MUSCLE Cryptographic Card Edge Definition for Java[1] Enabled Smartcards

David Corcoran corcoran@linuxnet.com
Tommaso Cucinotta cucinotta@sssup.it

This document is provided on an as-is basis. Neither the authors nor the MUSCLE project are responsible for any mishaps that may occur from the use of this document. This document may be distributed freely but modifications must be returned to the authors and the authors' names must retain on the document as original authors.

[1] *Java® and Java Card® are trademarks of Sun Microsystems, Inc.*

Change Log:

Version 1.0

November 22, 2000

Original writing, Dave Corcoran, corcoran@linuxnet.com

Version 1.0.1

July 16, 2001

Function clarification, added return code lookup table, Alex Russell, alex@netWindows.org

Version 1.1.0

Sept 10, 2001

Added ACLs, KeyBlob definitions.

Tommaso Cucinotta, cucinotta@sssup.it

David Corcoran, corcoran@linuxnet.com

Version 1.2.0

Sept 25, 2001

Allocated instruction codes

Added List commands

Added ISO Verify compatibility

Version 1.2.1

October 4, 2001

Modified some instruction bytes

Added GetStatus command

Fixed some global defines

First release with Alpha implementation

Table of Contents

Document Scope

The scope of this document is to provide a definition of command set to provide base cryptographic functionality through and abstract interface using Java enabled smartcards and cryptographic tokens.

Smartcards require large amounts of complex middleware that communicates with the card and exports the card's functionality to the host. These cards typically vary from release to release so this middleware generally is in constant change. Currently each card must have it's own CSP (crypto/card service provider) on the host creating large support problems and security trust well beyond most OS vendor's preferences.

Using this applet approach, it is required that only one host CSP be written for the middleware, thus reducing the time spent migrating to new card releases and vastly reducing the number of CSP's on the host. At the time of this writing, this definition will be supported on Java Card 2.1 compliant cards. The applet will be loaded on the card with a static application identifier (AID) and the host based CSP will communicate to the card through this applet. The Java Card API's support a wide array of cryptographic capability including both symmetric and asymmetric functions, random number generation, key generation/management, and PIN management. Although the scope of this document describes Java Card’s, any programmable smartcard can work with this definition.

This specification was not written to encompass all the functionality of the Java Card platform but rather to provide a minimum subset of calls to enable most cryptographic applications the ability to make use of the smartcards as a key token. This is an evolving specification so future commands and calls might be added to provide compatibility with other standards such as PKCS-15 and existing infrastructures on other platforms.

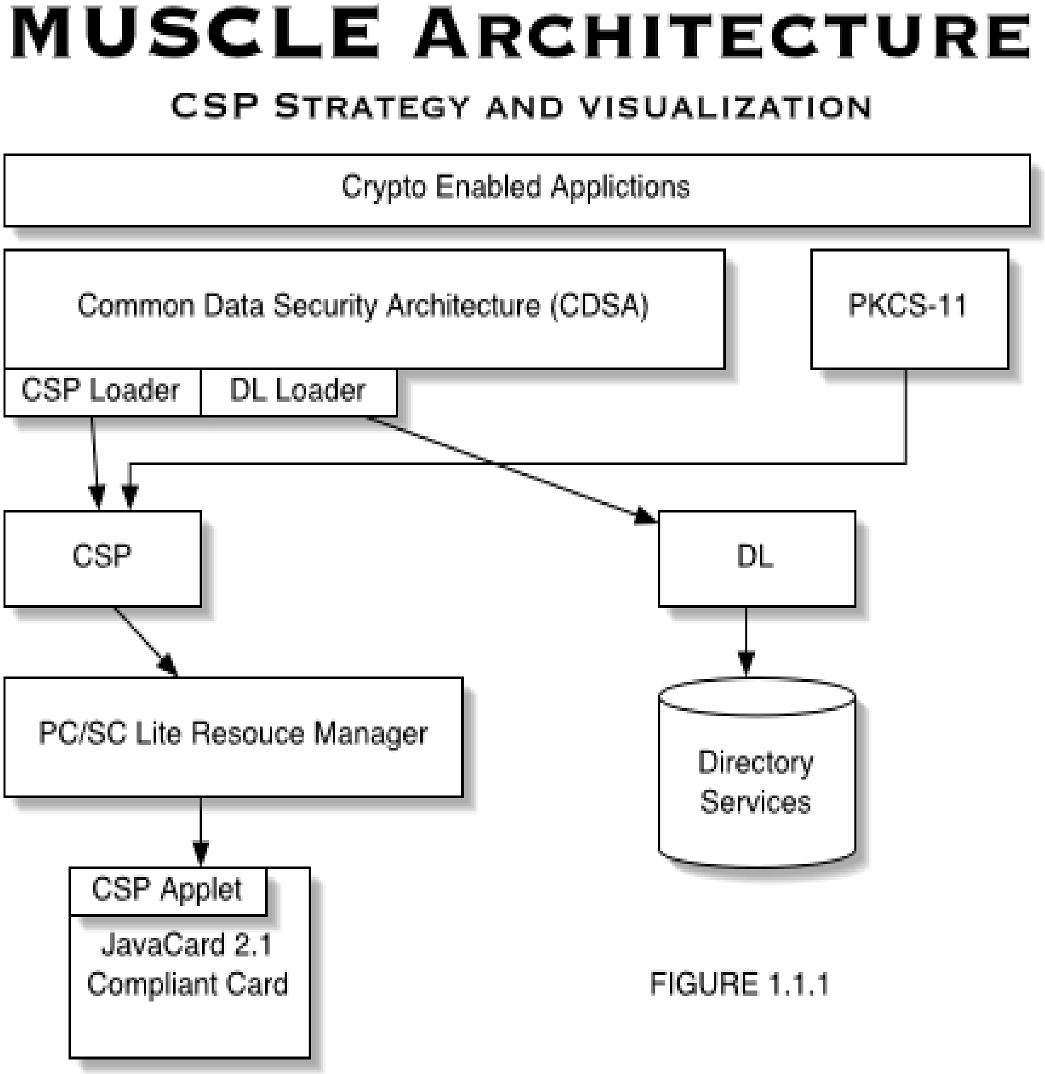

MUSCLE Architecture
CSP Strategy and Visualization
Crypto Enabled Applictions
Common Data Security Architecture (CDSA)
PKCS-11
CSP Loader
DL Loader
CSP
DL
PC/SC Lite Resouce Manager
Directory Services
CSP Applet
JavaCard 2.1 Compliant Card
FIGURE 1.1.1

Figure 1.1.1 describes how this applet can communicate with other security and cryptographic components in the larger schema.

Section 1. Context and conventions

1.1. Introduction

The Applet is capable of generating cryptographic keys on the card, and allows external keys to be inserted onto the card. These keys can be used in cryptographic operations, after proper user (or host application) authentication.

The Applet is capable of handling generic *objects*. An object is a sequence of bytes whose meaning is determined by the application. The Applet allows a host application to read and/or modify objects' contents, after proper user (or host application) authentication.

