CoolKey Applet Interface

(Based strongly on "MUSCLE Cryptographic Card Edge Definition for Java[1] Enabled SmartCards" by David Corcoran and Tommaso Cucinotta)

[1] *Java® and Java Card® 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

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. 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 0xFFFFFF00, 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:

Hex Value	**Meaning**
0x0000	Operation *never* allowed
0x0004	Identity n.2 (PIN) required
0x4001	*Either* Identity n.0 (PIN) *or* identity n.14 (strong) required
0xFFFF	Operation *always* 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) 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:

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

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 `(page Error: Reference source not found)` and Error: Reference source not found`(Page Error: Reference source not found)` commands for further details).

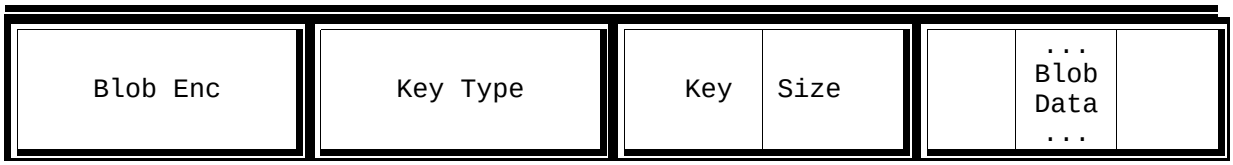
2.2 Key blobs

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;
```

|----------Key Blob Header-----------|



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
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

	Blob	Header					
P	Size			. . . P Value . . .			
Q	Size			. . . Q Value . . .			
PQ	Size			. . . PQ Value . . .			
DP1	Size			. . . DP1 Value . . .			
DQ1	Size			. . . DQ1 Value . . .			

Key Type RSA_PRIVATE

	Blob	Header					
Mod	Size			. . . Modulus Value . . .			
Prv Exp	Size		Private	. . . Exponent . . .	Value		

Key Type RSA_PUBLIC

	Blob	Header				
Mod		Size			. . . Modulus Value . . .	
Pub Exp		Size		Public	. . . Exponent . . .	Value

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_PRIVATE

	Blob	Header				
G		Size			. . . G Value . . .	
P		Size			. . . P Value . . .	
Q		Size			. . . Q Value . . .	
X		Size			. . . X Value . . .	

Key Type DSA_PUBLIC

	Blob	Header				
G	Size			. . . G Value . . .		
P	Size			. . . P Value . . .		
Q	Size			. . . Q Value . . .		
Y	Size			. . . Y Value . . .		

2.2.1.3 DES KeyBlob Definitions

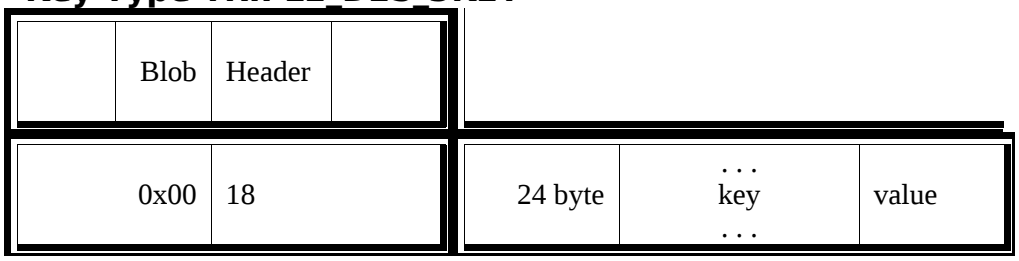
Key Type DES

	Blob	Header				
0x00	08		8 byte	. . . key . . .	value	

Key Type TRIPLE_DES

	Blob	Header				
0x00	10		16 byte	. . . key . . .	value	

Key Type TRIPLE_DES_3KEY



2.3 Summary of commands

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

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.

2.5.1.1 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

Value	*Symbolic Name*	*Description*
		the identity to block
9C 03	SW_OPERATION_NOT_ALLOWED	Operation not allowed because of the 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_SEQUENCE_END	No more data in list.
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

Value	*Symbolic Name*	*Description*
6D 00	SW_ERROR_INS	(ISO) Instruction code not recognized

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
      Byte[]      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

Symbolic Name	*Description*
SW_SUCCESS	Success
SW_UNAUTHORIZED	Security Requirement not satisfied
SW_XXX	Other errors due to decryption problems and format errors in the key that is being loaded or incorrect parameter values

2.6.1.9 Returned data

