.

  **Note**: The buffer string must be NULL terminated.

- Example 2

  The following call returns BSI_BAD_PARAM because value and value length are inconsistent (value length should be 7):

  ```
  gscBsiGcUpdateValue(hCard, uszAID, unAIDLen, ucTag, “01FE32”, 6)
  ```

## 2.6 Determining the Size of Buffer to Allocate

Some provider functions require the calling application to provide pre-allocated tables or buffers for the function output. For ASCII hex-encoded data, the length must include the NULL terminator. If the buffer passed is not large enough, the function returns the BSI_INSUFFICIENT_BUFFER error code.

In order to determine how big a buffer is needed, the calling application can call the function twice:

1. If, in the first call, the calling application sets the buffer address parameter to NULL, the function places the number of bytes needed into the buffer size parameter.
2. Then, before the second call, the calling application allocates a buffer of the appropriate size and calls the function with the address of this buffer.

## 2.7 Support for Multi-threading

This API is thread-safe. However, it provides no built-in multithreading methods. To perform operations simultaneously, the calling application must implement multiple threads.

## 2.8 Support for Concurrency

The PC/SC layer as leveraged by the ActivClient APIs supports transactions that serialize all operations from concurrent applications safely. ActivClient uses the PC/SC Transactions API to ensure card security and consistency.

For additional information, see http://msdn2.microsoft.com/En-US/library/aa379469.aspx.

# 3.0 Packing List

All wrappers, samples, and documentation are for BSI 2.1.

## 3.1 Header Files

- bsi.h
- bsid.h
- bsif.h
- bsit.h

## 3.2 Static Libraries

- acbsi21.lib, x86 version
- acbsi21.lib, x64 version

## 3.3 Samples

The samples demonstrate the following scenarios:

- How to use a private key to sign data.
- How to read data items from the card.
- How to send APDUs to an applet instance.

### 3.3.1 BSI C Sample

Both x86 and x64 versions are provided.

The BSI C x86 sample works with both the ActivClient 32 and 64-bit editions.

### 3.3.2 BSI Java Sample

The Java sample uses the BSI Java wrapper to demonstrate the same features as the C/C++ samples.

This sample can be run on both 32 and 64-bit platforms. Javadoc is provided as well.

# 4.0 Constants

## 4.1 General Constants

### 4.1.1 BSI_AUTHENTICATOR_MAX_LEN

The maximum size of the `uszAuthValue` attribute of the `BSIAuthenticator` structure.

```
#define BSI_AUTHENTICATOR_MAX_LEN 255
```

### 4.1.2 BSI_ERROR_TEXT_LEN

Maximum size of the error text buffer returned by the `gscBsiUtilGetExtendedErrorText()` function.

```
#define BSI_ERROR_TEXT_LEN 255
```

### 4.1.3 BSI_KEY_LENGTH

The size of the key passed in the `uszAuthValue` attribute of the `BSIAuthenticator` structure when using external authentication in weak mode.

```
#define BSI_KEY_LENGTH 64
```

### 4.1.4 BSI_RSA_NO_PAD

Identifies the cryptographic algorithm to be used to perform a private key computation on a message digest. It is passed as the `ucAlgoID` attribute of the `gscBsiPkiCompute()` function.

```
#define BSI_RSA_NO_PAD 0xA3
```

## 4.2 Access Method Types

### 4.2.1 BSI_AM_PIN

PIN code is required

**Value**: 0x06

### 4.2.2 BSI_AM_SECURE_CHANNEL_GP

Secure Channel (Global Platform)

**Value**: 0x04

### 4.2.3 BSI_AM_SECURE_CHANNEL_ISO

Secure Channel (ISO 7816-4)

**Value**: 0x0B

### 4.2.4 BSI_AM_XAUTH

External Authentication

**Value**: 0x02

## 4.3 Access Control Rule Types (ACRType)

Identifies the access control rule type.

These values are returned by the `gscBsiGcGetContainerProperties()` and `gscBsiGetCryptoProperties()` to identify the required access controls for the respective Generic Container and PKI card services.

### 4.3.1 BSI_ACR_ALWAYS

The service is always accessible; there is no need to first authenticate to the card to get access to the service.

**Access methods in this access control rule**: None

**Value**: 0x00

### 4.3.2 BSI_ACR_NEVER

Operation is never possible.

**Access methods in this access control rule**: None

**Value**: 0x01

### 4.3.3 BSI_ACR_XAUTH

External Authentication.

**Access methods in this access control rule**: BSI_AM_XAUTH

**Value**: 0x02

### 4.3.4 BSI_ACR_XAUTH_OR_PIN

Service can be requested after an External Authentication or after a successful PIN presentation. Not implemented.

**Access methods in this access control rule**: BSI_AM_XAUTH OR BSI_AM_PIN

**Value**: 0x03

### 4.3.5 BSI_SECURE_CHANNEL_GP

Secure Channel (Global Platform). Not implemented.

**Access methods in this access control rule**: BSI_AM_SECURE_CHANNEL_GP

**Value**: 0x04

### 4.3.6 BSI_ACR_PIN_ALWAYS

Verify PIN immediately prior to service request.

**Access methods in this access control rule**: BSI_AM_PIN

**Value**: 0x05

### 4.3.7 BSI_ACR_PIN

PIN code is required.

**Access methods in this access control rule**: BSI_AM_PIN

**Value**: 0x06

### 4.3.8 BSI_ACR_XAUTH_THEN_PIN

External Authentication followed by a PIN presentation.

**Access methods in this access control rule**: BSI_AM_XAUTH AND BSI_AM_PIN

**Value**: 0x07

### 4.3.9 BSI_ACR_UPDATE_ONCE

You can only update the Target object once during its lifetime. Not implemented.

**Access methods in this access control rule**: None

**Value**: 0x08

### 4.3.10 BSI_ACR_PIN_THEN_XAUTH

PIN presentation followed by External Authentication.

**Access methods in this access control rule**: BSI_AM_PIN AND BSI_AM_XAUTH

**Value**: 0x09

### 4.3.11 BSI_SECURE_CHANNEL_OP

Secure Channel (Global Platform) (GSC 1.8).

**Value**: 0x04

### 4.3.12 BSI_SECURE_CHANNEL_GP

Secure Channel (Global Platform).

**Value**: 0x04

### 4.3.13 BSI_SECURE_CHANNEL_ISO

Secure Channel. (ISO 7816-4). Not implemented.

**Access methods in this access control rule**: BSI_AM_SECURE_CHANNEL_ISO

**Value**: 0x0B

### 4.3.14 BSI_ACR_XAUTH_AND_PIN

PIN presentation and External Authentication in any order are required.

**Access methods in this access control rule**: BSI_AM_XAUTH AND BSI_AM_PIN

**Value**: 0x0C

# 5.0 Data Types and Structures

Data types and other structures used by the BSI API are described in this chapter.

## 5.1 Basic Types

These types must be used for parameters passed to the BSI functions.

BSI_ULONG

```
typedef unsigned long BSI_ULONG
```

BSI_BYTE

```
typedef unsigned char BSI_BYTE
```

BSI_BYTE_PTR

```
typedef BSI_BYTE BSI_BYTE_PTR
```

BSI_INT

```
typedef BSI_ULONG BSI_INT
```

BSI_UINT

```
typedef BSI_ULONG BSI_UINT
```

BSI_CHAR

```
typedef char BSI_CHAR
```

## 5.2 BSI Return Type

All BSI functions return error codes of this type.

BSI_RETURN

```
typedef BSI_ULONG BSI_RETURN
```

## 5.3 Card Connection Types

Use this type for card connection handle parameters required by BSI functions.