An object is identified by means of a 4-byte *object identifier*. Any object ID is available from `0x00000000` to `0xFFFFFF00`. Other object IDs are reserved. IDs `0xFFFFFFFE` and `0xFFFFFFFF` are reserved, respectively, as import and export buffers for transporting data to and from the card when it does not fit into a single `APDU`. The use of these special objects allows large keys and cryptogram to be exchanged and alleviates the problem of 255-byte maximum transfer size. For security reasons the Applet must delete these objects as soon as possible.

1.2. Security model

An *identity number* refers to one of 16 mechanisms (at maximum) by which the card can authenticate external applications running on the host. Each mechanism can be:

- based on a PIN verification: identity numbers from 0 to 7 (PIN-identities) that are associated to PIN numbers from 0 to 7
- based on a challenge/response cryptographic protocol: identity numbers from 8 to 13 (strong identities) that are associated to key numbers from 0 to 5
- reserved for alternative authentication[2] schemes: identity numbers 14 and 15

After an authentication mechanism has been run successfully, the corresponding identity is said to be "logged in". Each identity is associated a counter for the maximum number of times an authentication mechanism can be run unsuccessfully for that identity. On a successful authentication the counter is reset. On an unsuccessful authentication the counter is decreased and, if it goes to zero, the corresponding identity is blocked and can not be logged in anymore. PIN codes have an unblock mechanism[3].

A PIN-identity login requires a PIN code verification. The PIN number is the same as the identity number. Strong identities involve use of cryptographic keys. Strong identity n.8 requires use of key n.0, identity n.9 requires key n.1, and so on up to identity n.13. Login mechanisms for identities 14 and 15 are not specified in this release of the Card Edge specifications.

Each key or object on the card is associated with an *Access Control List* (*ACL*) that establishes which identities are required to be logged in to perform certain operations. The security model is designed in such a way to allow at least four levels of protection for card services:

- *no protection*: the operation is always allowed; in such a case the ACL requires only the anonymous identity to be logged in for the operation

[2] Such as biometric recognition
[3] See CreatePIN and UnblockPIN commands for details.

- *PIN protection*: the operation is allowed after a PIN verification; in such a case the ACL requires a PIN-based identity to be logged in for the operation
- *strong protection*: the operation is allowed only after a cryptography based, strong authentication of the host application (and optionally a PIN based authentication of the user); in such a case the ACL requires a strong identity to be logged in for the operation (and optionally a PIN based one)
- *full protection* (operation disabled): the operation is never allowed.

The use of a private key on the smartcard is usually PIN protected, but some applications could require a strong protection. Reading of a private key is usually disabled. Public objects may be always readable, but their modification could be PIN protected. Private objects could require PIN protection for reading and protection with another PIN or strong protection for writing.

1.3. ACL for objects

Object related operations are:

- creation
- read object
- write object
- deletion

Only read, write, and delete are regulated on a per object basis. An object creation is always allowed after pin #0 verification, if the object does not already exist. Every object is associated with an ACL of three bytes, where each byte corresponds to reading, writing and deletion permissions, respectively:

```
ObjectACL:
      Short       Read Permissions;
      Short       Write Permissions;
      Short       Delete Permissions;
```

A permission 2-bytes word has the following format:

```
Bit 16 (M.S. Bit) Identity #15 required (reserved identity)
Bit 15            Identity #14 required (reserved identity)
Bit 14            Identity #13 required (strong identity)
      ...
Bit 9             Identity #8 required (strong identity)
Bit 8             Identity #7 required (PIN identity)
      ...
Bit 2             Identity #1 required (PIN identity)
Bit 1 (L.S. Bit)  Identity #0 required (PIN identity)
```

If no bit is set on a permission word, then no authentication is required for the operation. If one or more bits are set, but not all, then all identities corresponding to set bits must be logged in to perform the operation. The special value `0xFFFF` (all bits set) disables the operation at all. Possibilities are clarified in the following examples:

Hex Value	**Meaning**
0x0000	Operation *always* allowed
0x0004	Identity n.2 (PIN) required
0x0101	*Both* Identity n.0 (PIN) *and* identity n.8 (strong) required
0xFFFF	Operation *never* allowed

1.4. ACL for keys

Operations involving cryptographic keys are:

- creation (injection or on-board generation)
- read key
- write key
- computation (encrypt, decrypt, sign, verify)

Only read, write, and computation are regulated on a per key basis. A key creation is always allowed after pin #0 verification, if the key does not exist yet. Every key is associated with an ACL of three 2-bytes words, where each word corresponds to reading, writing and using permissions, respectively:

```
KeyACL:
      Short       Read Permissions;
      Short       Write Permissions;
      Short       Use Permissions;
```

A permission word has the following format:

```
Bit 16 (M.S. Bit) Identity #15 required (reserved identity)
Bit 15            Identity #14 required (reserved identity)
Bit 14            Identity #13 required (strong identity)
      ...
Bit 9             Identity #8 required (strong identity)
Bit 8             Identity #7 required (PIN identity)
      ...
Bit 2             Identity #1 required (PIN identity)
Bit 1 (L.S. Bit)  Identity #0 required (PIN identity)
```

If no bit is set on a permission word, then *no authentication* is required for the operation. If one or more bits are set, but not all, then *all identities* corresponding to set bits must be logged in to perform the operation. The special value `0xFFFF` (all bits set) *disables* the operation *at all*[4]. See Object ACL description for some examples.

Note that a key write operation overwrites the associated ACL, too.

[4] Note that, when overwriting a key contents (if allowed to), the host application can also change the key ACL.

Section 2. Functional declarations

This section describes which functions, values, parameters, and behavior are defined in this document. Return codes for functions can be found at the end of this document.

2.1. Basic data types' encoding

A *byte* is an unsigned integer number, ranging from 0 to 255. Inside APDUs a byte is encoded with a byte.

A *short* is an unsigned integer number, ranging from 0 to 65535. Inside APDUs a short is always encoded as a 2 consecutive bytes, in 2-complement, most significant byte first.

A *big number* is an unsigned integer number with a variable encoding size. A big number is always encoded as follows:

- a short encoding the number's total size (in bytes)
- the big number value's bytes, most significant byte first

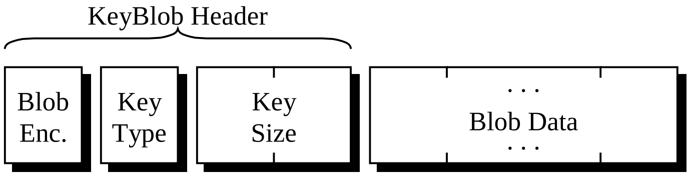
A *key number* uniquely identifies a cryptographic key inside the Cardlet. Key numbers are in the range from 0 to 15 and are always encoded as a single byte. Two cryptographic keys can be the public and private keys of a key pair. It is up to the host application to know and correctly handle such situations (see `InjectKey` and `GenerateKey` commands for further details).

2.2. Key blobs

A *key blob* is a sequence of bytes encoding a cryptographic key or key pair for import/export purposes. Whenever a key or key pair is transferred to the card, the application first transfers the corresponding key blob into the input temporary object then invokes the `ImportKey` command referencing it. Conversely, on a key or key pair export operation, the application first invokes an `ExportKey` operation, then retrieves the key blob from the output temporary object.

