CoolKey Applet Interface

(Based strongly on “MUSCLE Cryptographic Card Edge Definition for Java\(^1\) Enabled SmartCards” by David Corcoran and Tommaso Cucinotta)

The original document was provided on an as-is basis. Neither the authors nor Red Hat 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\(^\circledR\) and Java Card\(^\circledR\) are trademarks of Sun Microsystems, Inc
Change Log:

[MUSCLE Cryptographic Card Edge Definition]
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

[CoolKey Applet Interface]
Version 1.0
March 3, 2004
Created AOLKey spec from original muscle spec, relyea@aol.net
Added RA Secure APDUs from “AOL NetKey Enrollment and Token Management Protocol”,
thayes0993@aol.com
Added Nelson’s AOLKey Object ID and Data format draft spec, misterSSL@aol.com

Version 1.0.1
April 27, 2004
Minor corrections found with document use. relyea@aol.net

Version 1.0.2
February 28, 2005
Convert to OpenOffice. Minor corrections, add new APDU definitions for KeyGen and Issuer information,
rrelyea@redhat.com

Version 1.02
March 30, 2005
Changed from AOLKey to CoolKey. Added support for PKCS 8.
rrelyea@rehat.com

Version 1.03
August 19, 2005
Added description of the application initialization string.
rrelyea@rehat.com
# Table of Contents

1 Context and conventions

1.1 Introduction.................................................................................................................. 5
1.2 Security model.................................................................................................................. 5
1.3 ACL for objects................................................................................................................. 6
1.4 ACL for keys.................................................................................................................... 6
1.5 Default ACL settings for CoolKey................................................................................... 7

2 Functional declarations

2.1 Basic data types’ encoding............................................................................................... 9
2.2 Key blobs........................................................................................................................ 9
2.3 Summary of commands.................................................................................................. 15
2.4 Authentication................................................................................................................ 16
2.5 General return codes...................................................................................................... 16
2.6 APDU Reference

2.6.1 SecureStartEnrollment................................................................................................. 18
2.6.2 SecureImportKeyEncrypted........................................................................................ 19
2.6.3 CKIImportKey............................................................................................................... 22
2.6.4 CKIComputeCrypt......................................................................................................... 25
2.6.5 CKIListKeys.................................................................................................................. 27
2.6.6 CKICreatePIN............................................................................................................... 30
2.6.7 CKIVerifyPIN............................................................................................................... 32
2.6.8 CKIChangePIN............................................................................................................. 34
2.6.9 CKIListPINs................................................................................................................. 36
2.6.10 SecureSetPIN............................................................................................................. 38
2.6.11 CKILogout................................................................................................................... 39
2.6.12 CKICreateObject....................................................................................................... 40
2.6.13 CKIDeleteObject........................................................................................................ 41
2.6.14 CKIWriteObject.......................................................................................................... 43
2.6.15 CKIReadObject.......................................................................................................... 45
2.6.16 SecureReadIOBuffer................................................................................................. 47
2.6.17 CKIListObjects......................................................................................................... 49
2.6.18 CKIGetStatus.............................................................................................................. 50
2.6.19 CKINoop.................................................................................................................... 52
2.6.20 CKIGetRandom.......................................................................................................... 53
2.6.21 CKIGetBuildID........................................................................................................... 54
2.6.22 CKIGetLifeCycle....................................................................................................... 55
2.6.23 CKISeedRandom........................................................................................................ 56
2.6.24 CKIGetIssuerInfo....................................................................................................... 57
2.6.25 SecureSetIssuerInfo................................................................................................. 58
2.6.26 CKIGetBuiltInACL..................................................................................................... 59
2.6.27 SecureSetLifeCycle................................................................................................. 60
2.6.28 SecureSetBuiltInACL............................................................................................... 61

3 CoolKey Object ID and Data Format............................................................................... 62

4 Glossary.......................................................................................................................... 63
Document Scope
The scope of this document is to provide a definition of command set to provide base cryptographic functionality through an abstract interface using Java enabled SmartCards and cryptographic tokens. This interface is restricted to applet supplied functions.

This specification was not written to encompass all the functionality of the Java Card platform but rather the subset of calls provided by the CoolKey applet. 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.
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 0xFFFFFFF00, Appendix A describes how the CoolKey infrastructure uses these IDs. 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 stores these buffers in volatile memory, which clears on applet deselect.

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 Secure Channel verification: identity numbers 14

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 can only be reset by secure channel requests.

A PIN-identity login requires a PIN code verification. The PIN number is the same as the identity number. Identity n.14 is only logged in during a secure channel operation.

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
- **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 during a secure channel operation.
• *full protection*: (operation disabled): the operation is never allowed.

The use of a private key on the SmartCard is usually PIN protected or not protected. Reading of a private key is disabled. Public objects will always be readable, but their modification could be PIN protected or strongly protected.

### 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. 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) RFU
- Bit 15 Identity #14 required (strong identity)
- Bit 14 RFU
- ...
- Bit 9 RFU
- 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 all bits are set on a permission word, then no authentication is required for the operation. If one or more bits are set, then at least one identity corresponding to set bits must be logged in to perform the operation. If no bits are set then the operation is disabled for all identities. Possibilities are clarified in the following examples:

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>Operation <em>never</em> allowed</td>
</tr>
<tr>
<td>0x0004</td>
<td>Identity n.2 (PIN) required</td>
</tr>
<tr>
<td>0x4001</td>
<td><em>Either</em> Identity n.0 (PIN) <em>or</em> identity n.14 (strong) required</td>
</tr>
<tr>
<td>0xFFFF</td>
<td>Operation <em>always</em> allowed</td>
</tr>
</tbody>
</table>
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) RFU
Bit 15 Identity #14 required (strong identity)
Bit 14 RFU
...
Bit 9 RFU
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 all bits are set on a permission word, then no authentication is required for the operation. If one or more bits are set, then at least one identity corresponding to set bits must be logged in to perform the operation. If no bits are set then the operation is disabled for all identities.² See Object ACL description for some examples.