```
MacData:
     Byte[]          Calculated Mac value
```

2.6.1.10Issues

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

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_UNAUTHORIZED	Security Requirement not satisfied
SW_XXX	Other errors due to decryption problems and format errors in the key that is being loaded

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 `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 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_ENC_PLAIN (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.

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.

2.6.3.7

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_ON_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 `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_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.

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
`SW_INCORRECT_ALG`	Algorithm does not match the key type
`SW_OPERATION_NOT_ALLOWED`	Operation not allowed for this key type or algorithm

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

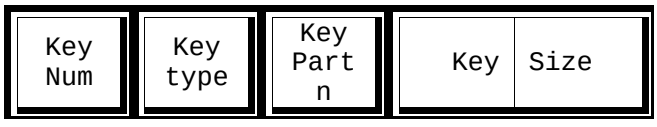
The data will be trailed with `SW_SUCCESS`. When the list has no more entries just `SW_SEQUENCE_END` will be returned.

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
```



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.

Symbolic Name	***Description***
`SW_INCORRECT_P1`	Sequence option is not valid
`SW_SEQUENCE_END`	No more keys in the list

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.

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

2.6.7 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.

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

2.6.7.10Returns

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

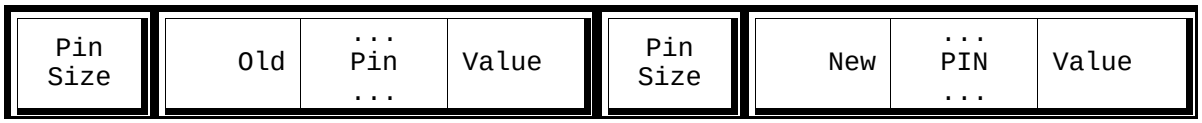
Supplied Pin must be valid.

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
```



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.

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

2.6.9 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.

Least significant byte:

Bit	PIN Number	Bitmask Value
1	Pin #0	0x01
2	Pin #1	0x02
3	Pin #2	0x04
…	…	…
8	Pin #7	0x80

Most significant byte is RFU.

Symbolic Name	***Description***
`SW_SUCCESS`	Success

2.6.10 SecureSetPIN

2.6.10.1Function Parameters

```
CLA   0x80
INS   0x04
P1    Pin number
P2    0x00
P3    <Pin length>

DATA  Pin

Pin:
      Byte[]      New Pin value
```

2.6.10.2Required Authentication

Secure Channel

2.6.10.3Definition

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.

2.6.10.4Notes

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

2.6.10.5Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_UNAUTHORIZED	Security Requirement not satisfied

2.6.10.6Returned data

None

2.6.10.7Issues

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

2.6.11.1Function Parameters

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

2.6.11.2Required Authentication

PIN nonce

2.6.11.3Definition

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

2.6.12.1Function 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;
```

	Object	ID			Object	Size	

R	R	Object W	ACL W	D	D

2.6.12.3Required Authentication

Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.12.4Definition

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.

2.6.12.6Notes

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.

2.6.12.8Return 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.

Symbolic Name	***Description***
`SW_UNAUTHORIZED`	Appropriate Identity 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

2.6.13 CKYDeleteObject

2.6.13.1Function Parameters

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

[DATA]
Long Object ID
```

2.6.13.2Required Authentication

Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.13.3Definition

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.

2.6.13.4Parameters

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

2.6.13.5Notes

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.

2.6.13.6Return 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

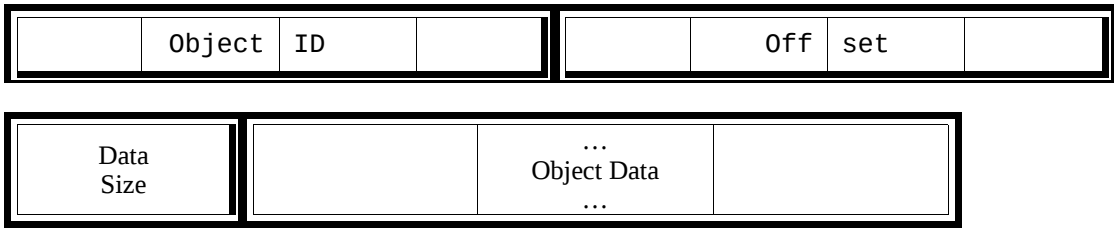
2.6.14 CKYWriteObject

2.6.14.1Function Parameters

2.6.14.2