A key blob has the following format:

```
KeyBlob:
      Byte        Blob Encoding;
      Byte        Key Type;
      Short       Key Size;   // In bits
      Byte[]      Blob Data;
```



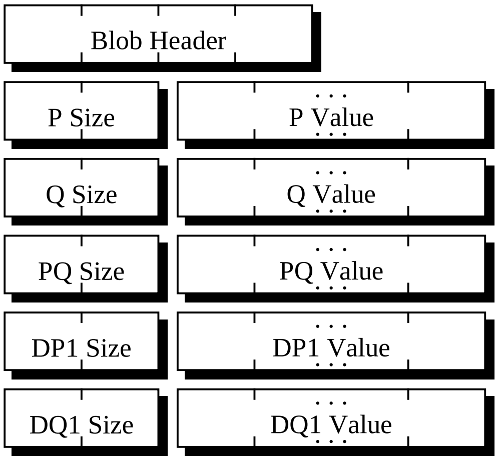
```
Values for Blob Encoding:
      0x00  BLOB_ENC_PLAIN;
      0x01  BLOB_ENC_ENCRYPTED (RFU)

Values for Key Type:
      RSA_PUBLIC        0x01  Public RSA key
      RSA_PRIVATE       0x02  Private RSA key
      RSA_PRIVATE_CRT   0x03  Private RSA CRT key
      DSA_PUBLIC        0x04  Public DSA key
      DSA_PRIVATE       0x05  Private DSA key
      DES               0x06  Standard DES key
      TRIPLE_DES        0x07  Standard Triple DES key
      TRIPLE_DES_3KEY   0x08  Standard 3 key Triple DES key

Allowed Values for Key Size:
```

RSA	512, 768, 1024, 2048 …
DSA	512, 768, 1024, 2048 …
DES	64
3DES	128
3DES3	192

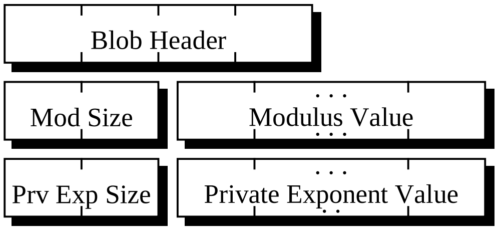
RSA KeyBlob Definitions

In the following Key Blob definitions, names of key components follow the same conventions as specified in JavaCard 2.1.1 API.

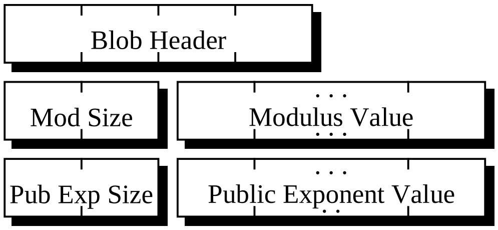
Key Type RSA_PRIVATE_CRT



Key Type RSA_PRIVATE



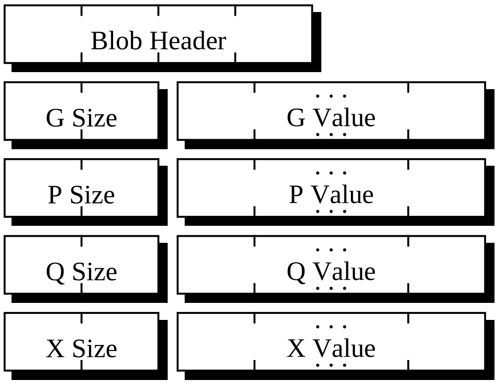
Key Type RSA_PUBLIC



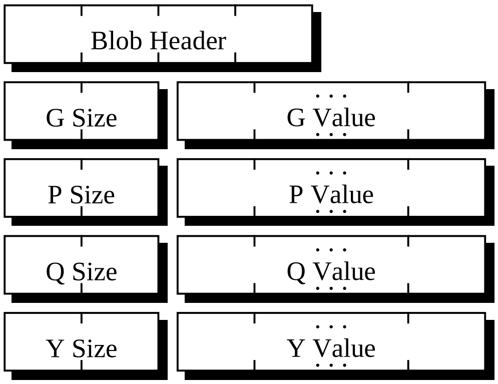
DSA KeyBlob Definitions

In the following Key Blob definitions, names of key components follow the same conventions as specified in JavaCard 2.1.1 API.

Key Type DSA_PRIVATE

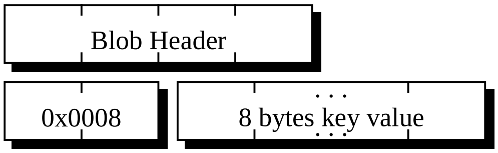


Key Type DSA_PUBLIC

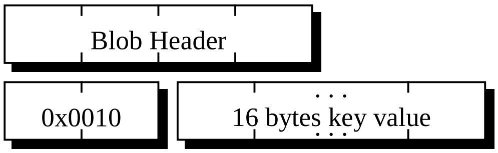


DES KeyBlob Definitions

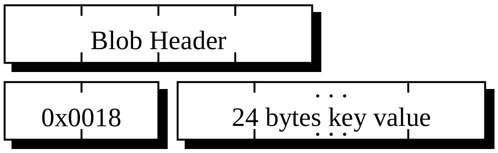
Key Type DES



Key Type TRIPLE_DES



Key Type TRIPLE_DES_3KEY



2.3. Summary of commands

Command Name	**S/R**	**INS (hex)**	**P1**	**P2**	**P3**	**DATA**
Key handling commands						
GenerateKeyPair	S	30	Prv Key N.	Pub Key N.	Size	Gen Params
ImportKey	S	32	Key N.	0x00	Size	Import Params
ExportKey	S	34	Key N.	0x00	Size	Export Params
ComputeCrypt	S	36	Key N.	Operation	Size	Ext Data
ExtAuthenticate	S	38	Key N.	0x00	Size	Ext Data
ListKeys	R	3A	Seq Option	0x00	0x0B	-
PIN related commands						
CreatePIN	S	40	PIN N.	Max Attempts	Size	PIN Params
VerifyPIN	S	42	PIN N.	0x00	Size	PIN Code
ChangePIN	S	44	PIN N.	0x00	Size	Params
UnblockPIN	R	46	PIN N.	0x00	Size	Unblock Code
ListPINs	R	48	0x00	0x00	0x02	-
Object related commands						
CreateObject	S	5A	0x00	0x00	0x0E	Create Params
DeleteObject	S	52	0x00	Zero Flag	0x04	Object ID
WriteObject	S	54	0x00	0x00	Size	Params
ReadObject	S/R	56	0x00	0x00	Size	Params
ListObjects	R	58	Seq Option	0x00	0x0E	-
Other						
LogOutAll	S	60	0x00	0x00	0x02	0x0000
GetChallenge	S	62	0x00	Output Data Location	Size	Chall. Params

GetStatus	R	3C	0x00	0x00	0x10	-
ISOVerify	S	20	0x00	PIN N.	Size	PIN Code
ISOGetResponse	R	C0	0x00	0x00	Expected Size	-

The `S/R` column is to be interpreted as follows:

- “`S`”: the command only sends data to the card with the `APDU`; the `P3` parameter specifies the amount of sent data
- “`R`”: the command only expects data to be returned from the card with the response `APDU`; the `P3` parameter specifies the maximum amount of expected data
- “`S/R`”: the command sends data to the card with the `APDU` and expects a response to be retrieved with an `ISO GET_RESPONSE` command; the `P3` parameter specifies the amount of sent data

2.4. General return codes

The following table shows all the possible status words returned from the Applet commands, along with a symbolic name and a short description. More specific information about the meaning of error codes is listed on individual function description pages.