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

1.5 Default ACL settings for CoolKey

² Note that, when overwriting a key’s contents (if allowed to), the host application can also change the key ACL.
In CoolKey deployments, only the RA holds the keys which enable secure channel connections. The ACLs are initialized as follows:

<table>
<thead>
<tr>
<th>ACL</th>
<th>Value</th>
<th>Set by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create object</td>
<td>0x4000 (RA)</td>
<td>Factory</td>
</tr>
<tr>
<td>Create key</td>
<td>0x4000 (RA)</td>
<td>Factory</td>
</tr>
<tr>
<td>Create pin</td>
<td>0x4000 (RA)</td>
<td>Factory</td>
</tr>
<tr>
<td>Private Key Read</td>
<td>0x0000 (No one)</td>
<td>Applet</td>
</tr>
<tr>
<td>Private Key Write</td>
<td>0x4000 (RA)</td>
<td>Applet</td>
</tr>
<tr>
<td>Private Key Use</td>
<td>0xffff or single bit between 0x0001 and 0x0080</td>
<td>Applet under direction of the RA</td>
</tr>
<tr>
<td>Public Key Read</td>
<td>0xffff (Any one)</td>
<td>Applet</td>
</tr>
<tr>
<td>Public Key Write</td>
<td>0x4000 (RA)</td>
<td>Applet</td>
</tr>
<tr>
<td>Public Key Use</td>
<td>0xffff (Any one)</td>
<td>Applet</td>
</tr>
<tr>
<td>Object Read</td>
<td>0xffff (Any one)</td>
<td>RA</td>
</tr>
<tr>
<td>Object Write</td>
<td>0x4000 (RA)</td>
<td>RA</td>
</tr>
<tr>
<td>Object Delete</td>
<td>0x4000 (RA)</td>
<td>RA</td>
</tr>
</tbody>
</table>

The RA is free to modify the ACLs on objects it creates.
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, most significant byte first.

A long is an unsigned integer number, ranging from 0 to 4,294,967,295. Inside APDUs a short is always encoded as a 2 consecutive bytes, 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 applet. 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 Error: Reference source not found and Error: Reference source not found 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_ENCPLAIN;
0x01 BLOB_ENCENCRYPTED (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
RSA_PKCS8_PAIR 0x09 Private and Public RSA key encoded in pkcs8

Allowed Values for Key Size:
RSA 512, 768, 1024, 2048 …
DSA 512, 768, 1024, 2048 …
DES 64
3DES 128
3DES3 192


### 2.2.1.1 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

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Size</td>
</tr>
<tr>
<td>Q</td>
<td>Size</td>
</tr>
<tr>
<td>PQ</td>
<td>Size</td>
</tr>
<tr>
<td>DP1</td>
<td>Size</td>
</tr>
<tr>
<td>DQ1</td>
<td>Size</td>
</tr>
</tbody>
</table>

#### Key Type RSA_PRIVATE

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mod</td>
<td>Size</td>
</tr>
<tr>
<td>Prv Exp</td>
<td>Size</td>
</tr>
</tbody>
</table>

---

11
2.2.1.2 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_PUBLIC**

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Size</td>
</tr>
<tr>
<td>P</td>
<td>Size</td>
</tr>
<tr>
<td>Q</td>
<td>Size</td>
</tr>
<tr>
<td>Y</td>
<td>Size</td>
</tr>
</tbody>
</table>

**2.2.1.3 DES KeyBlob Definitions**

**Key Type DES**

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>08</td>
</tr>
</tbody>
</table>

**Key Type TRIPLE_DES**

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>10</td>
</tr>
</tbody>
</table>
**Key Type TRIPLE DES 3KEY**

<table>
<thead>
<tr>
<th>Blob</th>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>24 byte key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### 2.3 Summary of commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Auth</th>
<th>S/R</th>
<th>CLA(hex)</th>
<th>INS(hex)</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Handling Commands</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartEnrollment</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>0C</td>
<td>User/PrvKey</td>
<td>Usage/PubKey</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>ImportKeyEncrypted</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>0A</td>
<td>User/PrvKey</td>
<td>Usage/Pubkey</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>ImportKey</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>32</td>
<td>Key N.</td>
<td>0x00</td>
<td>Size</td>
<td>Import Params</td>
</tr>
<tr>
<td>ComputeCrypt</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>36</td>
<td>Key N.</td>
<td>Operation</td>
<td>Size</td>
<td>Ext Data</td>
</tr>
<tr>
<td>ListKeys</td>
<td>N</td>
<td>S</td>
<td>B0</td>
<td>3A</td>
<td>Seq Option</td>
<td>0x00</td>
<td>0x0B</td>
<td>-</td>
</tr>
<tr>
<td><strong>PIN related commands</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreatePIN</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>40</td>
<td>PIN N.</td>
<td>Max Attempts</td>
<td>Size</td>
<td>New Pin</td>
</tr>
<tr>
<td>VerifyPIN</td>
<td>X</td>
<td>S/R</td>
<td>B0</td>
<td>42</td>
<td>PIN N.</td>
<td>0x00</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>ChangePIN</td>
<td>X</td>
<td>S</td>
<td>B0</td>
<td>44</td>
<td>PIN N.</td>
<td>0x00</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>ListPINs</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>48</td>
<td>0x00</td>
<td>0x00</td>
<td>0x02</td>
<td>-</td>
</tr>
<tr>
<td>SetPIN</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>04</td>
<td>PIN N.</td>
<td>0x00</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>Logout</td>
<td>P</td>
<td>S</td>
<td>B0</td>
<td>61</td>
<td>PIN N.</td>
<td>0x00</td>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td><strong>Object related commands</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateObject</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>5A</td>
<td>0x00</td>
<td>0x00</td>
<td>0x0E</td>
<td>Create Params</td>
</tr>
<tr>
<td>DeleteObject</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>53</td>
<td>0x00</td>
<td>Zero Flag</td>
<td>0x04</td>
<td>Object ID</td>
</tr>
<tr>
<td>WriteObject</td>
<td>A</td>
<td>S</td>
<td>B0/84</td>
<td>54</td>
<td>0x00</td>
<td>0x00</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>Read Object</td>
<td>A</td>
<td>S/R</td>
<td>B0/84</td>
<td>56</td>
<td>0x00</td>
<td>0x00</td>
<td>Size</td>
<td>Params</td>
</tr>
<tr>
<td>ReadIOBuffer</td>
<td>S</td>
<td>S/R</td>
<td>84</td>
<td>08</td>
<td>Data Len</td>
<td>0x00</td>
<td>0x02</td>
<td>Offset</td>
</tr>
<tr>
<td>ListObjects</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>58</td>
<td>Seq Option</td>
<td>0x00</td>
<td>0x0E</td>
<td>-</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetStatus</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>3C</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>Size</td>
</tr>
<tr>
<td>Noop</td>
<td>N</td>
<td>S</td>
<td>B0</td>
<td>71</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>GetBuildID</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>70</td>
<td>0x00</td>
<td>0x00</td>
<td>0x04</td>
<td>-</td>
</tr>
<tr>
<td>GetLifeCycle</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>F2</td>
<td>0x00</td>
<td>0x00</td>
<td>0x01</td>
<td>-</td>
</tr>
<tr>
<td>SetLifeCycle</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>F0</td>
<td>Life Cycle</td>
<td>0x00</td>
<td>0x00</td>
<td>-</td>
</tr>
<tr>
<td>SeedRandom</td>
<td>N</td>
<td>S</td>
<td>B0</td>
<td>73</td>
<td>0x00</td>
<td>0x00</td>
<td>Size</td>
<td>Data</td>
</tr>
<tr>
<td>GetRandom</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>72</td>
<td>0x00</td>
<td>0x00</td>
<td>Size</td>
<td>-</td>
</tr>
<tr>
<td>GetIssuerInfo</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>F6</td>
<td>0x00</td>
<td>0x00</td>
<td>0xe0</td>
<td>Data</td>
</tr>
<tr>
<td>SetIssuerInfo</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>F4</td>
<td>0x00</td>
<td>0x00</td>
<td>0xe0</td>
<td>Data</td>
</tr>
<tr>
<td>GetBuiltinACL</td>
<td>N</td>
<td>R</td>
<td>B0</td>
<td>FA</td>
<td>0x00</td>
<td>0x00</td>
<td>0x07</td>
<td>Data</td>
</tr>
<tr>
<td>SetBuiltinACL</td>
<td>S</td>
<td>S</td>
<td>84</td>
<td>F8</td>
<td>0x00</td>
<td>0x00</td>
<td>0x07</td>
<td>Data</td>
</tr>
<tr>
<td><strong>Secure Channel Setup</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InitializeUpdate</td>
<td>X</td>
<td>S/R</td>
<td>80</td>
<td>50</td>
<td>Key Set</td>
<td>Key Index</td>
<td>0x08</td>
<td>Params</td>
</tr>
<tr>
<td>ExternalAuthenticate</td>
<td>X</td>
<td>S</td>
<td>84</td>
<td>82</td>
<td>Sec Level</td>
<td>0x00</td>
<td>0x0a</td>
<td>Params</td>
</tr>
</tbody>
</table>