BSI_CARD_HANDLE

```
typedef BSI_ULONG BSI_CARD_HANDLE
```

BSI_CARD_HANDLE_PTR

```
typedef BSI_CARD_HANDLE * BSI_CARD_HANDLE_PTR
```

## 5.4 Generic Container Types

### 5.4.1 GCContainerSize

```
typedef struct strctContainerSizes {
BSI_ULONG unMaxNbDataItems;
BSI_ULONG unMaxValueStorageSize;
} GCContainerSize;
typedef GCContainerSize BSI_GC_SIZE;
typedef BSI_GC_SIZE BSI_PTR BSI_GC_SIZE_PTR;
```

**Fields**

unMaxNbDataItems

unsigned long. Maximum number of {tag, length, value} data items a given Generic Container instance can hold.

unMaxValueStorageSize

unsigned long. The maximum size of the value buffer of the container.

### 5.4.2 Generic Container Data Identification Types

The Generic Container data identification types are:

- GCTag

  The type for Generic Container tags. Provided for backward compatibility. Use BSI_GCT instead.

  typedef BSI_BYTE GCTag

- BSI_GCT

  GSC-IS 2.1 Generic Container tag type definition.

  typedef GCtag BSI_GCT

## 5.5 Access Control Rule Structures

### 5.5.1 BSIAuthenticator

The BSIAuthenticator structure defines an access method. Client applications use this structure to establish a security context with the card. This structure contains the access method type and its associated authenticator. An array of BSIAuthenticator structures defines an access control rule and is provided as a parameter to the gscBsiUtilAcquireContext() function. The gscBsiUtilAcquireContext() function extracts the authenticators from the structure and passes them to the card applet for verification.

To release the access rights acquired, use the gscBsiUtilReleaseContext() function.

For further information see Data Encoding on page 13, External Authentication Access Control Rule Modes on page 10, and the gscBsiGetChallenge() method.

```
typedef struct {
BSI_ULONG unAccessMethodType; BSI_ULONG unKeyIDOrReference;
BSI_BYTE uszAuthValue [BSI_AUTHENTICATOR_MAX_LEN]; BSI_ULONG unAuthValueLen;
} BSIAuthenticator;
typedef BSIAuthenticator* BSI_AUTH_PTR
```

**Fields**

unAccessMethodType

BSI_ULONG. Access method type. This parameter must correspond to the one returned by the ACR discovery function provided by the Provider Module:

- gscBsiGcGetContainerProperties() for the Generic Container Provider Module.
- gscBsiGetCryptoProperties() for the Cryptographic Provider Module.

**Values**:

Either BSI_AM_PIN or BSI_AM_XAUTH.

If the usAuthValue parameter is set to the external authentication key to be used in the Provider's weak mode or to the external authentication cryptogram used in provider’s strong mode, the unAccessMethodType field should be set to BSI_AM_XAUTH.

unKeyIDOrReference

BSI_ULONG. Key identifier or reference of the authenticator. Use this field to distinguish between multiple authenticators with the same access method type.

**Values**:

If the usAuthValue parameter is set to the External Authentication key to be used in the Provider's weak mode or to the External Authentication cryptogram used in provider’s strong mode, the unKeyIDOrReference field should be set to the value returned by the ACR discovery function.

usAuthValue

char. ASCII hex-encoded. This parameter holds the value of the authenticator.

**Values**:

The values this field can take are:

- PIN code. In this case, the unAccessMethodType field of this structure should be set to BSI_AM_PIN.
- The External Authentication cryptogram used in provider's strong mode. In this case, you must call the gscBsiGetChallenge() function prior to obtaining the context. or
- The External Authentication key to be used in the Provider's weak mode. In this case, the unAuthValueLen field of this structure should be set to 0.

**Note**: If the unAuthValueLen field is set to 0, then this field is ignored.

unAuthValueLen

BSI_ULONG. Authenticator value length.

**Values**:

The values this field can take are:

- If unAccessMethodType is BSI_AM_PIN, if unAuthValueLen is set to 0, the usAuthValue is ignored and the BSI API implementation displays a user interface that allows user to enter the PIN code. Otherwise, set unAuthValueLen to the ASCII encoded PIN code length (including the NULL terminator).
- If unAccessMethodType is BSI_AM_XAUTH, if unAuthValueLen is set to 0, the authentication is assumed to be performed using weak mode. If unAuthValueLen is non-zero, the authentication is assumed to be performed using strong mode. Set this value to the length of the ASCII-encoded cryptogram (including NULL terminator).

### 5.5.2 BSIAcr

The BSIAcr structure is the ACR type returned by the gscBsiGcGetContainerProperties() and gscBsiGetCryptoProperties() ACR discovery methods. It is used by the application to discover which access controls protect various card services.

This structure is part of the BSI_GC_ACR and BSI_CRYPTO_ACR structures.

```
#define MaxNbAM 4
typedef struct strctBSIAcr {
BSI_ULONG unACRType;
BSI_ULONG unKeyIDOrReference[MaxNbAM]; BSI_ULONG unAuthNb;
BSI_ULONG unACRID;
} BSIAcr;
```

**Fields**

unACRType

BSI_ULONG. Access control type.

**Values**:

See possible values in Access Control Rule Types (ACRType) on page 16.

unKeyIDOrReference

BSI_ULONG. Array of keyIDOrReferences that contains the key identifier or reference for each access method contained in the ACR in order of appearance. These key references are used when calling the gscBsiUtilAcquireContext() function to establish the Security Context.

unAuthNb

BSI_ULONG. The number of access methods logically combined in the ACR.

unACRID

Not used.

### 5.5.3 GCAcr

Structure indicating access control conditions for all Generic Container operations:

- Create Container
- Delete Container
- Read Tag
- Read Value
- Update Value

For further information, see the Government Smart Card Interoperability Specification GSC-IS v2.1 http://csrc.nist.gov/publications/nistir/nistir-6887.pdf, section 4.6.3.

```
typedef struct strctGCacr {
BSIAcr strctCreateACR;
BSIAcr strctDeleteACR;
BSIAcr strctReadTagListACR;
BSIAcr strctReadValueACR;
BSIAcr strctUpdateValueACR;
} GCacr;

typedef GCacr GCAcr;
```

```
typedef GCAcr BSI_GC_ACR;
typedef BSI_GC_ACR BSI_PTR BSI_GC_ACR_PTR;
```

### 5.5.4 CRYPTOAcr

Structure indicating access control conditions for all cryptographic operations:

- Get challenge
- Internal authenticate
- PKI compute
- Create container
- Delete container
- Read tag list
- Read value
- Update value

For further information, see the Government Smart Card Interoperability Specification GSC-IS v2.1 http://csrc.nist.gov/publications/nistir/nistir-6887.pdf, section 4.7.5.

```
typedef struct strctCRYPTOacr {
BSIAcr strctGetChallengeACR;
BSIAcr strctInternalAuthenticateACR;
BSIAcr strctPkiComputeACR;
BSIAcr strctCreateACR;
BSIAcr strctDeleteACR;
BSIAcr strctReadTagListACR;
BSIAcr strctReadValueACR;
BSIAcr strctUpdateValueACR;
} CRYPTOacr;

typedef CRYPTOacr CRYPTOAcr;
typedef CRYPTOAcr BSI_CRYPTO_ACR;
typedef BSI_CRYPTO_ACR * BSI_CRYPTO_ACR_PTR;
```

# 6.0 Utility Provider

The Utility Provider is used to get access to services in the other BSI providers. For example, in order to invoke the `gscBSIGC...()` and `gscBSIPKI...()` methods, an application first has to invoke `gscBsiUtilAcquireContext()`.