MSC Return Codes (Status Words)

Value	*Symbolic Name*	*Description*
`90 00`	`SW_SUCCESS (ISO)`	Operation successfully completed
`9C 01`	`SW_NO_MEMORY_LEFT`	Insufficient memory onto the card to complete the operation
`9C 02`	`SW_AUTH_FAILED`	Unsuccessful authentication. Multiple consecutive failures cause the identity to block
`9C 03`	`SW_OPERATION_NOT_ALLOWED`	Operation not allowed because of the internal state of the Applet

MSC Return Codes (Status Words)

		internal state of the Applet
9C 05	SW_UNSUPPORTED_FEATURE	The requested feature is not supported either by the card or by the Applet
9C 06	SW_UNAUTHORIZED	Logged in identities don’t have enough privileges for the requested operation
9C 07	SW_OBJECT_NOT_FOUND	An object either explicitly or implicitly involved in the operation was not found
9C 08	SW_OBJ_EXISTS	Object already exists
9C 09	SW_INCORRECT_ALG	Input data to the command contained an invalid algorithm
9C 0B	SW_SIGNATURE_INVALID	The signature provided in a verify operation was incorrect
9C 0C	SW_IDENTITY_BLOCKED	Authentication operation not allowed because specified identity is blocked
9C 0D	SW_UNSPECIFIED_ERROR	An error occurred. No further information is given.
9C 0E	SW_INVALID_PARAMETER	Input data provided either in the APDU or by means of the input object is invalid
9C 10	SW_INCORRECT_P1	Incorrect P1 value
9C 11	SW_INCORRECT_P2	Incorrect P2 value
9C 12	SW_INCORRECT_LE	When receiving data from the card, expected length is not correct.
63 00	SW_INVALID_AUTH (ISO)	Unsuccessful authentication (for an ISO Verify). Multiple consecutive failures cause the PIN to block
69 83	SW_AUTH_BLOCKED (ISO)	The PIN referenced into an ISO Verify command is blocked
6A 86	SW_INCORRECT_P1P2 (ISO)	Incorrect values of either P1 or P2 parameter or both of them

MSC Return Codes (Status Words)		
		parameter or both of them
6D 00	SW_ERROR_INS (ISO)	Instruction code not recognized

2.5. APDU Reference

This section describes command APDUs to be exchanged between the card and the host computer. For each command we specify what parameters are to be provided as input and their format, and what parameters are to be expected as output and their format.

For each command we eventually specify error codes that the command can return in addition to the general ones listed in the previous paragraph.

2.5.1. MSCGenerateKeyPair

Function Parameters:

CLA	0xB0	
INS	0x30	
P1	Private Key Number	(0x00-0x0F)
P2	Public Key Number	(0x00-0x0F)
P3	Data Size	
DATA	Key Generation Parameters	

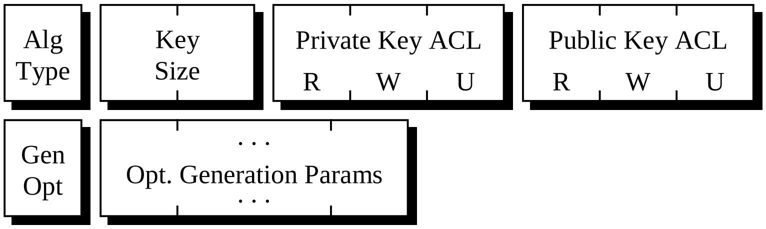
Definition:

This function generates a key or key pair using the card's on board key generation process. The key number (or numbers if a key pair is being generated), algorithm type, and algorithm parameters are specified by arguments P1 and P2 and by provided DATA. Appropriate values for these is specified below:

[DATA]

Key Generation Parameters:

Byte	Algorithm Type
Short	Key Size (in bits)
KeyACL	ACL for the private key
KeyACL	ACL for the public key
Byte	Key generation options
Byte[]	Optional generation parameters



```
Values for Algorithm Type:
    ALG_RSA       0x01    Private exponent in mod/exp format
    ALG_RSA_CRT   0x02    Private exponent in CRT format
    ALG_DSA       0x03
```

Allowed values for Key Generation Options:

```
    OPT_DEFAULT (0x00)
        No generation parameters provided.
    OPT_RSA_PUB_EXP (0x01)
        Use provided public exponent for RSA generation.
    OPT_DSA_SET_GPQ (0x02)
        Use provided G, P and Q parameters
```

```
Generation Parameters for RSA key, OPT_RSA_PUB_EXP option:
    Bignum   Public Exponent
```

For DSA key and OPT_DSA_SET_GPQ option, optional generation parameters are stored in the *import object*, in the following format:

```
    Bignum   G parameter;
    Bignum   P parameter;
    Bignum   Q parameter;
```

Notes

After a key pair generation, public key can be retrieved using the ExportKey command. It is supposed that an RSA public exponent is small enough to fit into a single APDU, so it is directly contained in the APDU itself.

If the specified key numbers are not in use, then the operation is allowed only if identity n.0 has already been verified.

If the specified key numbers are already in use, the operation overwrites actual key value(s) only if current logged in identities have sufficient privileges to write key contents, according to both private and public key ACLs. Furthermore key overwriting *could* be forbidden if new key parameters don't match in *type* and *size* old ones, or if a previous generation operation involved only one of specified keys. The exact behavior in these cases depends on the particular implementation and is out of the scope of this document.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_INCORRECT_P1`	Private key number is invalid
`SW_INCORRECT_P2`	Public key number is invalid
`SW_INCORRECT_ALG`	Key generation algorithm is incorrect
`SW_OBJECT_NOT_FOUND`	Import object, supposed to contain additional key generation parameters, was not found
`SW_OPERATION_NOT_ALLOWED`	Operation is not allowed due to the internal state of the Applet. This could be returned if trying to overwrite a key with different parameters but the Applet does not allow that.
`SW_UNAUTHORIZED`	One or both keys already exist and logged in identities don’t have sufficient privileges to overwrite them
`SW_DATA_INVALID`	Key generation parameters are incorrect

2.5.2. MSCImportKey

Function Parameters:

CLA	0xB0
INS	0x32
P1	Key Number (0x00 – 0x0F)
P2	0x00
P3	Import Parameters Length
DATA	Import Parameters

Definition:

This function allows the import of a key into the card by (over)-writing the Cardlet memory. Object ID `0xFFFFFFFE` needs to be initialized with a key blob before invocation of this function so that it can retrieve the key from this object. The exact key blob contents depend on the key’s algorithm, type and actual import parameters. The key's number, algorithm type, and parameters are specified by arguments P1, P2, P3, and DATA. Appropriate values for these is specified below:

```
[DATA]

Import Parameters:
    KeyACL       ACL for the imported key;
    Byte[]       Additional parameters;  // Optional
```

If KeyBlob’s Encoding is BLOB_ENC_PLAIN (0x00), there are no Additional Parameters.