The Auth column is to be interpreted as follows:
- “A”: Either nonce or secure channel authentication required.
- “S”: Secure channel authentication required
- “P”: Nonce (PIN) authentication required
- “N”: No additional authentication needed for this command.
• “X”: This command is part of an authentication sequence, and special knowledge will be required to complete this command (either a PIN or a key).

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 Authentication
Commands that require authentication modify their APDUs before sending them to the applet. Secure channel authentication adds a MAC at then end of the APDU signed by the Card Manager Auth Key. These commands may also be encrypted by the Card Manager Encryption Key. In the CoolKey infrastructure only the RA holds these keys. These commands are all set using the secure class (0x84). This authentication is documented in the Java Global Platform spec.

CardEdge class commands (0xB0) that require authentication, modify their APDUs by appending an 8-byte nonce, which is returned from VerifyPIN. This nonce is valid until the card is reset, or the application calls Logout.

In both cases the P3 parameter is modified to include the size of the MAC or nonce.

Commands that are denoted ‘X’ above do not have an appended nonce or MAC, but have authentication information embedded in the APDU or the response (a PIN or a signed challenge).

2.5 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 00</td>
<td>SW_SUCCESS (ISO)</td>
<td>Operation successfully completed</td>
</tr>
<tr>
<td>9C 01</td>
<td>SW_NO_MEMORY_LEFT</td>
<td>Insufficient memory onto the card to complete the operation</td>
</tr>
<tr>
<td>9C 02</td>
<td>SW_AUTH_FAILED</td>
<td>Unsuccessful authentication. Multiple consecutive failures cause</td>
</tr>
<tr>
<td>Value</td>
<td>Symbolic Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>9C 03</td>
<td>SW_OPERATION_NOT_ALLOWED</td>
<td>Operation not allowed because of the internal state of the Applet</td>
</tr>
<tr>
<td>9C 05</td>
<td>SW_UNSUPPORTED_FEATURE</td>
<td>The requested feature is not supported either by the card or by the Applet</td>
</tr>
<tr>
<td>9C 06</td>
<td>SW_UNAUTHORIZED</td>
<td>Logged in identities don’t have enough privileges for the requested operation</td>
</tr>
<tr>
<td>9C 07</td>
<td>SW_OBJECT_NOT_FOUND</td>
<td>An object either explicitly or implicitly involved in the operation was not found</td>
</tr>
<tr>
<td>9C 08</td>
<td>SW_OBJ_EXISTS</td>
<td>Object already exists</td>
</tr>
<tr>
<td>9C 09</td>
<td>SW_INCORRECT_ALG</td>
<td>Input data to the command contained an invalid algorithm</td>
</tr>
<tr>
<td>9C 0B</td>
<td>SW_SIGNATURE_INVALID</td>
<td>The signature provided in a verify operation was incorrect</td>
</tr>
<tr>
<td>9C 0C</td>
<td>SW_IDENTITY_BLOCKED</td>
<td>Authentication operation not allowed because specified identity is blocked</td>
</tr>
<tr>
<td>9C 0D</td>
<td>SW_UNSPECIFIED_ERROR</td>
<td>An error occurred. No further information is given.</td>
</tr>
<tr>
<td>9C 0E</td>
<td>SW_INVALID_PARAMETER</td>
<td>Input data provided either in the APDU or by means of the input object is invalid</td>
</tr>
<tr>
<td>9C 10</td>
<td>SW_INCORRECT_P1</td>
<td>Incorrect P1 value</td>
</tr>
<tr>
<td>9C 11</td>
<td>SW_INCORRECT_P2</td>
<td>Incorrect P2 value</td>
</tr>
<tr>
<td>9C 12</td>
<td>SW_SEQUENCE_END</td>
<td>No more data in list.</td>
</tr>
<tr>
<td>63 00</td>
<td>SW_INVALID_AUTH (ISO)</td>
<td>Unsuccessful authentication (for an ISO Verify). Multiple consecutive failures cause the PIN to block</td>
</tr>
<tr>
<td>69 83</td>
<td>SW_AUTH_BLOCKED (ISO)</td>
<td>The PIN referenced into an ISO Verify command is blocked</td>
</tr>
<tr>
<td>6A 86</td>
<td>SW_INCORRECT_P1P2 (ISO)</td>
<td>Incorrect values of either P1 or P2 parameter or both of them</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Symbolic Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>6D 00</td>
<td>SW_ERROR_INS</td>
<td>(ISO) Instruction code not recognized</td>
</tr>
</tbody>
</table>

### 2.6 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.
2.6.1 SecureStartEnrollment

2.6.1.1 Function Parameters

CLA 0x84
INS 0x0C
P1 <User><Prv Key number>
P2 <Usage><Pub Key number>
P3 KeyGenParams Size

DATA KeyGenParams
KeyGenParams:
   Byte Algorithm Type (0x03 RSA Private CRT)
   Short Key Size (in bits)
   Byte Options (set if options are provided in the temporary buffer)
WrappedKey MacKey

WrappedKey:
   Byte Key Type - Mac (0x85)
   Byte Key Size - key size in bytes
   Byte[] Key Data - encrypted and padded to the correct 3DES block boundary
   Byte Key Check Size
   Byte[] Key Check Data

2.6.1.2 Required Authentication
Secure Channel

2.6.1.3 Definition
This APDU generates a new signing key of the requested type and key size. Additional parameters may be provided, such as PQG values for DSA or public exponent value for RSA keys. These additional parameters are provided in the temporary buffer (see Write Buffer). The public portion of the key is written into the temporary buffer along with the a Mac value that is computed using SHA-1, the Mac key from the key blob, and the key blob data for the public key.