```
CLA 0x84 or 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
```



2.6.14.3Required Authentication

Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.14.4Definition

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.

2.6.14.6Notes

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.

2.6.14.8Return 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

2.6.15 CKYReadObject

2.6.15.1Function 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
```

2.6.15.3Required Authentication

Secure Channel (class 0x84) or PIN nonce (class 0xB0)

2.6.15.4Definition

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.

2.6.15.6Notes

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.

2.6.15.8Return 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.

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

2.6.15.10Returned data

[DATA]

Byte[] readData;

Short Status Word;

2.6.16 SecureReadIOBuffer

2.6.16.1Function Parameters

```
CLA   0x80
INS   0x08
P1    <Data length>
P2    0x00
P3    0x02

DATA  Parameters

Parameters:
      Short Offset
```

2.6.16.2Required Authentication

Secure Channel

2.6.16.3Definition

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.

2.6.16.4Notes

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

2.6.16.5Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_UNAUTHORIZED	Security Requirement not satisfied
SW_INVALID_PARAM	The offset and size are too large for the temporary buffer

2.6.16.6Returned data

None

2.6.16.7Issues

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

2.6.17.1Function Parameters

2.6.17.2

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

2.6.17.3Required Authentication

None.

2.6.17.4Definition

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
```

2.6.17.6Notes

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.

2.6.17.8Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_SEQUENCE_END	No more objects in the list

2.6.17.9Returned data

2.6.17.10

Data returned if an object was found:

	Object	ID			Object	Size	

R	R	Object W	ACL W	D	D	Status	Word

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

2.6.18.1Function Parameters

2.6.18.2

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

2.6.18.3Required Authentication

None

2.6.18.4Definition

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.

2.6.18.5Return 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

2.6.19.1Function Parameters

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

2.6.19.2Required Authentication

None

2.6.19.3Definition

This function returns success if the applet is selected.

2.6.19.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success

2.6.19.5Returned data

None

2.6.20 CKYGetRandom

2.6.20.1Function Parameters

```
CLA 0xB0
INS 0x73
P1 0x00
P2 0x00
P3 0x04
```

2.6.20.2Required Authentication

None

2.6.20.3Definition

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

2.6.20.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success

2.6.20.5Returned data

Returned Data has the following format:

```
Byte[] randomData
```

2.6.21 CKYGetBuildID

2.6.21.1Function Parameters

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

2.6.21.2Required Authentication

None

2.6.21.3Definition

This function returns the unique applet build ID. Each applet build has it's own unique build ID calculated at build time.

2.6.21.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success

2.6.21.5Returned data

Returned Data has the following format:

```
short Major Build ID
short Minor Build ID
```

2.6.22 CKYGetLifeCycle

2.6.22.1Function Parameters

```
CLA 0xB0
INS 0xF2
P1 0x00
P2 0x00
P3 0x01 or 0x04
```

2.6.22.2Required Authentication

None

2.6.22.3Definition

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.

2.6.22.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success

2.6.22.5Returned 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

2.6.23.1Function Parameters

```
CLA 0xB0
INS 0x73
P1 Life Cycle
P2 0x00
P3 size
```

2.6.23.2Required Authentication

None

2.6.23.3Definition

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

2.6.23.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success

2.6.23.5Returned data

None

2.6.24 CKYGetIssuerInfo

2.6.24.1Function Parameters

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

2.6.24.2Required Authentication

None

2.6.24.3Definition

Read the free form Issuer Info from the card.

2.6.24.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_WRONG_LENGTH	P3 was not 0xE0

2.6.24.5Returned data

The issuer info:

```
byte[]  issuerInfo
```

2.6.25 SecureSetIssuerInfo

2.6.25.1Function Parameters

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

Data:
      byte[] issuerInfo
```

2.6.25.2Required Authentication

Secure Channel

2.6.25.3Definition

This function sets the free form issuerInfo field.