Notes

If the specified key number is not in use, then the operation is allowed only if identity n.0 has already been verified.

If the specified key number is already in use, the operation overwrites actual key values only if current logged in identities have sufficient privileges to write key contents, according to the actual key ACLs. Furthermore key overwriting *could* be forbidden if new key parameters don't match in *type* and *size* old ones. The exact behavior in these cases depends on the particular implementation and is out of the scope of this document.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_INCORRECT_P2`	Key number is not valid
`SW_UNAUTHORIZED`	Specified key already exists and logged in identities don't have sufficient privileges to overwrite it
`SW_OBJECT_NOT_FOUND`	Import object was not found
`SW_OPERATION_NOT_ALLOWED`	Operation is not allowed due to the internal state of the Applet. This could be returned if trying to overwrite a key with different parameters but the Applet does not allow that.
`SW_DATA_INVALID`	Key blob is not valid.

2.5.3. MSCExportKey

Function Parameters:

CLA	0xB0
INS	0x34
P1	Key Number (0x00 – 0x0F)
P2	0x00
P3	Data Size
DATA	Export Parameters

Definition:

This function export a single key from the card by reading it from the Cardlet memory and creating a keyblob, according to the format defined in 2.2. The output data is placed in the export object (ID `0xFFFFFFFF`) to be read with one or more read object commands. Key number and export parameters are specified by arguments P1, P2, P3, and DATA. Appropriate values for these are specified below.

```
[DATA]
Export Parameters:
    Byte         Blob Encoding;
    Byte[]       Additional Parameters;
```

See Blob Format Specification for allowed values for Blob Encoding. If Blob Encoding is `BLOB_ENC_PLAIN` (`0x00`), then there are no Additional Parameters.

Note

The operation succeeds only if current logged identities have sufficient privileges to read key contents according to the key ACL.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_INCORRECT_P2`	Key number is not valid or specified key does not exist
`SW_UNAUTHORIZED`	Logged in identities don’t have sufficient privileges to read key contents
`SW_NO_MEMORY_LEFT`	There is not enough memory to create the export object.
`SW_DATA_INVALID`	Specified blob encoding or additional export parameters are incorrect

2.5.4. MSCComputeCrypt

Function Parameters:

CLA	0xB0
INS	0x36
P1	Key Number (0x00 – 0x0F)
P2	Operation
P3	Data Length
DATA	Extended Data

Definition:

This function performs the required operation on provided data, using a key on the card. It also allows proper initialization of the card cipher with custom data, if required by the application. Usually, this function is called 1 time for cipher initialization (`CIPHER_INIT`), 0 or more times for intermediate data processing (`CIPHER_UPDATE`) and 1 time for last data processing (`CIPHER_FINAL`).

Input and output data exchange can be arranged either directly in the command APDU itself or, for bigger data chunks, using the I/O objects.

When encrypting or decrypting, the command outputs processed data both on `UPDATE` and on `FINAL` operations. When signing the command outputs processed data (the signature) only on the `FINAL` operation. When verifying there is never processed data output and result is returned using the status word `SW1, SW2`. The `FINAL` verify command must provide both last data chunk and the signature to be verified.

Appropriate values for input parameters is specified below:

`Value of Operation:`

0x01	CIPHER_INIT	Initialize Cipher
0x02	CIPHER_PROCESS	Process more data

```
    0x03     CIPHER_FINAL     Process last data chunk

Extended data when Operation is CIPHER_INIT:
    Byte     cipher_mode;
    Byte     cipher_direction;
    Byte     data_location;

Values for Cipher Mode:
    RSA or RSA_CRT key:
        0x01     RSA_NO_PAD (No padding)
        0x02     RSA_PAD_PKCS1


    DSA key:
        0x10     DSA_SHA


    DES, 3DES or 3DES3 key:
        0x20     DES_CBC_NOPAD
        0x21     DES_ECB_NOPAD

Values for Cipher Direction:
    0x01     DIR_SIGN         Sign data
    0x02     DIR_VERIFY       Verify data
    0x03     DIR_ENCRYPT      Encrypt data
    0x04     DIR_DECRYPT      Decrypt data

Values for Data Location:
    0x01     DL_APDU      Initialization data in APDU;
    0x02     DL_OBJECT    Initialization data in input object;
```

`Initialization data` is a `DataChunk` (as defined below) and it either follows in the `APDU` (if `Data Location` is `DL_APDU`) or is contained in the input object with ID `0xFFFFFFFE` (if `Data Location` is `DL_OBJECT`). In order to provide no initialization data the application must supply a `DataChunk` with the `Size` field set to 0.

```
Extended Data when Operation is CIPHER_PROCESS
    Byte          Data Location
    DataChunk     Input Data        // If Location == APDU

Values for Data Location:
    0x01     DL_APDU     Input data contained in APDU;
                         Out data (if any) is returned in APDU
    0x02     DL_OBJECT   Input data in object 0xFFFFFFFE;
                         Out data (if any) in object 0xFFFFFFFF

Extended Data when Operation is CIPHER_FINAL and direction is not
DIR_SIGN:
    Byte          Data Location
    DataChunk     Input Data        // If Location == APDU
```

When operation is `CIPHER_FINAL` and direction is `DIR_SIGN`, last data chunk must be followed by the signature data to be verified.

```
Extended Data when Operation is CIPHER_FINAL and direction is not
DIR_SIGN:
    Byte          Data Location
    DataChunk     Input Data        // If Location == APDU
    DataChunk     Signature Data    // If Location == APDU
```

Data must be provided and is returned in the following format:

```
DataChunk:
    Short         Size;
    Byte[]        Data;   // exactly Size bytes of data;
```

Returns

If processed data must be returned to the host application, it is either placed into an APDU or into the export object (with ID `0xFFFFFFFF`), in the format defined above as `DataChunk`:

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. These are to be considered in addition to the general ones in 0.

Symbolic Name	*Description*
`SW_INCORRECT_P1`	Key number is not valid or specified key does not exist
`SW_INCORRECT_P2`	Specified operation is not valid
`SW_UNAUTHORIZED`	Logged in identities don't have sufficient privileges to use the key
`SW_NO_MEMORY_LEFT`	There is not enough memory to complete the operation
`SW_DATA_INVALID`	Data supplied either in the APDU itself, or in the input object, is not valid.
`SW_SIGNATURE_INVALID`	Signature verify operation failed

2.5.5. MSCExtAuthenticate

Function Parameters:

CLA	0xB0
INS	0x38
P1	Key Number (0x00 – 0x05)
P2	0x00
P3	Data Length
DATA	Extended Data

Definition:

This function performs the last step in a challenge/response cryptographic protocol that allows strong authentication of the host application to the card. A call to this function must occur after a call to MSCGetChallenge. The host should encrypt the challenge with the appropriate key and pass the encrypted data to this function.