P1 contains 2 parameters, the User in the upper 4 bits and the key index for the private key in the lower 4 bits. User specifies which PIN user should be granted use privilege of the generated private key, or 0xf if all users have use privilege for the private key.

P2 also contains 2 parameters, the Usage in the upper 4 bits and the key index for the public key in the lower. Valid Usage values are:
   0 default usage (Signing only for this APDU).
   1 signing only
   2 decryption only
   3 signing and decryption
Usage only specifies the usage for the private key.
The MAC key data is encrypted using the token KEK in 3DES-ECB mode. Once the data is decrypted it will contain data as defined by the following

MACKey:

| Short Size | Key Data |

The public part of the newly generated key is placed in the temporary buffer using the standard MUSCLE key blob format, where it may be read by using the Read Buffer APDU.

2.6.1.4 Muscle Key Blob Format (RSA Public Key)

The key generation operation places the newly generated key into the output buffer encoding in the standard Muscle key blob format. For an RSA key the data is as follows:

- Byte Encoding (0 for plaintext)
- Byte Key Type (1 for RSA public)
- Short Key Length (1024 - high byte first)
- Short Modulus Length
- Byte[] Modulus
- Short Exponent Length
- Byte[] Exponent

2.6.1.5 Signature Format (Proof)

The key generation operation creates a proof-of-location for the newly generated key. This proof is a signature computed with the new private key using the RSA-with-MD5 signature algorithm. The signature is computed over the Muscle Key Blob representation of the new public key and the challenge sent in the key generation request. These two data fields are concatenated together to form the input to the signature, without any other data or length fields.

- Byte[] Key Blob Data
- Byte[] Challenge

2.6.1.6 Key Generation Result

The key generation command puts the key blob and the signature (proof) into the output buffer using the following format:

- Short Length of the Key Blob
- Byte[] Key Blob Data
- Short Length of the Proof
- Byte[] Proof (Signature) Data

2.6.1.7 Notes

This APDU must be a part of a secure channel providing at least a MAC on the incoming requests.
2.6.1.8 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Security Requirement not satisfied</td>
</tr>
<tr>
<td>SW Xxx</td>
<td>Other errors due to decryption problems and format errors in the key that is being loaded or incorrect parameter values</td>
</tr>
</tbody>
</table>

2.6.1.9 Returned data

MacData:

```
Byte[] Calculated Mac value
```

2.6.1.10 Issues

Need to define errors for bad key formats and sizes.
2.6.2 SecureImportKeyEncrypted

2.6.2.1 Function Parameters

CLA  0x80
INS  0x0A
P1   <User><Prv Key number>
P2   <Usage><Pub Key number>
P3   Size of Data

DATA:
Long   ObjectID
WrappedKey   DesKey
Byte   IV_Length
Byte[]   IV_Data

WrappedKey:
   Byte   Key Type - DES3 ()
   Byte   Key Size - key size in bytes
   Byte[]  Key Data - encrypted and padded to the correct 3DES block boundary
   Byte   Key Check Size
   Byte[]  Key Check Data

2.6.2.2 Required Authentication

Secure Channel

2.6.2.3 Definition

This APDU loads the applet encryption key and it’s related public key from the data stored in Object ID.

P1 contains 2 parameters, the User in the upper 4 bits and the key index for the private key in the lower 4 bits. User specifies which PIN user should be granted use privilege of the generated private key, or 0xf if all users have use privilege for the private key.

P2 also contains 2 parameters, the Usage in the upper 4 bits and the key index for the public key in the lower. Valid Usage values are:
   0 default usage (decryption only for this APDU).
   1 signing only
   2 decryption only
   3 signing and decryption
Usage only specifies the usage for the private key.

Data should already have been loaded into the token temporary buffer using the write object. The data in the temporary buffer follows the MUSCLE blob format for keys as defined in the following paragraphs.
KeyBlob:
  Byte  Blob Encoding - must be 0x01 (encrypted)
  Byte  Key Type - RSA Private CRT (0x03) or RSA Private (0x02)
  Short Key Size - key size in bits
  Byte[] Key Data - encrypted and padded to the correct 3DES block boundary

The key data is encrypted using the DesKey and IV in 3DES-CBC mode. Once the data is decrypted it will contain data as defined by one of the following structures (depending on the key type).

RSAPrivate:
  Short Modulus Size
  Byte[] Modulus Data
  Short Private Exponent Size
  Byte[] Private Exponent Data

RSAPrivateCRT:
  Short P Size
  Byte[] P Data
  Short Q Size
  Byte[] Q Data
  Short PQ Size
  Byte[] PQ Data (PQ mod p)
  Short DP1 Size
  Byte[] DP1 Data
  Short DP2 Size
  Byte[] DP2 Data

RSAPKCS8Pair
  public and private key encoded with PKCS8.

The DesKey is wrapped with the KEK in 3DES-CBC mode.

2.6.2.4 Notes
This APDU must be a part of a secure channel providing at least a MAC on the incoming requests.

When importing Private Keys, the corresponding public key should already be imported using ImportKey, or the Private Key will not be usable. When importing RSAPKCS8Pair, the public key is automatically imported.

2.6.2.5 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Security Requirement not satisfied</td>
</tr>
<tr>
<td>SW_XXX</td>
<td>Other errors due to decryption problems and format errors in the key that is being loaded</td>
</tr>
</tbody>
</table>
2.6.2.6 Returned data
None

2.6.2.7 Issues
Need to define errors for bad key formats and sizes.
2.6.3 CKYImportKey

2.6.3.1 Function Parameters

CLA 0x84 or 0xB0
INS 0x32
P1 Key Number (0x00–0x0F)
P2 0x00
P3 Import Parameters Length
DATA Import Parameters

2.6.3.2 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.3.3 Definition

This function allows the import of a key into the card by (over)-writing the Cardlet memory. Object ID 0xFFFFFFFF 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 are specified below:

[DATA]
Import Parameters:
   Long   ObjectID
   KeyACL ACL for the imported key;
   KeyACL ACL for public key // only for RSA PKCS8 key pairs
   Byte[] Additional parameters; // Optional

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

2.6.3.4 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.