## 6.1 Access Control Rules

No specific access control rule is required to use the Utility Provider. Most Utility Provider Module functions do not need a Security Context. For example, the function `gscBsiUtilAcquireContext()` need not be called before calling one of the Utility Provider Module functions.

## 6.2 Return Codes

For further information, see section 9.0 Return Codes on page 61 or Table 4-1 in section 4.4 of Government Smart Card Interoperability Specification GSC-IS v2.1 http://csrc.nist.gov/publications/nistir/nistir-6887.pdf.

## 6.3 Function Summary

| Method | Description |
|---|---|
| gscBsiUtilAcquireContext() | Establishes the Security Context required by an SKI, GC, or Cryptographic Provider Module. |
| gscBsiUtilBeginTransaction() | Initiates an exclusive transaction with card. |
| gscBsiUtilConnect() | Connects to the card. |
| gscBsiUtilDisconnect() | Disconnects from the card. |
| gscBsiUtilEndTransaction() | Completes a previously started exclusive transaction. |
| gscBsiUtilGetCardProperties() | Retrieves card information, such as the serial number and card capabilities. |
| gscBsiUtilGetCardStatus() | Retrieves the status of the card whose handle is specified. |
| gscBsiUtilGetExtendedErrorText() | Retrieves additional information about an error. |
| gscBsiUtilGetReaderList() | Retrieves reader list. |
| gscBsiUtilGetVersion() | Retrieves version of Utility Provider. |
| gscBsiUtilPassthru() | Sends an APDU to the card or applet and returns its response. |
| gscBsiUtilReleaseContext() | Releases previously established Security Context. |

## 6.4 Function Details

### 6.4.1 gscBsiUtilAcquireContext()

Establishes the security context required by the SKI, PKI or GC Provider Module by executing the Access Control Rule (ACR) specific to that service.

This ACR is defined as a list of `BSIAuthenticator` structures. Each structure defines an access method. A successful call to `gscBsiUtilAcquireContext()` grants access rights for functions called after it.

**Syntax**:

```
BSI_RETURN gscBsiUtilAcquireContext(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_AUTH_PTR pstrAuthenticator,
IN BSI_ULONG unAuthNb);
```

**Parameters**:

`hCard`

[in] Card connection handle that was returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the applet instance corresponding to the Provider Module providing the requested protected service.

`unAIDLen`

[in] Length of the AID value including the NULL terminator.

`pstrAuthenticator`

[in] Array of structures containing authenticators specifying the access control rule required to access a service. Each authenticator corresponds to an ACR returned by the ACR discovery function provided by each Provider Module:

- `gscBsiGcGetContainerProperties()` for the Generic Container Provider Module (GC applets).
- `gscBsiGetCryptoProperties()` for the Cryptographic Provider Module (PKI or SKI applets).

`unAuthNb`

[in] Number of `BSIAuthenticator` structures contained in the `pstrAuthenticator` array. This permits authenticator chaining.

If two authentication methods are chained, as with XAUTH_THEN_PIN, then set this parameter to 2. The provider is responsible for verifying the authentication methods in sequence.

**Returns**:

BSI_OK if successful.

| |
|---|
| BSI_ACR_NOT_AVAILABLE |
| BSI_BAD_AID |
| BSI_BAD_AUTH |
| BSI_BAD_HANDLE |
| BSI_CARD_REMOVED |
| BSI_PIN_BLOCKED |
| BSI_UNKNOWN_ERROR |

### 6.4.2 gscBsiUtilBeginTransaction()

Starts an exclusive transaction with card.

**Syntax**:

```
BSI_RETURN gscBsiUtilBeginTransaction(
IN BSI_BYTE_PTR hCard,
IN BOOL blType);
```

**Parameters**:

hCard

[in] Card connection handle obtained by a successful call to the gscBsiUtilConnect() function.

blType

[in] Specifies the type of transaction call:

- TRUE transaction in blocking mode.
- FALSE transaction in non-blocking mode.

**Note**: Only blocking mode transactions are supported. Do not set the blType parameter to FALSE. If you do, the function returns BSI_NO_SPSSERVICE and no transaction is started.

**Returns**:

BSI_OK if successful.

| |
|---|
| BSI_BAD_HANDLE |
| BSI_NO_SPSSERVICE |
| BSI_NOT_TRANSACTED |
| BSI_SC_LOCKED |
| BSI_UNKNOWN_ERROR |

### 6.4.3 gscBsiUtilConnect()

Establishes a connection with the card that is currently inserted in the specified reader so that you can communicate with the card. Returns an error if no smart card is inserted.

**Syntax**:

```
BSI_RETURN gscBsiUtilContext(
IN BSI_BYTE_PTR uszReaderName,
IN BSI_ULONG unReaderNameLen,
OUT BSI_CARD_HANDLE_PTR hCard);
```

**Parameters**:

`uszReaderName`

[in] MBCS-formatted name of the reader that contains the card to which to connect. This parameter can be NULL, in which case the function connects to the first reader in which it detects a smart card inserted.

`unReaderNameLen`

[in] Length in bytes, including the NULL terminator of the reader name containing the card to which to connect. This parameter is ignored if the reader name is null.

If `unReaderNameLen` is 0, then the function connects to the first reader in which it detects a smart card inserted.

`hCard`

[out] Card connection handle returned by the function. The `hCard` handle returned is required in all other BSI API functions that communicate with the card.

**Returns**:

```
BSI_OK if successful.
BSI_BAD_PARAM
BSI_CARD_ABSENT
BSI_TIMEOUT_ERROR
BSI_UNKNOWN_ERROR
BSI_UNKNOWN_READER
```

### 6.4.4 gscBsiUtilDisconnect()

Releases the connection handle established with the card using `gscBsiUtilConnect()`.

**Syntax**:

```
BSI_RETURN gscBsiUtilDisconnect(
IN BSI_CARD_HANDLE hCard);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function. After this call, the card connection handle is no longer valid and should not be used in subsequent calls to other functions.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_HANDLE
BSI_CARD_REMOVED
BSI_UNKNOWN_ERROR
```

### 6.4.5 gscBsiUtilEndTransaction()

Completes a previously started exclusive transaction that allows other blocked applications to begin or resume interactions with the card.

**Syntax**:

```
BSI_RETURN gscBsiUtilEndTransaction(
IN BSI_CARD_HANDLE hCard);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_HANDLE
BSI_NO_SPSSERVICE
BSI_NOT_TRANSACTED
BSI_UNKNOWN_ERROR
```

### 6.4.6 gscBsiUtilGetCardProperties()

Retrieves card CUID and capability.

**Syntax**:

```
BSI_RETURN CALLBACK gscBsiUtilGetCardProperties(
IN BSI_CARD_HANDLE hCard,
INOUT BSI_BYTE_PTR uszCCCUniqueID,
INOUT BSI_ULONG_PTR punCCCUniqueIDLen,
OUT BSI_ULONG_PTR punCardCapability);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszCCCUniqueID`

[in, out] Buffer for Card Capability Container version. Returned buffer is empty in this version.

`punCCCUniqueIDLen`

[in, out] Card Capability Container unique ID length. 0 in this version.

`punCardCapability`

[out] Card capabilities. The value returned is a bitwise OR of one or more of the following constants:

```
#define BSI_GCCDM 0x00000001
#define BSI_SKI   0x00000002
#define BSI_PKI   0x00000004
#define BSI_GCCDM_EXT    0x00000008
#define BSI_SKI_EXT 0x00000010
#define BSI_PKI_EXT 0x00000020
```

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_INSUFFICIENT_BUFFER
BSI_NO_CARDSERVICE
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

### 6.4.7 gscBsiUtilGetCardStatus()

Checks whether a given card handled is associated with a card that is inserted into a powered-up reader.

**Syntax**:

```
BSI_RETURN gscBsiUtilGetCardStatus(
IN BSI_CARD_HANDLE hCard);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

**Returns**:

`BSI_OK`

Card is present, powered up, and was not removed since the handle was acquired.