In DECRYPT mode, this function uses the key specified in P1 to decrypt the challenge and then compares obtained data with original random data generated by MSCGetChallenge. In VERIFY mode, this function uses the public key specified in P1 to verify that the provided Encrypted Data is a valid digital signature for the original random data generated by MSCGetChallenge.

An exact match grants host authentication and the strong identity corresponding to the key number[5] is logged in. The try counter for the key is also reset.

A bad match result in decreasing the try counter for the key and, if it goes to zero, the key is blocked.

Appropriate values for input parameters are specified below:

[5] Refer to section 1.2 for details

```
Extended Data when Cipher Direction is DIR_DECRYPT:
     Byte          Cipher Mode
     Byte          Cipher Direction
     Byte          Data Location
     DataChunk     Input Data        // If Location == APDU

Extended Data when Cipher Direction is DIR_VERIFY:
     Byte          Cipher Mode
     Byte          Cipher Direction
     Byte          Data Location
     DataChunk     Input Data        // If Location == APDU
     DataChunk     Signature Data    // If Location == APDU

Values for Data Location:
     0x01     DL_APDU      Input data contained in APDU;
     0x02     DL_OBJECT    Input data in input object 0xFFFFFFFE;
```

For `Cipher Mode` and `Cipher Direction`, refer to the `ComputeCrypt` command.

Notes

- With RSA keys `RSA_NO_PAD` mode is not allowed for external authentication.
- Only `VERIFY` and `DECRYPT` directions are allowed for external authentication.
- Referenced key must be either a symmetric key or a public asymmetric one.
- Only keys with numbers from 0 to 5 can be used for external authentication.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
SW_INCORRECT_P1	Key number is not valid or specified key does not exist
SW_UNAUTHORIZED	Logged in identities don’t have sufficient

	privileges to use the key
SW_NO_MEMORY_LEFT	There is not enough memory to complete the operation
SW_DATA_INVALID	Data supplied either in the APDU itself, or in the input object, is not valid.
SW_IDENTITY_BLOCKED	Authentication was not possible because specified identity is blocked.
SW_AUTH_FAILED	Authentication failed. Multiple failures of this type cause the identity to block.
SW_OBJECT_NOT_FOUND	Specified operation requires input data from the input object, but it does not exist.

2.5.6. MSCListKeys

Function Parameters:

CLA	0xB0
INS	0x3A
P1	Sequence Option
P2	0x00
P3	0x0B

DATA

Definition:

This function returns a list of current keys and their properties including id, type, size, partner, and access control. This function is initially called with the reset sequence set for sequence type. The function only returns one object id at a time and must be called in repetition until SW_SUCCESS is returned.

Values for Sequence Option:

0x00	Reset sequence and get first entry
0x01	Get next entry

Notes:

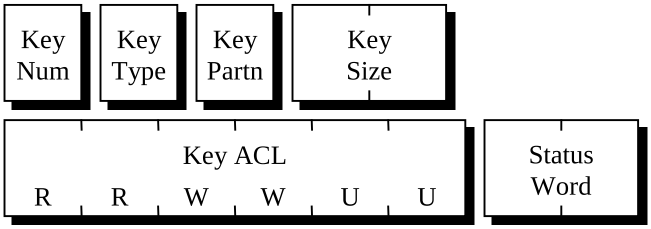
The data will be trailed with SW_SUCCESS. When the list has no more entries just SW_SUCCESS will be returned.

Reset sequence can be called at any time to reset the key pointer to the first in the list.

Returned data

Returned data if a key was found:

`Byte`	`Key Number`
`Byte`	`Key Type`
`Byte`	`Key Partner`
`Short`	`Key Size`
`KeyACL`	`ACL for this key`
`Short`	`Status Word`



If the key is part of a key pair and the other key is also stored on the card, the field `Key Partner` can contain the key number of the other key. This information is optional, and the special value `0xFF` means that it is not available.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_INCORRECT_P1`	Sequence option is not valid

2.5.7. MSCCreatePIN

Function Parameters:

CLA	0xB0
INS	0x40
P1	PIN Number
P2	PIN Maximum attempts
P3	Data Length
DATA	PIN creation parameters

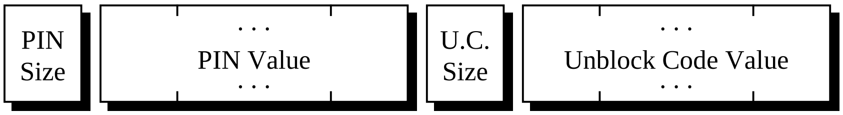
Definition:

This function creates a PIN with parameters specified by the P1, P2 and DATA values. P2 specifies the maximum number of consecutive unsuccessful verifications before the PIN blocks.

PIN Number	0x01-0x07

PIN creation parameters:

Byte	PIN Length
Byte[]	PIN value
Byte	Unblock code length
Byte[]	Unblock code value



Notes

Command succeeds and a new PIN code is initialized only if identity n.0 is logged in and specified PIN number is actually unused.

Right after a PIN creation command the new PIN identity is *not* logged in.

PIN number 0 cannot be created as it is reserved as a pre-defined PIN.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	Identity n.0 is not actually logged in
`SW_INCORRECT_P1`	Specified PIN number is invalid or is already in use
`SW_DATA_INVALID`	Provided PIN or unblock code data is not valid

2.5.8. MSCVerifyPIN

Function Parameters:

CLA	0xB0
INS	0x42
P1	PIN Number
P2	0x00
P3	Data Length
DATA	PIN Value

Definition:

This function verifies a PIN number sent by the DATA portion. The length of this PIN is specified by the value contained in P3.

Notes

Multiple consecutive unsuccessful PIN verifications will block the PIN. If a PIN blocks, then an UnblockPIN command can be issued.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	*Description*
SW_AUTH_FAILED	PIN verification failed. Multiple verification failures cause the PIN to block
SW_INCORRECT_P1	Specified PIN number is invalid or PIN code does not exist

SW_IDENTITY_BLOCKED	Specified PIN is actually blocked

2.5.9. MSCChangePIN

Function Parameters:

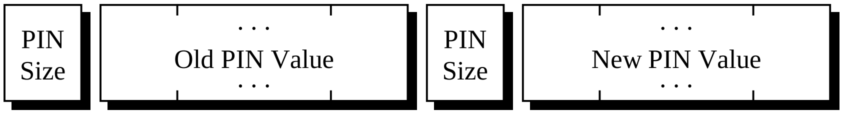
CLA	0xB0
INS	0x44
P1	PIN Number
P2	0x00
P3	Data Length
DATA	Pin Change Parameters

Definition:

This function changes a PIN code. The DATA portion contains both the old and the new PIN codes.