This function works identically to SecureImportKeyEncrypted, except the imported key is in plaintext.
2.6.3.5 Return codes

2.6.3.6
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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_INCORRECT_P2</td>
<td>Key number is not valid</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Specified key already exists and logged in identities don’t have sufficient privileges to overwrite it</td>
</tr>
<tr>
<td>SW_OBJECT_NOT_FOUND</td>
<td>Import object was not found</td>
</tr>
<tr>
<td>SW_OPERATION_NOT_ALLOWED</td>
<td>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.</td>
</tr>
<tr>
<td>SW_DATA_INVALID</td>
<td>Key blob is not valid.</td>
</tr>
</tbody>
</table>

2.6.3.7
2.6.4 CKYComputeCrypt

2.6.4.1 Function Parameters

2.6.4.2
CLA 0x84 or 0xB0
INS 0x36
P1 Key Number (0x00 - 0x0F)
P2 Operation
P3 Data Length
DATA Extended Data

2.6.4.3 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.4.4 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). Alternately for single fixed block operations, this function can be called with CIPHER_ONE_STEP, which is equivalent to calling CIPHER_INIT and CIPHER_FINAL in one APDU.

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 are specified below:

Value of Operation:
0x01 CIPHER_INIT Initialize Cipher
0x02 CIPHER_PROCESS Process more data
0x03 CIPHER_FINAL Process last data chunk
0x04 CIPHER_ONE_STEP Same as Initialize and Final in one step.

Extended data when Operation is CIPHER_INIT or CIPHER_ONE_STEP:
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 0xFFFFFFFF (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 0xFFFFFFFF;
      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_VERIFY, 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;
2.6.4.5 Notes
When doing signing with RSA keys, the applet verifies the signature against the public key before it returns the result. DSA keys are not supported.

2.6.4.6 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:

2.6.4.7 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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_INCORRECT_P1</td>
<td>Key number is not valid or specified key does not exist</td>
</tr>
<tr>
<td>SW_INCORRECT_P2</td>
<td>Specified operation is not valid</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Logged in identities don’t have sufficient privileges to use the key</td>
</tr>
<tr>
<td>SW_NO_MEMORY_LEFT</td>
<td>There is not enough memory to complete the operation</td>
</tr>
<tr>
<td>SW_DATA_INVALID</td>
<td>Data supplied either in the APDU itself, or in the input object, is not valid.</td>
</tr>
<tr>
<td>SW_SIGNATURE_INVALID</td>
<td>Signature verify operation failed</td>
</tr>
<tr>
<td>SW_INCORRECT_ALG</td>
<td>Algorithm does not match the key type</td>
</tr>
<tr>
<td>SW_OPERATION_NOT_ALLOWED</td>
<td>Operation not allowed for this key type or algorithm</td>
</tr>
</tbody>
</table>
2.6.5 CKYListKeys

2.6.5.1 Function Parameters

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

2.6.5.2 Required Authentication

None

2.6.5.3 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

2.6.5.4 Notes

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

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

2.6.5.5 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

<table>
<thead>
<tr>
<th>Key Num</th>
<th>Key Type</th>
<th>Key Partner</th>
<th>Key Size</th>
<th>KeyACL</th>
<th>Status Word</th>
</tr>
</thead>
</table>


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.

### 2.6.5.6 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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_INCORRECT_P1</td>
<td>Sequence option is not valid</td>
</tr>
<tr>
<td>SW_SEQUENCE_END</td>
<td>No more keys in the list</td>
</tr>
</tbody>
</table>
2.6.6 CKYCreatePIN

2.6.6.1 Function Parameters

2.6.6.2
CLA 0x84 or 0xB0
INS 0x40
P1 PIN Number
P2 PIN Maximum attempts
P3 Pin Length
Data
[Data]
   Byte[] Pin Value

2.6.6.3 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.6.4 Definition

2.6.6.5
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 0x00-0x07

2.6.6.6 Notes

2.6.6.7
Command succeeds and a new PIN code is initialized only if one of the identities specified by the create PIN ACL is authenticated and specified PIN number is actually unused.

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

2.6.6.8 Return codes

2.6.6.9
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.
<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Identity n.0 is not actually logged in</td>
</tr>
<tr>
<td>SW_INCORRECT_P1</td>
<td>Specified PIN number is invalid or is already in use</td>
</tr>
<tr>
<td>SW_DATA_INVALID</td>
<td>Provided PIN or unblock code data is not valid</td>
</tr>
</tbody>
</table>
2.6.7 CKYVerifyPIN

2.6.7.1 Function Parameters

2.6.7.2
CLA 0xB0
INS 0x42
P1 PIN Number
P2 0x00
P3 Data Length
DATA PIN Value

2.6.7.3 Required Authentication
Supplied PIN must be valid.

2.6.7.4 Definition

2.6.7.5
This function verifies a PIN number sent by the DATA portion. The length of this PIN is specified by the value contained in P3.

On success the applet returns an 8 byte nonce used for all PIN authenticated operations. This value is appended to all commands requiring nonce authenticated issued in CardEdge class (0xB0).

2.6.7.6 Notes

2.6.7.7
Multiple consecutive unsuccessful PIN verifications will block the PIN. If a PIN blocks, then a SetPIN command can be issued from the secure channel to reset it.

2.6.7.8 Return codes

2.6.7.9
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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_AUTH_FAILED</td>
<td>PIN verification failed. Multiple verification failures cause the PIN to block</td>
</tr>
<tr>
<td>SW_INCORRECT_P1</td>
<td>Specified PIN number is invalid or PIN code does not exist</td>
</tr>
</tbody>
</table>
2.6.7.10 Returns
Returned data if Pin was valid.
byte[] 8-byte nonce.
2.6.8 CKYChangePIN

2.6.8.1 Function Parameters

2.6.8.2
CLA 0xB0
INS 0x44
P1 PIN Number
P2 0x00
P3 Data Length
DATA Pin Change Parameters

2.6.8.3 Required Authentication
Supplied Pin must be valid.

2.6.8.4 Definition

2.6.8.5
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

<table>
<thead>
<tr>
<th>Pin Size</th>
<th>Old</th>
<th>...</th>
<th>Pin</th>
<th>...</th>
<th>Value</th>
<th>Pin Size</th>
<th>New</th>
<th>...</th>
<th>PIN</th>
<th>...</th>
<th>Value</th>
</tr>
</thead>
</table>

2.6.8.6 Notes

2.6.8.7
Right after a successful PIN change command, the corresponding PIN identity is not logged in for any application (all existing nonces are invalid).

2.6.8.8 Return codes

2.6.8.9
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.
<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_AUTH_FAILED</td>
<td>PIN verification failed. Multiple verification failures cause the PIN to block</td>
</tr>
<tr>
<td>SW_INCORRECT_P1</td>
<td>Specified PIN number is invalid or PIN code does not exist</td>
</tr>
<tr>
<td>SW_IDENTITY_BLOCKED</td>
<td>Specified PIN is actually blocked and cannot be changed</td>
</tr>
</tbody>
</table>
2.6.9 CKYListPINs

2.6.9.1 Function Parameters

2.6.9.2
CLA 0xB0
INS 0x48
P1 0x00
P2 0x00
P3 0x02

2.6.9.3 Required Authentication
None

2.6.9.4 Definition

2.6.9.5
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.