`BSI_BAD_HANDLE`

Supplied handle is incorrect.

`BSI_CARD_REMOVED`

Card was present but is present no longer.

`BSI_UNKNOWN_ERROR`

### 6.4.8 gscBsiUtilGetExtendedErrorText()

Retrieves additional information about the error code returned by last function call.

**Syntax:**

```
BSI_RETURN CALLBACK gscBsiUtilGetExtendedErrorText(
IN BSI_CARD_HANDLE hCard,
OUT BSI_CHAR uszErrorText[BSI_ERROR_TEXT_LEN]);
```

**Parameters:**

`hCard`

[in] This parameter is not used.

`uszErrorText`

[out] Fixed length buffer containing an implementation specific error text string.

**Returns:**

`BSI_OK` if successful.

| |
|---|
| BSI_BAD_PARAM<br>BSI_NO_TEXT_AVAILABLE |

### 6.4.9 gscBsiUtilGetReaderList()

Retrieves a list of readers.

**Syntax**:

```
BSI_RETURN gscBsiUtilGetReaderList(
INOUT BSI_BYTE_PTR puszReaderList,
INOUT BSI_ULONG_PTR punReaderListLen);
```

**Parameters**:

`puszReaderList`

[in, out] Multi-string (MBCS formatted) list of reader names. Each reader name is terminated with a '\0' character and two '\0' characters end the list. The calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13.

`punReaderListLen`

[in] Size in bytes of the buffer pointed to by `puszReaderList` parameter.

[out] Set by the function in bytes for the buffer to allocate to receive the reader list.

**Returns**:

`BSI_OK` if successful.

| |
|---|
| BSI_BAD_PARAM<br>BSI_INSUFFICIENT_BUFFER<br>BSI_UNKNOWN_ERROR |

### 6.4.10 gscBsiUtilGetVersion()

Retrieves version of the Utility Provider.

**Syntax**:

```
BSI_RETURN gscBsiUtilGetVersion(
OUT unsigned char *uszVersion,
INOUT unsigned long *unVersionLength);
```

**Parameters**:

`uszVersion`

[out] Version of the Provider in the format “major,minor,revision,build_number”. Version is a null-terminated ASCII-encoded string. The calling application must pre-allocate this buffer. See Determining the Size of Buffer to Allocate on page 13.

`unVersionLength`

[in, out] Length of version string.

**Returns**:

`BSI_OK` if successful.

`BSI_BAD_PARAM`
`BSI_INSUFFICIENT_BUFFER`
`BSI_UNKNOWN_ERROR`

### 6.4.11 gscBsiUtilPassthru()

Provides a passthrough service so an application can send APDUs to an applet instance while still using the provider’s other services. The application can use the passthrough to establish a Global Platform (GP) secure channel with the card.

**Note**: Using an APDU may cause side effects in ActivClient. In particular, changing the PIN or ActivClient buffers using the BSI API pass-through is not recommended because it may desynchronize the values cached by ActivClient and result in an inconsistent state.

**Syntax**:

```
BSI_RETURN gscBsiUtilPassthru(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszCardCommand,
IN BSI_ULONG unCardCommandLen,
INOUT BSI_BYTE_PTR pusCardResponse,
INOUT BSI_ULONG_PTR unCardResponseLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszCardCommand`

[in] Binary, ASCII hex-encoded APDU command to be sent to the card by the calling application. This allows an application to enforce APDU level security like a secure channel.

`unCardCommandLen`

[in] Length of APDU command, including the NULL terminator to be sent to the card by the calling application.

`pusCardResponse`

[in, out] APDU response returned by the card to the calling application. Buffer must be pre-allocated by the caller with a fixed size of (255+2)*2 + 1.

This size is defined by ISO 7816-4 and it includes a status word and a NULL terminator. The APDU value is binary data and the returned data is ASCII hex encoded.

`unCardResponseLen`

[in, out] Length of the APDU response, including the NULL terminator returned by the card.

**Returns**:

`BSI_OK` if successful.

| |
|---|
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_CARD_REMOVED |
| BSI_INSUFFICIENT_BUFFER |
| BSI_SC_LOCKED |
| BSI_UNKNOWN_ERROR |

## 6.4.12 gscBsiUtilReleaseContext()

Releases the Security Context previously established with a call to the `gscBsiUtilAcquireContext()` function and loses the corresponding access rights.

**Syntax**:

```
BSI_RETURN gscBsiUtilReleaseContext(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex-encoded AID value of the applet instance used in the `gscBsiUtilAcquireContext()` call.

`unAIDLen`

[in] Length of the AID value including the NULL terminator.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

## 6.5 Extended Functionality

HID Global provides additional functionality with the following four functions that are not part of the BSI API specification.

### 6.5.1 gscXsiUtilGetCardATR()

Retrieves the card's ATR (Answer To Reset). The ATR contains information used to identify a card.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilGetCardATR(
IN BSI_CARD_HANDLE hCard,
INOUT BSI_BYTE_PTR pusATR,
INOUT BSI_ULONG *punATRLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`pusATR`

[in, out] Buffer to be filled by the function with the card ATR. The calling application must pre-allocate this buffer.

The returned buffer is not ASCII hex-encoded.

For further information, see Determining the Size of Buffer to Allocate on page 13.

`punATRLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the returned value length.

**Returns:**

`BSI_OK` if successful.

| |
|---|
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_CARD_REMOVED |
| BSI_INSUFFICIENT_BUFFER |
| BSI_NO_CARDSERVICE |
| BSI_SC_LOCKED |
| BSI_UNKNOWN_ERROR |

### 6.5.2 gscXsiUtilGetCuid()

Retrieves the card's unique identifier computed from Chip Identifiers.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilGetCuid(
IN BSI_CARD_HANDLE hCard,
INOUT BSI_BYTE_PTR pusSerialNumber,
INOUT BSI_ULONG_PTR punSerialNumberLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`pusSerialNumber`

[in, out] Buffer to be filled by the function with the card's unique identifier computed from Chip identifiers. The calling application must pre-allocate this buffer.

The returned buffer is not ASCII hex-encoded.

For further information, see Determining the Size of Buffer to Allocate on page 13.

`punSerialNumberLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the returned value length.

**Returns:**

BSI_OK if successful.

| |
|---|
| BSI_CARD_REMOVED |
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_INSUFFICIENT_BUFFER |
| BSI_UNKNOWN_ERROR |

### 6.5.3 gscXsiUtilGetIdentifier()

Returns a card identifier. Each card identifier is unique among all smart cards, whatever the card type. The identifier value may be different from the value returned by gscXsiUtilGetCuid(), depending upon the card profile.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilGetIdentifier(
IN BSI_CARD_HANDLE hCard,
INOUT BSI_BYTE_PTR pusIdentifier,
INOUT BSI_ULONG_PTR punIdentifierLen );
```

**Parameters**:

hCard

[in] Smart card connection handle returned by a successful call to the gscBsiUtilConnect() function.

pusIdentifier

[in, out] Buffer to be filled by the function with the card's unique identifier. The calling application must pre-allocate this buffer. The returned buffer is not ASCII hex-encoded. For further information, see Determining the Size of Buffer to Allocate on page 13.

punIdentifierLen

[in, out] Pointer to an unsigned long integer to be set by the function with the returned value length.

**Returns**:

BSI_OK if successful.

| |
|---|
| BSI_CARD_REMOVED |
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_INSUFFICIENT_BUFFER |
| BSI_UNKNOWN_ERROR |

### 6.5.4 gscXsiUtilChangePIN()

Change the card's PIN code.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilChangePIN(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_BYTE_PTR usOldPIN,
IN BSI_ULONG unOldPINLen,
IN BSI_BYTE_PTR uszNewPIN,
IN BSI_ULONG unNewPINLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex-encoded AID value of the PIN applet managing the PIN code to be changed.

`unAIDLen`

[in] Length of the PIN applet AID value including the NULL terminating character.

`usOldPIN`

[in] Old PIN code value. This parameter must be ASCII hex encoded.

`unOldPINLen`

[in] Length of old PIN code including the NULL terminating character.

`usNewPIN`

[in] New PIN code value. This parameter shall be ASCII hex encoded.

`unNewPINLen`

[in] Parameter length of new PIN code including the NULL terminating character.

**Returns**:

`BSI_OK` if successful.