PIN creation parameters:

Byte	Old PIN length
Byte[]	Old PIN value
Byte	New PIN length
Byte[]	New PIN value



Notes

Right after a PIN change command, the corresponding PIN identity is *not* logged in.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_AUTH_FAILED`	PIN verification failed. Multiple verification failures cause the PIN to block
`SW_INCORRECT_P1`	Specified PIN number is invalid or PIN code does not exist
`SW_IDENTITY_BLOCKED`	Specified PIN is actually blocked and cannot be changed

2.5.10. MSCUnblockPIN

Function Parameters:

CLA	0xB0
INS	0x46
P1	PIN Number
P2	0x00
P3	Data Length
DATA	PIN Number Value

Definition:

This function unblocks a PIN number using the unblock code specified in the DATA portion. The P3 byte specifies the unblock code length.

Note:

After 3 multiple consecutive unsuccessful unblock tries, it is not possible to unblock the PIN neither to use it.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	*Description*
SW_AUTH_FAILED	Unblock code verification failed. Multiple verification failures cause the unblock code to block
SW_INCORRECT_P1	Specified PIN number is invalid or PIN code

	does not exist
SW_IDENTITY_BLOCKED	Specified unblocked code is actually blocked and cannot be changed anymore. If the associated PIN is also blocked, it will not be possible anymore to verify it or to issue any operation protected by it

2.5.11. MSCListPINs

Function Parameters:

```
CLA          0xB0
INS          0x48
P1           0x00
P2           0x00
P3           0x02
```

Definition:

This function returns a 2 byte bit mask of the available PINs that are currently in use. Each set bit corresponds to an active PIN, according to the following table. Least significant byte:

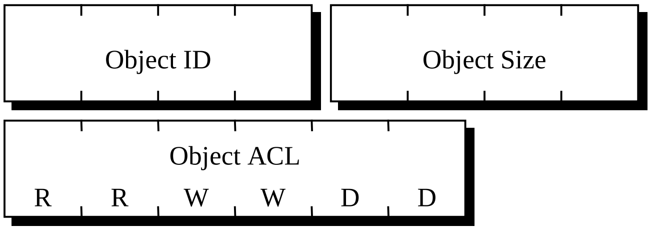
Bit	PIN Number	Bitmask Value
1	Pin #1	0x01
2	Pin #2	0x02
3	Pin #3	0x04
…	…	…

Most significant byte is RFU.

2.5.12. MSCCreateObject

Function Parameters:

```
CLA          0xB0
INS          0x5A
P1           0x00
P2           0x00
P3           0x0E

DATA         Object Parameters

[DATA]
Object Parameters
    Long         Object ID;
    Long         Object Size;
    ObjectACL    ObjectACL;
```



Definition:

This function creates an object that will be identified by the provided object ID. The object’s space and name will be allocated until deleted using MSCDeleteObject.

The object will be allocated upon the card's memory heap. For object lookup purposes, the Applet may allow up to a fixed amount of objects to reside on the card. The exact amount is beyond the scope of this document.

After creation, an object has “random” contents. Applications cannot rely on any particular contents right after an object creation.

Notes:

Object creation is only allowed if the object ID is available and logged in identity(-ies) have sufficient privileges to create objects.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	PIN number 0 has not been verified yet
`SW_OBJECT_EXISTS`	Specified object ID is already in use
`SW_NO_MEMORY_LEFT`	There is not enough free space on the card’s memory for the new object

2.5.13. MSCDeleteObject

Function Parameters:

```
CLA           0xB0
INS           0x52
P1            0x00
P2            Zero Flag
P3            0x04
DATA

[DATA]
Long          Object ID
```

Definition:

This function deletes the object identified by the provided object ID. The object's space and name will be removed from the heap and made available for other objects.

The zero flag denotes whether the object's memory should be zeroed after deletion. This kind of deletion is recommended if object was storing sensitive data.

Parameters:

```
Zero Flag
     0x01     Write zeros to object memory before release
     0x00     Memory zeroing not required
```

Notes

Object will be effectively deleted only if logged in identity(ies) have sufficient privileges for the operation, according to the object's ACL.

Not setting the zero flag doesn't guarantee future recovery of object data.

Return codes

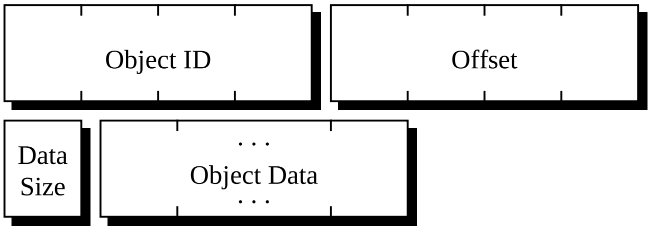
The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	Logged in identities don't have sufficient privileges to delete the specified object
`SW_OBJECT_NOT_FOUND`	Specified object does not exist

2.5.14. MSCWriteObject

Function Parameters:

```
CLA          0xB0
INS          0x54
P1           0x00
P2           0x00
P3           Data Size + 9

DATA         Parameters

[DATA]
Parameters:
     Long     Object ID
     Long     Offset
     Byte     Data Size
     Byte[]   Object Data
```



Definition:

This function (over-)writes data to an object that has been previously created with MSCCreateObject. Provided Object Data is stored starting from the byte specified by the Offset parameter. The size of provided object data must be exactly (Data Length – 8) bytes. Provided offset value plus the size of provided Object Data must not exceed object size.

Up to 246 bytes can be transferred with a single APDU. If more bytes need to be transferred, then multiple WriteObject commands must be used with different offsets.

Notes:

Object data will be effectively written only if logged in identity(ies) have sufficient privileges for the operation, according to the object's ACL.

Return codes

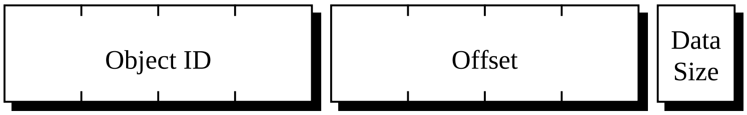
The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	Logged in identities don't have sufficient privileges to overwrite object's contents
`SW_OBJECT_NOT_FOUND`	Specified object does not exist

2.5.15. MSCReadObject

Function Parameters:

```
CLA          0xB0
INS          0x56
P1           0x00
P2           0x00
P3           0x09

DATA         Reading Parameters

[DATA]
Reading Parameters
    Long     Object ID
    Long     Offset
    Byte     Data Size
```



Definition:

This function reads data from an object that has been previously created with MSCCreateObject. Object data is read starting from the byte specified by the Offset parameter.

Up to 255 bytes can be transferred with a single APDU. If more bytes need to be transferred, then multiple ReadObject commands must be used with different offsets.

Notes

Object data will be effectively read only if logged in identity(ies) have sufficient privileges for the operation, according to the object's ACL.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	Logged in identities don't have sufficient privileges to read object's contents
`SW_OBJECT_NOT_FOUND`	Specified object does not exist

Returned data

```
[DATA]
    Byte[]        readData;
    Short         Status Word;
```

2.5.16. MSCListObjects

Function Parameters:

CLA	0xB0
INS	0x58
P1	Sequence Option
P2	0x00
P3	0x0E

DATA

Definition:

This function returns a list of current objects and their properties including id, size, and access control. This function must be initially called with the reset option. The function only returns one object information at a time and must be called in repetition until SW_SUCCESS is returned with no further data.