2.6.25.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_WRONG_LENGTH	P3 was not 0xE0

2.6.25.5Returned data

None

2.6.26 CKYGetBuiltInACL

2.6.26.1Function Parameters

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

2.6.26.2Required Authentication

None

2.6.26.3Definition

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

2.6.26.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_WRONG_LENGTH	P3 was not 0xE0

2.6.26.5Returned data

The issuer info:

```
short create_object_ACL
short create_key_ACL
short create_pin_ACL
byte  enable_ACL_change
```

2.6.27 SecureSetLifeCycle

2.6.27.1Function Parameters

```
CLA 0x84
INS 0xF0
P1 Life Cycle
P2 0x00
P3 0x00
```

2.6.27.2Required Authentication

Secure Channel

2.6.27.3Definition

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

2.6.27.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_WRONG_LENGTH	P3 was not zero
SW_INVALID_PARAMETERS	'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.

2.6.27.5Returned data

None

2.6.28 SecureSetBuiltInACL

2.6.28.1Function 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
```

2.6.28.2Required Authentication

Secure Channel

2.6.28.3Definition

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.

2.6.28.4Return codes

Symbolic Name	***Description***
SW_SUCCESS	Success
SW_WRONG_LENGTH	P3 was not 0x07
SW_OPERATION_NOT_ALLOWED	enable_ACL_change is set to false

2.6.28.5Returned 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 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:

Attribute type	Length	Value
CKA_CLASS	4	CKO_CERTIFICATE (1)
CKA_TOKEN	1	1 (true)
CKA_LABEL	Var	nickname (a.k.a. "friendly name")
CKA_CERTIFICATE_TYPE	4	CKC_X_509 (zero)
CKA_SUBJECT	Var	DER subject name of associated cert
CKA_ID	20	SHA-1 hash of cert's SPKI
CKA_ISSUER	Var	
CKA_SERIAL_NUMBER	Var	
CKA_VALUE	Var	DER encoded cert

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

Attribute type	Length	value
CKA_CLASS	4	CKO_PUBLIC_KEY (2)
CKA_TOKEN	1	1 (true)
CKA_LABEL	Var	nickname (a.k.a. "friendly name")
CKA_KEY_TYPE	4	CKK_RSA (zero)
CKA_ID	20	SHA-1 hash of cert's SPKI
CKA_DERIVE	1	
CKA_SUBJECT	Var	
CKA_ENCRYPT	1	
CKA_VERIFY	1	
CKA_VERIFY_RECOVER	1	
CKA_WRAP	1	
CKA_MODULUS_BITS	2	number of significant bits in CKA_MODULUS
CKA_MODULUS	Var	RSA public key modulus
CKA_PUBLIC_EXPONENT	Var	(typically 3 bytes, value 0x010001)

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

Attribute type	Length	value
CKA_CLASS	4	CKO_PRIVATE_KEY (3)
CKA_TOKEN	1	1 (true)
CKA_PRIVATE	1	1 (true)

CKA_LABEL	var.	nickname of associated cert.
CKA_KEY_TYPE	4	CKK_RSA (zero)
CKA_ID	20	SHA-1 hash of cert's SPKI
CKA_DERIVE	1	0 or 1 (0 for signature-only keys, 1 for others)
CKA_SUBJECT	var.	DER subject name of associated certificate
CKA_SENSITIVE	1	1 (true)
CKA_DECRYPT	1	
CKA_SIGN	1	
CKA_SIGN_RECOVER	1	
CKA_UNWRAP	1	
CKA_MODULUS_BITS	2	number of significant bits in CKA_MODULUS
CKA_MODULUS	Var	RSA public key modulus
CKA_PUBLIC_EXPONENT	Var	(typically 3 bytes, value 0x010001)

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:

Attribute type	Length	value
CKA_CLASS	4	CKO_MOZ_READER (1)
CKA_TOKEN	1	1 (true)
CKA_LABEL	Var	reader name string
CKA_MOZ_IS_Cool_KEY	1	1 (true)

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 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.