```
BSI_ACR_NOT_AVAILABLE
BSI_BAD_AID
BSI_BAD_AUTH
BSI_BAD_HANDLE
```

| |
|---|
| BSI_CARD_REMOVED<br>BSI_PIN_BLOCKED |
| BSI_UNKNOWN_ERROR |

### 6.5.5 gscXsiUtilIsPINVerified()

Checks if the PIN code is verified. (Verifies that a PIN code is already validated.)

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilIsPINVerified(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
OUT int* pnLeftPINTries);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex-encoded AID value of the PIN applet managing the PIN status to be checked.

`unAIDLen`

[in] Length of the PIN applet AID value including the NULL terminating character.

`pnLeftPINTries`

[out] Pointer to an unsigned long integer to be set by the function with the number of remaining wrong attempts before the card's PIN code is locked. If the PIN is already verified, then this parameter is set to -1.

**Returns**:

`BSI_OK` if successful.

| |
|---|
| BSI_ACR_NOT_AVAILABLE<br>BSI_BAD_AID<br>BSI_BAD_AUTH<br>BSI_CARD_REMOVED<br>BSI_BAD_HANDLE<br>BSI_PIN_BLOCKED<br>BSI_UNKNOWN_ERROR |

### 6.5.6 gscXsiNotifySynchronizationStart()

Starts to notify synchronization

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiNotifySynchronizationStart(
IN BSI_CARD_HANDLE hCard);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_BAD_HANDLE
BSI_UNKNOWN_ERROR
```

### 6.5.7 gscXsiNotifySynchronizationEnd()

Stops to notify synchronization

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiNotifySynchronizationEnd(
IN BSI_CARD_HANDLE hCard);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_BAD_HANDLE
BSI_UNKNOWN_ERROR
```

### 6.5.8 gscXsiUtilGetCardManagerState()

Retrieves the Card Manager state.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilGetCardManagerState (
IN BSI_CARD_HANDLE hCard
OUT BSI_INT_PTR piState);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`piState`

[out] Pointer to the retrieved card manager state: BSI_CARD_OP_READY or BSI_CARD_LOCKED.

**Returns**:

`BSI_OK` if successful.

```
BSI_BAD_PARAM
BSI_BAD_HANDLE
BSI_CARD_LOCKED
BSI_CARD_REMOVED
BSI_UNKNOWN_ERROR
```

### 6.5.9 gscXsiUtilGetForceChangePIN()

Retrieves the Force Change PIN flag.

**Syntax**:

```
BSI_RETURN CALLBACK gscXsiUtilGetForceChangePIN(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
OUT BOOL* pbForceChangePIN);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex-encoded AID value of the PIN applet managing the PIN status to be checked.

`unAIDLen`

[in] Length of the PIN applet AID value including the NULL terminating character.

`pbForceChangePIN`

[out] ] Pointer to bool to be set by the function with the Force Change PIN flag value.

**Returns**:

`BSI_OK` if successful.

| |
|---|
| BSI_BAD_PARAM |
| BSI_BAD_HANDLE |
| BSI_BAD_AID |
| BSI_CARD_REMOVED |
| BSI_UNKNOWN_ERROR |

# 7.0 Generic Container Provider

A Generic Container is a protected storage area on a smart card.

The Generic Container (GC) Provider makes it possible to create, delete, read, and update collections of {tag, length, value} data items stored on a card within a container. Each of these actions is protected with specific access controls.

The Generic Container Provider is compatible with the Common Access Card (CAC) demographic information data and J.8 identity information data models. The BSI API GC services can be used to read this information.

The Generic Provider can also be used to read/store any other application-specific data such as health, certification, training, or biometric data or rosters.

## 7.1 Discovery Functions

The GC Provider offers a number of discovery functions to applications:

- Discover the list of data items managed by the container.
- Discover the properties of the container in terms of size, access control rules, and so on.

## 7.2 Differentiating Between Data Sets

Each container has its own set of access control rules for create, delete, read, and update operations. If data sets require different security settings, you must place them in separate containers with different access control rules. Within a container, different keys can be defined for read and write access. The available size of the container limits the number and the size of data items within the card.

Each data item in a Generic Container is tagged. This tag allows the GC Provider to access the data item value stored on the card. The Provider also offers the service of listing the tags managed by a container, thus allowing an application to discover what container is managed and to pick the value it needs by referring to it by its tag.

## 7.3 Access Control Rules

Before it calls any Generic Container Provider function, the client application must first establish a security context with the Generic Container Provider Module by:

- Calling the discovery function, `gscBsiGcGetContainerProperties()`, to return a GCAcr structure defining the access control rules for a GC service.
- Establishing the security context according to the retrieved ACR using the `gscBsiUtilAcquireContext()` function.

### 7.3.1 Compatibility with the Card Edge Interface

In order to use data stored in a Generic Container Applet instance that is accessible to GSI/XSI applications, the application must:

- Use the Card Edge API as specified in the GSC-IS standard documentation.
- Know the format of the data that is stored in the Generic Container, including:
  - Byte order
  - Single vs. multi-byte data storage

### 7.3.2 SIMPLE-TLV Data Object

Internal Tag Length Value (TLV) format is required to access the card services through the Card Edge directly. Each SIMPLE-TLV data object consists of two or three consecutive fields:

- TLV tag field

  The tag field identifies the data stored in the corresponding V buffer. The tag field T consists of a single byte encoding only a number between 1 and 254. No class or construction types are coded.

- TLV length field

  The length field consists of one or three consecutive bytes.

  The meaning of the length field changes depending upon the value of the leading byte in the length field:

  - If the leading byte of the length field is in the range from 00 to FE, the length field is the leading byte and consists of a single byte integer L with a value between 0 and 254.
    - If L is not equal to 0, the value field V consists of L consecutive bytes.
    - If L equals 0, or if a tag is omitted from its file or buffer, the data object is empty, and there is no value field for that tag.
  - If the leading byte equals FF, the length field continues through the two subsequent bytes, which encode an integer L with a value between 0 and 65,535. (The leading field is upper byte of the three-byte integer L.)

- The V field

  The V field contains the data item for the corresponding Tag. Its format and encoding is left to the calling application. Its length is defined by the L field.

### 7.3.3 Byte Order

The byte order in V depends upon the application.

T is always only 1 byte long.

In the L field, the byte ordering of multi-byte attributes is LSB first.

## 7.4 Generic Container Data Types and Structures

The Generic Container Provider operates on tagged data. The data structures it uses are:

- GCTag
- BSI_GCT

The container size structure is `GCContainerSize`. A pointer to a `GCContainerSize`, BSI_GC_SIZE_PTR, is also provided.

## 7.5 Function Summary

This section includes a full description of each function in the Generic Container Provider Module.