<table>
<thead>
<tr>
<th>Bit</th>
<th>PIN Number</th>
<th>Bitmask Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pin #0</td>
<td>0x01</td>
</tr>
<tr>
<td>2</td>
<td>Pin #1</td>
<td>0x02</td>
</tr>
<tr>
<td>3</td>
<td>Pin #2</td>
<td>0x04</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>Pin #7</td>
<td>0x80</td>
</tr>
</tbody>
</table>

Most significant byte is RFU.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>
2.6.10 SecureSetPIN

2.6.10.1 Function Parameters

<table>
<thead>
<tr>
<th>CLA</th>
<th>0x80</th>
</tr>
</thead>
<tbody>
<tr>
<td>INS</td>
<td>0x04</td>
</tr>
<tr>
<td>P1</td>
<td>Pin number</td>
</tr>
<tr>
<td>P2</td>
<td>0x00</td>
</tr>
<tr>
<td>P3</td>
<td>&lt;Pin length&gt;</td>
</tr>
</tbody>
</table>

DATA Pin

Pin: Byte[] New Pin value

2.6.10.2 Required Authentication

Secure Channel

2.6.10.3 Definition

This APDU sets or resets a PIN. It is used to allow the card management service to set the PIN to a value chosen by the user. The invalid PIN verification counter on the token is also reset to the initial value. The card management system verifies that the user is authorized to request this change.

2.6.10.4 Notes

This APDU must be a part of a secure channel providing at least a MAC on the incoming requests.

2.6.10.5 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Security Requirement not satisfied</td>
</tr>
</tbody>
</table>

2.6.10.6 Returned data

None

2.6.10.7 Issues

It may be more appropriate to change this command into a “Reset PIN” command, which simply removes the PIN from the applet. The client would then prompt the user for a new PIN, and use the normal ChangePIN (SetPIN) interface to the applet to establish the new PIN value.
2.6.11 CKYLogout

2.6.11.1 Function Parameters

CLA 0xB0
INS 0x61
P1 Pin number
P2 0x00
P3 0x00

2.6.11.2 Required Authentication

PIN nonce

2.6.11.3 Definition

This function logs out the given identity. Application should discard their nonces once Logout has been called. Other pins may still be valid.
2.6.12 CKYCreateObject

2.6.12.1 Function Parameters

2.6.12.2
CLA 0x84 or 0xB0
INS 0x5A
P1 0x00
P2 0x00
P3 0x0E
DATA Object Parameters

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

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Object Size</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>R</th>
<th>Object ACL</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
</table>

2.6.12.3 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.12.4 Definition

2.6.12.5
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 CKYDeleteObject. 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.
2.6.12.6 Notes

2.6.12.7
Object creation is only allowed if logged in identity(-ies) have sufficient privileges to create objects. PIN identities may not create objects if the object ID already exists.

2.6.12.8 Return codes

2.6.12.9
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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Appropriate Identity has not been verified yet</td>
</tr>
<tr>
<td>SW_OBJECT_EXISTS</td>
<td>Specified object ID is already in use</td>
</tr>
<tr>
<td>SW_NO_MEMORY_LEFT</td>
<td>There is not enough free space on the card’s memory for the new object</td>
</tr>
</tbody>
</table>
2.6.13 CKYDeleteObject

2.6.13.1 Function Parameters

CLA 0x84 or 0xB0
INS 0x52
P1 0x00
P2 Zero Flag
P3 0x04
DATA

[DATA]
Long Object ID

2.6.13.2 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.13.3 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.

2.6.13.4 Parameters

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

2.6.13.5 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.

2.6.13.6 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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Logged in identities don’t have sufficient privileges to delete the specified object</td>
</tr>
<tr>
<td>SW_OBJECT_NOT_FOUND</td>
<td>Specified object does not exist</td>
</tr>
</tbody>
</table>
2.6.14 CKYWriteObject

2.6.14.1 Function Parameters

2.6.14.2
CLA 0x84 or 0xB0
INS 0x54
P1 0x00
P2 0x00
P3 Data Size + 9

DATA Parameters

<table>
<thead>
<tr>
<th>[DATA]</th>
<th>Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long Object ID</td>
<td></td>
</tr>
<tr>
<td>Long Offset</td>
<td></td>
</tr>
<tr>
<td>Byte Data Size</td>
<td></td>
</tr>
<tr>
<td>Byte[] Object Data</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Offset</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Object Data</th>
</tr>
</thead>
</table>

2.6.14.3 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.14.4 Definition

2.6.14.5
This function (over-)writes data to an object that has been previously created with CKYCreateObject. 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 specified in create.

Up to 240 bytes can be transferred with a single APDU. If more bytes need to be transferred, then multiple CKYWriteObject commands must be used with different offsets.
2.6.14.6 Notes

2.6.14.7
Object data will be effectively written only if logged in identity(ies) have sufficient privileges for the operation, according to the object’s ACL.

2.6.14.8 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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Logged in identities don’t have sufficient privileges to overwrite object’s contents</td>
</tr>
<tr>
<td>SW_OBJECT_NOT_FOUND</td>
<td>Specified object does not exist</td>
</tr>
</tbody>
</table>
2.6.15 CKYReadObject

2.6.15.1 Function Parameters

2.6.15.2
CLA 0x84 or 0xB0
INS 0x56
P1 0x00
P2 0x00
P3 0x09
DATA Reading Parameters

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

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Offset</th>
<th>Data Size</th>
</tr>
</thead>
</table>

2.6.15.3 Required Authentication
Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.15.4 Definition

2.6.15.5
This function reads data from an object that has been previously created with
CKYCreateObject. 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.

2.6.15.6 Notes

2.6.15.7
Object data will be effectively read only if logged in identity(ies) have sufficient
privileges for the operation, according to the object’s ACL.
2.6.15.8 Return codes

2.6.15.9
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.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Logged in identities don’t have sufficient privileges to read object’s contents</td>
</tr>
<tr>
<td>SW_OBJECT_NOT_FOUND</td>
<td>Specified object does not exist</td>
</tr>
</tbody>
</table>

2.6.15.10 Returned data

[DATA]

    Byte[] readData;
    Short Status Word;
2.6.16 SecureReadIOBuffer

2.6.16.1 Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>0x80</td>
</tr>
<tr>
<td>INS</td>
<td>0x08</td>
</tr>
<tr>
<td>P1</td>
<td>&lt;Data length&gt;</td>
</tr>
<tr>
<td>P2</td>
<td>0x00</td>
</tr>
<tr>
<td>P3</td>
<td>0x02</td>
</tr>
</tbody>
</table>

**DATA Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Offset</td>
<td></td>
</tr>
</tbody>
</table>

2.6.16.2 Required Authentication

Secure Channel

2.6.16.3 Definition

This APDU reads data from a temporary buffer on the token. The temporary buffer allows a data longer than can be handled in one APDU to be used in a token operation. For example, an RSA 2048-bit public key is (at least) 256 bytes long, which is more than the maximum data size of 255 bytes.