Applications cannot rely on any special ordering of the sequence of returned objects.

```
Values for Sequence Option:
    0x00     Reset sequence and get first entry
    0x01     Get next entry
```

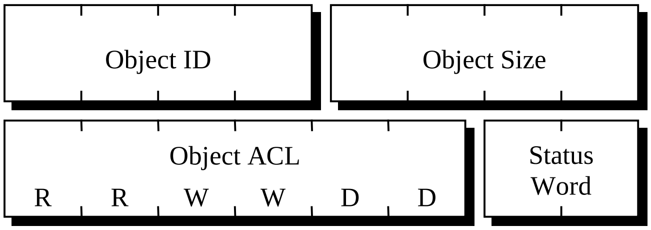
Notes:

The data will be trailed with SW_SUCCESS. When the list has no more entries just SW_SUCCESS will be returned and no data.

Reset sequence can be called at any time to reset the file pointer to the first in the list.

Returned data

Data returned if an object was found:



When the Reset Sequence option is selected, the first entry is returned.

If last object’s information was already retrieved, then no data and a status word of `SW_SUCCESS` are returned.

2.5.17. MSCLogoutAll

Function Parameters:

CLA	0xB0
INS	0x60
P1	0x00
P2	0x00
P3	0x02
DATA[0]	0x00
DATA[1]	0x00

Definition:

This function logs out any identity so that the card's state is in a non-authenticated state. DATA[0] and DATA[1] alleviates ambiguity by not using a ISO Case 1 transaction.

2.5.18. MSCGetChallenge

Function Parameters:

```
CLA           0xB0
INS           0x62
P1            0x00
P2            Output Data Location
P3            Data length

DATA
     Short    Random Data Size
     Short    Seed Length
     Byte[]   Seed Data       // Only if Seed Length > 0
```

Returns:

```
DATA          Random Number
```

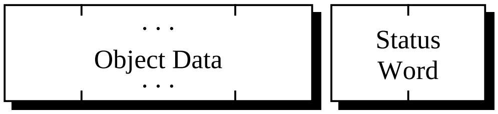
Definition:

This function return random data generated on the card with a length specified by the `Random Data Size` parameter. Data is either returned with a `GET_RESPONSE APDU` or is placed in the output object `0xFFFFFFFF`, according to the selected `Output Data Location`. After returned random data has been retrieved, an External Authenticate command should follow. This command also allows specifying optional input data to be fed into the random number generator. A zero value of the `Seed Length` parameter specifies no seeding data.

```
Values for Data Location:
    0x01     DL_APDU      Input data contained in APDU;
                          Out data (if any) is returned in APDU
    0x02     DL_OBJECT    Input data in object 0xFFFFFFFE;
                          Out data (if any) in object 0xFFFFFFFF
```

Returns

Returned data if `Data Location` is `DL_APDU`:



If `Data Location` is `DL_OBJECT` only the status word is returned.

2.5.19. MSCGetStatus

Function Parameters:

CLA	0x00
INS	0x3C
P1	0x00
P2	0x00
P3	Size of expected data

Definition:

This function retrieves general information about the Applet running on the smart card, and useful information about the status of current session, such as object memory information, currently used number of keys and PIN codes, currently logged in identities, etc…

Returns

Returned data has the following format:

Byte	Card Edge Major Version
Byte	Card Edge Minor Version
Byte	Software Major Version
Byte	Software Minor Version
Long	Total Object memory
Long	Free Object Memory
Byte	Number of used PINs
Byte	Number of used Keys
Short	Currently Logged in Identities

Card Edge Version reports the supported Card Edge command set version. Software Version reports the version of the Java Applet or other software running on the card that implements Card Edge command set. Currently Logged Identities is a word whose bits are to be interpreted according to the following table:

Bit 16 (M.S. Bit)	Reserved identity #2 currently logged in
Bit 15	Reserved identity #1 currently logged in
...	
Bit 10	Strong identity #1 currently logged in
Bit 9	Strong identity #0 currently logged in
Bit 8	PIN identity #7 currently logged in
...	
Bit 2	PIN identity #1 currently logged in
Bit 1 (L.S. Bit)	PIN identity #0 currently logged in

2.5.20. ISOVerify

Function Parameters:

CLA	0x00
INS	0x20
P1	0x00
P2	PIN Number
P3	Data Length
DATA	PIN Value

Definition:

This function verifies a PIN number sent by the DATA portion by a command supported by ISO 7816-4. The purpose of this command is to support readers with PIN pads which automatically send ISO Verify PIN commands to the card. The length of this PIN is specified by the value contained in P3.

Notes

Multiple consecutive unsuccessful PIN verifications will block the PIN. If a PIN blocks, then an UnblockPIN command can be issued.

Return codes

The following table shows how some error codes have to be interpreted when returned by this function. See section 2.4 for a list of all possible return codes.

Symbolic Name	*Description*
SW_AUTH_FAILED	PIN verification failed. Multiple verification failures cause the PIN to block

SW_INCORRECT_P1	Specified PIN number is invalid or PIN code does not exist
SW_IDENTITY_BLOCKED	Specified PIN is actually blocked

Section 3. Glossary

APDU	Application Protocol Data Unit
Applet	A Java application residing on a JavaCard compliant card
Applet Instance	An instance of a Java application residing on a JavaCard compliant card
Applet Selection	The process of selecting one of the Applet Instances residing onto a JavaCard compliant smartcard for processing further APDU commands.
Blocked PIN	A PIN whose verification has been unsuccessfully tried multiple consecutive times. Verification of a blocked PIN Code is not possible until unblocking.
External Authentication	A challenge-response cryptographic protocol by which an Applet Instance authenticates a host application.
Key Blob	A byte sequence encoding a cryptographic key
Key Number	A number from 0 to 7 that references a key on the Applet
Identity Number	A number from 0 to 15 referencing one of the 16 methods available to the host application to authenticate to an Applet Instance
Input Object	Object with ID 0xFFFFFFFE. It is used to store input data for commands that require large inputs.
Java Card ™	Java standard from Sun for Java enabled smart card interoperability. This document refers to the version 2.1.1 of the standard
Output Object	Object with ID 0xFFFFFFFF. It is used to store output data for commands that provide large outputs.
PIN Code (or PIN)	A byte sequence. Usually a PIN code is an ASCII character string. An Applet Instance can store multiple PIN codes and use them to authenticate a user
PIN Code Verification	The process by which an Applet Instance authenticates a host application comparing the host provided PIN Code with one of the on board stored ones.
PIN Number	A number from 0 to 7 that references a PIN code on the Applet
PIN Unblock Code	A code that, when entered successfully, unblocks a blocked PIN

Status Word (SW) A two byte code as defined in ISO-7816 as to the status of a smartcard command

T0/T1 Protocols Low level protocols used to communicate to a smartcard.