### 7.5.1 Access Control

| Method | Description |
|---|---|
| gscBsiGcGetContainerProperties() | A container may contain multiple data items each identified by a tag. The access control applies to all data items stored in this container. This function retrieves access conditions that enable reading and updating of tags and their values for the specified container. Access conditions returned are shared by all data items present in the container. Retrieves container sizes. |

### 7.5.2 Data Management Functions

| Method | Description |
|---|---|
| gscBsiGcDataCreate() | Creates a new data item in a specific container. This will store a value and a tag. |
| gscBsiGcDataDelete() | Deletes a data item from the specified container. |
| gscBsiGcReadTagList() | Retrieves the list of tags from the specified container. |
| gscBsiGcReadValue() | Retrieves the value identified by a given tag in a specific container. |
| gscBsiGcUpdateValue() | Updates the value identified by a given tag in a specific container with provided value. |

## 7.6 Function Details

### 7.6.1 gscBsiGcDataCreate()

Creates a new data item {tag, length, value} in the specified container. In order to create a data item successfully, the client application must first establish a Security Context with the Generic Container Provider Module for the requested service: Create.

**Syntax**:

```
BSI_RETURN gscBsiGcDataCreate(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_GCT ucTag,
IN BSI_BYTE_PTR uszValue,
IN BSI_ULONG unValueLen);
```

**Parameters:**

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] AID value of the applet (Generic Container on the card) in which the new data item is to be created. This parameter shall be ASCII hex encoded.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`ucTag`

[in] Tag identifying the value of the new data item to create.

`uszValue`

[in] New value to create. This parameter shall be ASCII hex encoded.

`unValueLen`

[in] Length of the encoded value string that includes the NULL terminator.

**Returns**:

```
BSI_OK BSI_ACCESS_DENIED
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
```

BSI_IO_ERROR
BSI_NO_CARDSERVICE
BSI_NO_MORE_SPACE
BSI_SC_LOCKED
BSI_TAG_EXISTS
BSI_UNKNOWN_ERROR

### 7.6.2 gscBsiGcDataDelete()

Deletes the data item identified by a given tag and its associated value from the specified container. In order to delete a data item successfully, the client application must first establish a Security Context with the Generic Container Provider Module for the requested service: Delete.

**Syntax**:

```
BSI_RETURN gscBsiGcDataDelete (
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR pszAID,
IN BSI_ULONG unAIDLen,
IN BSI_GCT ucTag);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] AID value of the applet (Generic Container on the card) in which the data item should be deleted. This parameter shall be ASCII hex encoded.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`ucTag`

[in] Tag identifying the value of the data item to delete.

**Returns**:

BSI_OK
BSI_ACCESS_DENIED
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_BAD_TAG

```
BSI_CARD_REMOVED
BSI_IO_ERROR
BSI_NO_CARDSERVICE
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

### 7.6.3 gscBsiGcGetContainerProperties()

This method returns three types of information:

- The Generic container size,
- The Generic container version,
- The Generic container access control rules for the GC services. The access control rules apply to all data items stored in the specified container.

**Syntax**:

```
BSI_RETURN gscBsiGcGetContainerProperties(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
OUT BSI_GC_ACR_PTR strctGCacr,
OUT BSI_GC_SIZE_PTR strctContainerSizes,
OUT BSI_BYTE_PTR containerVersion);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex-encoded AID value of the applet (Generic Container on the card) whose properties should be retrieved.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`strctGCacr`

[out] Pointer to a structure to be set by the function with the Access Control Rules defined for the services provided by this Generic Container applet. Caller must allocate this structure before passing it to this function. The structure is not populated if the input pointer is NULL.

`strctContainerSizes`

[out] Structure indicating the sizes of the two storage areas of the container. The caller must allocate this structure. This parameter is not returned if the pointer is set to NULL.

`containerVersion`

[out] Container version in ASCII format. The calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13.

**Returns**:

| |
|---|
| BSI_OK |
| BSI_BAD_AID |
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_CARD_REMOVED |
| BSI_NO_CARDSERVICE |
| BSI_SC_LOCKED |
| BSI_UNKNOWN_ERROR |

### 7.6.4 gscBsiGcReadTagList()

Retrieves the list of tags from the specified container.

In order to read a list of tags successfully, the client application must first establish a Security Context with the Generic Container Provider Module for the requested service: Read Tag List.

**Syntax**:

```
BSI_RETURN gscBsiGcReadTagList(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
INOUT BSI_GCT_PTR TagArray,
INOUT BSI_ULONG_PTR unNbTags);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the applet (Generic Container on the card) in which the tag list should be read.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`TagArray`

[in, out] Buffer to be filled by the function with the list of tags stored in the Generic Container. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13.

`unNbTags`

[in, out] Pointer to a long integer to be set by the function with the returned tag count.

**Returns**:

```
BSI_OK
BSI_ACCESS_DENIED
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_NO_CARDSERVICE
BSI_INSUFFICIENT_BUFFER
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

### 7.6.5 gscBsiGcReadValue()

Returns the value identified by a given tag in a specific container.

In order to read the value identified by a given tag successfully, the client application must first establish a Security Context with the Generic Container Provider Module for the requested service: Read Value.

**Syntax**:

```
BSI_RETURN gscBsiGcReadValue(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_GCT ucTag,
INOUT BSI_BYTE_PTR uszValue,
INOUT BSI_ULONG_PTR unValueLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the applet (Generic Container on the card) in which the value should be read.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`ucTag`

[in] Tag identifying the value to be read.

`uszValue`

[in, out] Buffer to be filled by the function with the value identified by the tag provided in the `ucTag` parameter. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13. Value returned in this buffer is ASCII hex encoded.

`punValueLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the returned value length. This includes the NULL terminator.

**Returns**:

BSI_OK
BSI_ACCESS_DENIED
BSI_BAD_AID BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_BAD_TAG
BSI_CARD_REMOVED
BSI_INSUFFICIENT_BUFFER
BSI_NO_CARDSERVICE
BSI_IO_ERROR
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR

### 7.6.6 gscBsiGcUpdateValue()

Updates the value identified by a given tag in a specific container.

In order to update the value identified by a given tag successfully, the client application must first establish a Security Context with the Generic Container Provider Module for the requested service: Update Value.

**Syntax**:

```
BSI_RETURN gscBsiGcUpdateValue(
IN BSI_CARD_HANDLE hCard, IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_GCT ucTag,
```

```
IN BSI_BYTE_PTR uszValue, IN BSI_ULONG unValueLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the applet (Generic Container on the card) in which the value should be updated.

`unAIDLen`

[in] Length of the Generic Container AID value including the NULL terminator.

`ucTag`

[in] Tag identifying the value to be updated.

`uszValue`

[in] ASCII hex encoded buffer containing the new value to associate with the tag provided in `ucTag` parameter.

`unValueLen`

[in] Length of the new value including the NULL terminator.

**Returns**:

```
BSI_OK
BSI_ACCESS_DENIED
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_BAD_TAG
BSI_CARD_REMOVED
BSI_IO_ERROR
BSI_NO_CARDSERVICE
BSI_NO_MORE_SPACE
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

# 8.0 Cryptographic Provider

## 8.1 About the Cryptographic Provider

The Cryptographic Provider provides fundamental cryptographic services such as random number generation, authentication, and digital signature generation.

## 8.2 Access Control Rules

Before it calls any Cryptographic Provider function, the client application must first establish a security context with the Cryptographic Provider Module by:

- Calling the discovery function, `gscBsiGetCryptoProperties()`, to return a `CRYPTOAcr` structure defining the access control rules for a CP service.
- Establishing the security context according to the retrieved ACR using the `gscBsiUtilAcquireContext()` function.