2.6.16.4 Notes

This APDU must be a part of a secure channel providing at least a MAC on the incoming requests.

2.6.16.5 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_UNAUTHORIZED</td>
<td>Security Requirement not satisfied</td>
</tr>
<tr>
<td>SW_INVALID_PARAM</td>
<td>The offset and size are too large for the temporary buffer</td>
</tr>
</tbody>
</table>

2.6.16.6 Returned data

None

2.6.16.7 Issues

Should the data length field be moved to the P1 parameter location? The P3 value would then be 4.

This APDU is redundant now that ReadObject can be called on a secure channel. It should be deprecated.
2.6.17 CKYListObjects

2.6.17.1 Function Parameters

2.6.17.2
CLA 0xB0
INS 0x58
P1 Sequence Option
P2 0x00
P3 0x0E
DATA

2.6.17.3 Required Authentication
None.

2.6.17.4 Definition

2.6.17.5
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

2.6.17.6 Notes

2.6.17.7
The data will be trailed with SW_SUCCESS. When the list has no more entries just SW_SEQUENCE_END 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.

2.6.17.8 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_SEQUENCE_END</td>
<td>No more objects in the list</td>
</tr>
</tbody>
</table>
2.6.17.9 Returned data

2.6.17.10

Data returned if an object was found:

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Object Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

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_SEQUENCE_END is returned.
2.6.18 CKYGetStatus

2.6.18.1 Function Parameters

2.6.18.2
CLA 0xB0
INS 0x3C
P1 0x00
P2 0x00
P3 Size of expected data

2.6.18.3 Required Authentication
None

2.6.18.4 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.

2.6.18.5 Return code

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) RFU
Bit 15 Identity #14 required (strong identity)
Bit 14 RFU
...
Bit 9 RFU
Bit 8 Identity #7 required (PIN identity)
...
Bit 2 Identity #1 required (PIN identity)
Bit 1 Identity #1 required (LSB)
2.6.19 CKYNoop

2.6.19.1 Function Parameters

CLA 0xB0
INS 0x71
P1 0x00
P2 0x00
P3 0x00

2.6.19.2 Required Authentication

None

2.6.19.3 Definition

This function returns success if the applet is selected.

2.6.19.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>

2.6.19.5 Returned data

None
2.6.20 CKYGetRandom

2.6.20.1 Function Parameters
CLA 0xB0
INS 0x73
P1 0x00
P2 0x00
P3 0x04

2.6.20.2 Required Authentication
None

2.6.20.3 Definition
This function returns size random bytes from the on card random number generator.

2.6.20.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>

2.6.20.5 Returned data
Returned Data has the following format:

Byte[] randomData
2.6.21 CKYGetBuildID

2.6.21.1 Function Parameters

CLA 0xB0
INS 0x70
P1 0x00
P2 0x00
P3 size

2.6.21.2 Required Authentication

None

2.6.21.3 Definition

This function returns the unique applet build ID. Each applet build has its own unique build ID calculated at build time.

2.6.21.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>

2.6.21.5 Returned data

Returned Data has the following format:

- short Major Build ID
- short Minor Build ID
2.6.22 CKYGetLifeCycle

2.6.22.1 Function Parameters
CLA 0xB0
INS 0xF2
P1 0x00
P2 0x00
P3 0x01 or 0x04

2.6.22.2 Required Authentication
None

2.6.22.3 Definition
This function returns the applet life cycle as defined by the OpenPlatform spec. If P3 is equal to 4, it also returns several pieces of status information as well.

2.6.22.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>

2.6.22.5 Returned data
Returned Data has the following format if P3 = 0x01:
byte current life cycle

Returned Data has the following format if P3 = 0x04:
byte current life cycle
byte pinenabled
byte major protocol version
byte minor protocol version
2.6.23 CKYSeedRandom

2.6.23.1 Function Parameters
CLA 0xB0
INS 0x73
P1 Life Cycle
P2 0x00
P3 size

2.6.23.2 Required Authentication
None

2.6.23.3 Definition
Write size bytes to the on card random number generator as a seed value.

2.6.23.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
</tbody>
</table>

2.6.23.5 Returned data
None
2.6.24 CKYGetIssuerInfo

2.6.24.1 Function Parameters

CLA 0x84
INS 0xF6
P1 0x00
P2 0x00
P3 0xE0

2.6.24.2 Required Authentication

None

2.6.24.3 Definition

Read the free form Issuer Info from the card.

2.6.24.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_WRONG_LENGTH</td>
<td>P3 was not 0xE0</td>
</tr>
</tbody>
</table>

2.6.24.5 Returned data

The issuer info:

```c
byte[] issuerInfo
```
2.6.25 SecureSetIssuerInfo

2.6.25.1 Function Parameters

CLA 0x84
INS 0xF4
P1 0x00
P2 0x00
P3 0xE0

Data:
   byte[] issuerInfo

2.6.25.2 Required Authentication

Secure Channel

2.6.25.3 Definition

This function sets the free form issuerInfo field.

2.6.25.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_WRONG_LENGTH</td>
<td>P3 was not 0xE0</td>
</tr>
</tbody>
</table>

2.6.25.5 Returned data

None
2.6.26 CKYGetBuiltInACL

2.6.26.1 Function Parameters

CLA 0x84  
INS 0xFA  
P1 0x00  
P2 0x00  
P3 0x07

2.6.26.2 Required Authentication

None

2.6.26.3 Definition

Read built in ACL's from the token. The built in ACL's control who is allowed to create objects, keys or pins.

2.6.26.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_WRONG_LENGTH</td>
<td>P3 was not 0xE0</td>
</tr>
</tbody>
</table>

2.6.26.5 Returned data

The issuer info:

```
short create_object_ACL
short create_key_ACL
short create_pin_ACL
byte enable_ACL_change
```
2.6.27 SecureSetLifeCycle

2.6.27.1 Function Parameters
CLA 0x84
INS 0xF0
P1 Life Cycle
P2 0x00
P3 0x00

2.6.27.2 Required Authentication
Secure Channel

2.6.27.3 Definition
This function sets the applet life cycle as defined by the OpenPlatform spec.

2.6.27.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_WRONG_LENGTH</td>
<td>P3 was not zero</td>
</tr>
<tr>
<td>SW_INVALID_PARAMETER S</td>
<td>‘Life Cycle’ was not recognized by the CardManager, or transition to this Life Cycle state from the current state is not allowed by the Card Manager.</td>
</tr>
</tbody>
</table>

2.6.27.5 Returned data
None
2.6.28 SecureSetBuiltInACL

2.6.28.1 Function Parameters

CLA 0x84
INS 0xF8
P1 0x00
P2 0x00
P3 0x07

Data:

short create_object_ACL
short create_key_ACL
short create_pin_ACL
byte enable_ACL_change

2.6.28.2 Required Authentication

Secure Channel

2.6.28.3 Definition

This function changes the global create ACL's for the token. These ACL's are initialized at applet install time to be RA only. When the applet is installed, it is possible to tell the applet to 'enable_ACL_change'. By default enable_ACL_change is set to false. Once enable_ACL_change is set to false, the ACL's are locked and cannot be change. This function will fail with SW_OPERATION_NOT_ALLOWED.

2.6.28.4 Return codes

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_SUCCESS</td>
<td>Success</td>
</tr>
<tr>
<td>SW_WRONG_LENGTH</td>
<td>P3 was not 0x07</td>
</tr>
<tr>
<td>SW_OPERATION_NOT_ALLOWED</td>
<td>enable_ACL_change is set to false</td>
</tr>
</tbody>
</table>

2.6.28.5 Returned data

None
3 CoolKey Object ID and Data Format

CoolKey tokens contain data "blobs" that are readable by an application using the INS_READ_OBJECT request/response APDUs. Each such "object" is identified by a 32-bit "ObjectID". The objects are placed onto the token by the RA, and are read by other applications.

The token itself does not interpret the ObjectIDs or the object contents. The format of the ObjectIDs and the object blobs themselves is known only to the RA and the applications that read and use the object blobs.

This chapter specifies the format and meaning of the ObjectIDs and of the object blobs. This specification contains the proposed standard specification to be used by the time the Thundekey tokens are first shipped.

ObjectIDs are 4 bytes long, format is as follows:

- objectID byte[0], an ASCII letter, from the list below.
- objectID bytes[2-4], big endian, 24-bit binary object number.

Letters for objectID[0]

- Lower case letters signify objects containing PKCS11 object attributes in the format described below.
  - 'c' An object containing PKCS11 attributes for a certificate.
  - 'k' An object containing PKCS11 attributes for a public or private key
  - 'r' An object containing PKCS11 attributes for an CoolKey "reader".

- Upper case letters are reserved.

The general format of all readable objects containing PKCS11 attributes is:

All data, beginning with byte 0, is in type, length, value triplets. Each triplet contains one PKCS11 attribute. Each triplet has this form:

- 4-byte int, big endian, the CK_ATTRIBUTE_TYPE
- 2-byte short, big endian, "n", the number of bytes in the attribute value
- n bytes, the attribute value, format and content as defined by the CK_ATTRIBUTE_TYPE.

All attribute values are big-endian, except for character strings and ASN.1 encoded binary strings, which are stored in natural order (which some might also describe as big endian).
All objects must have CKA_CLASS attributes, and CKA_TOKEN attributes.

The PKCS11 attributes stored in a 'c' object for a certificate are:

<table>
<thead>
<tr>
<th>Attribute type</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>4</td>
<td>CKO_CERTIFICATE (1)</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>Var</td>
<td>nickname (a.k.a. &quot;friendly name&quot;)</td>
</tr>
<tr>
<td>CKA_CERTIFICATE_TYPE</td>
<td>4</td>
<td>CKC_X_509 (zero)</td>
</tr>
<tr>
<td>CKA_SUBJECT</td>
<td>Var</td>
<td>DER subject name of associated cert</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>20</td>
<td>SHA-1 hash of cert's SPKI</td>
</tr>
<tr>
<td>CKA_ISSUER</td>
<td>Var</td>
<td></td>
</tr>
<tr>
<td>CKA_SERIAL_NUMBER</td>
<td>Var</td>
<td></td>
</tr>
<tr>
<td>CKA_VALUE</td>
<td>Var</td>
<td>DER encoded cert</td>
</tr>
</tbody>
</table>

The PKCS11 attributes stored in a 'k' object for an RSA public key are:

<table>
<thead>
<tr>
<th>Attribute type</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>4</td>
<td>CKO_PUBLIC_KEY (2)</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>Var</td>
<td>nickname (a.k.a. &quot;friendly name&quot;)</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>4</td>
<td>CKK_RSA (zero)</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>20</td>
<td>SHA-1 hash of cert's SPKI</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_SUBJECT</td>
<td>Var</td>
<td></td>
</tr>
<tr>
<td>CKA_ENCRYPT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_VERIFY</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_VERIFY_RECOVER</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_WRAP</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>2</td>
<td>number of significant bits in CKA_MODULUS</td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>Var</td>
<td>RSA public key modulus</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>Var</td>
<td>(typically 3 bytes, value 0x0100001)</td>
</tr>
</tbody>
</table>

The PKCS11 attributes stored in a 'k' object for an RSA private key are:

<table>
<thead>
<tr>
<th>Attribute type</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>4</td>
<td>CKO_PRIVATE_KEY (3)</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>CKA_PRIVATE</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>var.</td>
<td>nickname of associated cert.</td>
</tr>
<tr>
<td>CKA_KEY_TYPE</td>
<td>4</td>
<td>CKK_RSA (zero)</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>20</td>
<td>SHA-1 hash of cert's SPKI</td>
</tr>
<tr>
<td>CKA_DERIVE</td>
<td>1</td>
<td>0 or 1 (0 for signature-only keys, 1 for others)</td>
</tr>
<tr>
<td>CKA_SUBJECT</td>
<td>var.</td>
<td>DER subject name of associated certificate</td>
</tr>
<tr>
<td>CKA_SENSITIVE</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>CKA_DECRYPT</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_SIGN</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_SIGN_RECOVER</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_UNWRAP</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CKA_MODULUS_BITS</td>
<td>2</td>
<td>number of significant bits in CKA_MODULUS</td>
</tr>
<tr>
<td>CKA_MODULUS</td>
<td>Var</td>
<td>RSA public key modulus</td>
</tr>
<tr>
<td>CKA_PUBLIC_EXPONENT</td>
<td>Var</td>
<td>(typically 3 bytes, value 0x0100001)</td>
</tr>
</tbody>
</table>

The PKCS11 module presently constructs the CKA_VALUE attribute for each certificate from the content of the corresponding 'C' object.

The PKCS11 attributes stored in an 'r' object for a "reader" are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_CLASS</td>
<td>4</td>
<td>CKO_MOZ_READER (1)</td>
</tr>
<tr>
<td>CKA_TOKEN</td>
<td>1</td>
<td>1 (true)</td>
</tr>
<tr>
<td>CKA_LABEL</td>
<td>Var</td>
<td>reader name string</td>
</tr>
<tr>
<td>CKA_MOZ_IS_Cool_KEY</td>
<td>1</td>
<td>1 (true)</td>
</tr>
</tbody>
</table>

Note: the reader object is currently not stored on the token, nor created by the RA. It is created by the pkcs11 module itself. But it exists in the same ObjectID space as the objects on the token.
4 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 CodeVerification The process by which an Applet Instance authenticates a host application comparing the host provided PIN Code with one of the onboard 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) command A two byte code as defined in ISO-7816 as to the status of a SmartCard

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