### 8.2.1 Function Summary

#### 8.2.1.1 Access Control

| Method | Description |
|---|---|
| gscBsiGetCryptoProperties() | Retrieves the access control rules and private cryptographic key length managed by the Cryptographic Provider Module. |

#### 8.2.1.2 Challenge

| Method | Description |
|---|---|
| gscBsiGetChallenge() | Retrieves a randomly generated challenge from the card. |

#### 8.2.1.3 SKI

| Method | Description |
|---|---|
| gscBsiSkiInternalAuthenticate() | Computes the symmetric key cryptography crypto- gram in response to a challenge generated by the client application. |

#### 8.2.1.4 PKI

| Method | Description |
|---|---|
| gscBsiPkiCompute() | Performs a private key computation on a message digest using the private key associated with the specified PKI applet. |
| gscBsiPkiGetCertificate() | Reads the certificate. |

## 8.3 Function Details

### 8.3.1 gscBsiGetChallenge()

Retrieves a randomly generated challenge from the specified applet. Use this function only for strong key management requirements when you do not want the External Authentication key exposed in clear text. It is the first step for a strong authentication of the application to the card. The client application should then return the encrypted challenge to the card through a call to the `gscBsiUtilAcquireContext()` function. For further information about strong authentication, see Access Control Rules on page 10.

**Syntax**:

```
BSI_RETURN gscBsiGetChallenge(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
INOUT BSI_BYTE_PTR uszChallenge,
INOUT BSI_ULONG_PTR unChallengeLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the applet.

`unAIDLen`

[in] Length of the applet AID value including the NULL terminator.

`uszChallenge`

[in, out] Buffer to be filled by the function with the generated challenge. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13. This buffer is returned ASCII hex encoded.

`unChallengeLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the returned challenge length. This includes the NULL terminator.

**Returns**:

BSI_OK
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_INSUFFICIENT_BUFFER
BSI_NO_CARDSERVICE
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR

## 8.3.2 gscBsiGetCryptoProperties()

Retrieves the access control rules and private cryptographic key length managed by the Cryptographic Provider Module.

**Syntax**:

```
BSI_RETURN gscBsiGetCryptoProperties(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
INOUT BSI_CRYPTO_ACR_PTR strctCRYPTOacr,
INOUT BSI_ULONG_PTR punKeyLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the cryptographic (PKI or SKI) applet.

`unAIDLen`

[in] Length of cryptographic applet AID value including the NULL terminator.

`strctCRYPTOacr`

[in, out] Pointer to a structure to be set by the function with the access control rules defined for the services provided by this cryptographic applet. Caller must allocate structure. This parameter is not returned if the pointer is set to NULL.

`punKeyLen`

[in, out] Pointer to an unsigned long integer to be set by the function to the key length in bytes. Depending on applet type, it can be:

- Length of RSA key (PKI instance).
- Length of DES key (SKI instance).

If this pointer is set to NULL, then this parameter is not returned.

**Returns**:

```
BSI_OK
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_NO_CARDSERVICE
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

### 8.3.3 gscBsiPkiCompute()

Performs a private key computation on a message digest using the private key associated with the specified PKI applet.

**Syntax**:

```
BSI_RETURN gscBsiPkiCompute(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_BYTE ucAlgoID,
IN BSI_BYTE_PTR uszMessage,
IN BSI_ULONG unMessageLen,
INOUT BSI_BYTE_PTR uszResult,
INOUT BSI_ULONG_PTR unResultLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the PKI applet.

`unAIDLen`

[in] Length of PKI applet AID value including the NULL terminator.

`ucAlgoID`

[in] Algorithm Identifier. In this version of the interoperability specification, the only algorithm supported is RSA_NO_PAD. (For further information, see Cryptographic Algorithms on page 12.)

`uszMessage`

[in] ASCII hex encoded message to be computed.

`unMessageLen`

[in] Length of message to be computed. This value must be equal to the key length returned by the `gscBsiGetCryptoProperties()` function (64 or 128 bytes), plus 1 for the NULL terminator.

`uszResult`

[in, out] Buffer to be filled by the function with the computed message. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13. This buffer is returned ASCII hex encoded.

`unResultLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the length of the computed message length including the NULL terminator.

**Returns**:

| |
|---|
| BSI_OK |
| BSI_ACCESS_DENIED |
| BSI_BAD_AID |
| BSI_BAD_ALGO_ID |
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_CARD_REMOVED |
| BSI_INSUFFICIENT_BUFFER |
| BSI_NO_CARDSERVICES |
| BSI_SC_LOCKED |
| BSI_UNKNOWN_ERROR |

### 8.3.4 gscBsiPkiGetCertificate()

Reads a certificate from the card.

**Syntax**:

```
BSI_RETURN CALLBACK gscBsiPkiGetCertificate(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
OUT BSI_BYTE_PTR uszCertificate,
INOUT BSI_ULONG_PTR unCertificateLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the PKI applet whose certificate is to be read.

`unAIDLen`

[in] Length of the PKI applet AID value including the NULL terminator.

`uszCertificate`

[out] Buffer to be filled by the function with the read certificate. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13. This buffer is returned ASCII hex encoded.

`unCertificateLen`

[in, out] Pointer to an unsigned long integer to be set by the function with the read certificate length. This includes the NULL terminator.

**Returns**:

```
BSI_OK
BSI_ACCESS_DENIED
BSI_BAD_AID
BSI_BAD_HANDLE
BSI_BAD_PARAM
BSI_CARD_REMOVED
BSI_INSUFFICIENT_BUFFER
BSI_NO_CARDSERVICE
BSI_READ_ERROR
BSI_SC_LOCKED
BSI_UNKNOWN_ERROR
```

### 8.3.5 gscBsiSkiInternalAuthenticate()

Computes a symmetric key cryptography authenticator in response to a challenge generated by the client application.

**Syntax**:

```
BSI_RETURN gscBsiSkiInternalAuthenticate(
IN BSI_CARD_HANDLE hCard,
IN BSI_BYTE_PTR uszAID,
IN BSI_ULONG unAIDLen,
IN BSI_BYTE ucAlgoID,
IN BSI_BYTE_PTR uszChallenge,
IN BSI_ULONG unChallengeLen,
OUT BSI_BYTE_PTR uszCryptogram,
INOUT BSI_ULONG_PTR unCryptogramLen);
```

**Parameters**:

`hCard`

[in] Card connection handle returned by a successful call to the `gscBsiUtilConnect()` function.

`uszAID`

[in] ASCII hex encoded AID value of the SKI applet.

`unAIDLen`

[in] Length of the SKI applet AID value including the NULL terminator.

`ucAlgoID`

[in] Algorithm Identifier. In this version of the interoperability specification, the algorithms supported for this operation are DES3_ECB (Algorithm Identifier 0x81) and DES3_CBC (Algorithm Identifier 0x82). For further information, see Cryptographic Algorithms on page 12.

`uszChallenge`

[in] ASCII hex encoded challenge generated by the application and submitted to the card. Set this parameter to NULL if the cryptogram is synchronous.

`unChallengeLen`

[in] The length of the ASCII hex encoded challenge including the NULL terminator. In this version of the specification, the challenge length can be up to 8 bytes. Hence the value of `unChallengeLen` can be up to 17 (2*8+1).

`uszCryptogram`

[out] Buffer to be filled by the function with the computed cryptogram. Calling application must pre-allocate this buffer. For further information, see Determining the Size of Buffer to Allocate on page 13. This buffer is returned ASCII hex encoded.

`unCryptogramLen`

[in, out] Pointer to an unsigned long integer to be set by the function to the cryptogram length computed by the SKI applet. This includes the NULL terminator.

The result of a DES computation is always 8 bytes. Some cards truncate the cryptogram so that the `gscBsiSkiInternalAuthenticate()` function cannot be used to encrypt data.

**Returns**:

| |
|---|
| BSI_OK |
| BSI_ACCESS_DENIED |
| BSI_BAD_AID |
| BSI_BAD_ALGO_ID |
| BSI_BAD_HANDLE |
| BSI_BAD_PARAM |
| BSI_CARD_REMOVED |
| BSI_INSUFFICIENT_BUFFER |
| BSI_NO_CARDSERVICE |
| BSI_SC_LOCKED |
| BSI_TERMINAL_AUTH |
| BSI_UNKNOWN_ERROR |

# 9.0 Return Codes

| Exception | Description |
|---|---|
| BSI_OK | Execution completed successfully. |
| BSI_ACCESS_DENIED | Applicable ACR was not fulfilled. |
| BSI_ACR_NOT_AVAILABLE | Specified ACR is incorrect. |
| BSI_BAD_AID | Specified Application (AID) does not exist. |
| BSI_BAD_ALGO_ID | Specified cryptographic algorithm is not available. |
| BSI_BAD_AUTH | Invalid authentication data. |
| BSI_BAD_HANDLE | Specified card handle is not available. |
| BSI_BAD_PARAM | One or more of specified parameters is incorrect. |
| BSI_BAD_TAG | Invalid tag information. |
| BSI_CARD_ABSENT | Card associated with specified card handle is not present. |
| BSI_CARD_REMOVED | Card associated with specified card handle has been removed. |
| BSI_IO_ERROR | Error encountered during input or output of specified data. |
| BSI_INSUFFICIENT_BUFFER | Buffer allocated by calling application is too small. |
| BSI_NO_CARDSERVICE | Smart card associated with specified card handle does not provide requested service. |
| BSI_NO_MORE_SPACE | Insufficient space in selected container to store specified data. |
| BSI_NO_SPSSERVICE | SPS does not provide requested service. |
| BSI_NO_TEXT_AVAILABLE | No extended error text is available. |
| BSI_NOT_IMPLEMENTED | Function is not implemented. |
| BSI_NOT_TRANSACTED | Current transaction has not ended. |
| BSI_PIN_BLOCKED | PIN is blocked. |
| BSI_SC_LOCKED | Smart card associated with a specified card handle is under exclusive transaction of another client application. |
| BSI_TAG_EXISTS | Tag specified for a create operation already exists in target container. |
| BSI_TERMINAL_AUTH | Card reader has performed a successful Internal Authentication with card. |
| BSI_TIMEOUT_ERROR | Connection could not be established with card before time-out value expired. |
| BSI_UNKNOWN_ERROR | Requested operation has generated an unspecified error. |

| Exception | Description |
|---|---|
| BSI_UNKNOWN_READER | Specified reader does not exist. |

# Appendix A: Terms and Acronyms

This appendix lists terms and acronyms used throughout the full set of the set of technical publications for this product. Not all terms and acronyms appear in all documents in the set.

## A.1. Terms

**Certificate Authority (CA)**

The CA issues and manages security credentials and public keys for message encryption in a networked environment. As part of a Public Key Infrastructure (PKI), a CA checks with a registration authority (RA) to verify information provided by the requestor of a digital certificate. If the RA verifies the requestor's information, the CA issues a certificate.

**ActivID Credential Management System (CMS)**

Formally known as ActivID Card Management System, ActivID CMS is a web-based, smart card, credential and application lifecycle management system. ActivID CMS augments and works in concert with an enterprise's primary identity management infrastructure components, including popular directory, database, and PKI components.

**Challenge**

Random number generated by the server API for authentication of a user in the asynchronous (challenge/response) mode.

**Cryptographic Service Provider (CSP)**

An independent software module that performs cryptography algorithms for authentication, encoding, and encryption.

**Discovery mode**

Discovery mode enables a calling application to find out the size of the data that will be returned to by making a preliminary discovery call and then making a second call after it allocates a buffer large enough to accommodate the data that will be returned.

**End-point card**

The PIV standard defines two interfaces for communicating with PIV cards:

- The PIV transitional interface.
- The PIV end-point interface.

A PIV end-point card is a card that implements the second of these interfaces.

**Note:** The PIV transitional interface is not supported by the PIV API.

**Federal Information Processing Standard (FIPS 140-2)**

FIPS 140-2 is the standard for crypto-module security. FIPS 140-2 level 3 adds additional requirements to FIPS 140-2 level 2. These requirements concern physical security and a trusted path for entering a Cryptographic Service Provider, such as a PIN. FIPS 140-2 level 3 uses local ports and the key pad to enforce such security.

**Federal Information Processing Standard 201 (FIPS 201)**

FIPS 201 is the standard for Personal Identity Verification (PIV) cards defined for US Government employees and contractors.

**Force change PIN flag**

Flag which indicates whether the user must change the PIN on first use of the card.

| | |
|---|---|
| **Integrated circuit chip (ICC)** | The chip on the smart card. |
| **Mini Driver** | Smart card middleware for the Microsoft platform that works with the Microsoft Base Smart Card CSP (Cryptographic Service Provider). The ActivClient Mini Driver replaces the ActivClient CSP available in previous versions. The Mini Driver architecture provides stronger cryptographic services. |
| **My Digital ID Card (MDIDC)** | This CMS component allows end users to access the self-service CMS functions, which includes card and credential management. |
| **One-Time Password (OTP)** | A one-time password is a password used only once to authenticate to remote applications. One-Time Passwords are only present on smart cards issued with SKI credentials. |
| **Personal Identification Number (PIN)** | The Personal Identification Number (PIN) code used to access an HID Global device's services such as Windows PKI logon, remote access and email signature. HID Global devices can only be used after a correct PIN is entered. |
| **Public Key Infrastructure (PKI)** | PKI describes the laws, policies, standards, and software that regulate or manipulate certificates and public and private keys. |
| **Registration Authority (RA)** | RA is an authority in a network that verifies user requests for a digital certificate and instructs the CA to issue it. An RA is part of a PKI, a networked system that enables companies and users to exchange information safely and securely. |
| **Symmetric Key Infrastructure (SKI)** | SKI keys are used to perform strong authentication on remote applications. SKI keys encrypt passwords in:<br>• Synchronous mode (generates 1 password without any challenge. The server uses the same method to create a password than the smart card)<br>• Asynchronous: encrypts a challenge |
| **Standalone smart card** | Smart card with pre-loaded applets issued by the manufacturer. |
| **Unlock code** | Value that the card holder needs to provide in order to unlock a locked smart card. Depending upon the smart card unlock mechanism, the unlock code may or may not be different from the unlock key. |
| **Verification** | Process in which a signature that was produced by the signing operation is verified. |
| **Weak PIN** | A weak PIN is a PIN in which:<br>• The length is less than three characters or digits, or<br>• The difference between each character or digit and the following one is a constant.<br>For example, a PIN that is a sequence of the same number (1111) or an increasing/decreasing sequence of numbers (1234, 4321) is a weak PIN. |

## A.2. Acronyms

**CA** Certificate Authority

**CAC** Common Access Card (for the United States Department of Defense)

**CSP** Cryptographic Service Provider

**CUID** Card Unique Identifier

CUID is a number that uniquely identifies a card.

**FIPS** Federal Information Processing Standard

**GAL** Global Address List

**GP** GlobalPlatform

Replaces OpenPlatform (OP).

**OTP** One-Time Password

**PKI** Public Key Infrastructure

**PIV** Personal Identity Verification.

Smart card issued by the United States government to federal employees and contractors.

**RA** Registration Authority

**SKI** Symmetric Key Infrastructure

