CardMan 5x21-CL Reader Developer’s Guide

Document Version: 1.11

Abstract: Guide for developers who want to integrate contactless storage or CPU cards using OMNIKEY CardMan 5x21 smart card readers.

Last modified: 22-NOV-2006

Disclaimer:

Table Of Contents

1 Getting Started – Quick Guide to Access Contactless Cards

This document is intended as a guide for software developers who want to integrate contactless memory or CPU cards using OMNIKEY CardMan 5x21 smart card readers.

This chapter describes how to install the drivers necessary to operate the CardMan 5x21 in a Windows based environment. Please not that other operating systems such as Linux are also supported by the CardMan 5x21.

1.1 Driver Installation

The OMNIKEY CardMan 5x21 driver is mandatory for all systems that require support for contactless smart cards.

CardMan 5x21 is a CCID compliant device. This means that the contact interface can be operated without an OMNIKEY proprietary driver installed. However, for contactless cards, the OMNIKEY proprietary CardMan 5x21 driver is necessary.

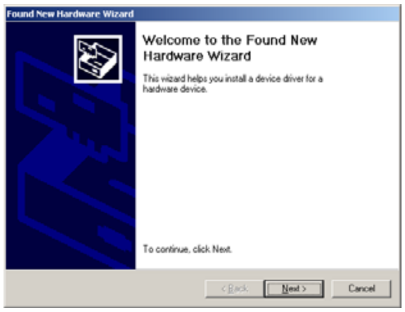
The following steps describe how to install the CardMan 5x21 driver:

1. First, go to www.omnikey.com. Then, select the support/download section and download the latest CardMan 5x21 driver installation package for Windows.

2. Run the installation package and follow the instructions on the screen. The installation package will extract all the necessary driver files to your hard drive. Please note the location to which the files were copied. Please note that at this time you have only extracted, not installed the driver files.

3. Connect the reader to a USB Port of your computer.

4. The "**Found New Hardware Wizard"** will appear. To continue driver installation, click "**Next"**.



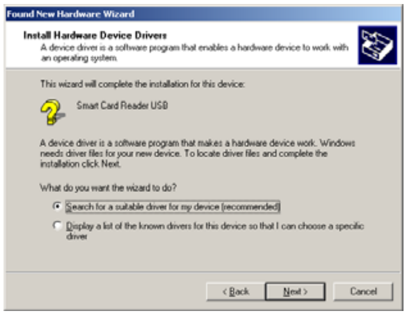
Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

Note: On Windows XP systems, the Microsoft Windows CCID Class driver may be activated without showing the "Found New Hardware Wizard". If this is the case, the Microsoft PC/SC driver` must be replaced manually with the OMNIKEY proprietary PC/SC driver. This can be done using the Device Manager.

5. In the next dialogue, choose "**Search for a suitable driver for my device (recommended)"** and click "**Next"**.



Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

Smart Card Reader USB

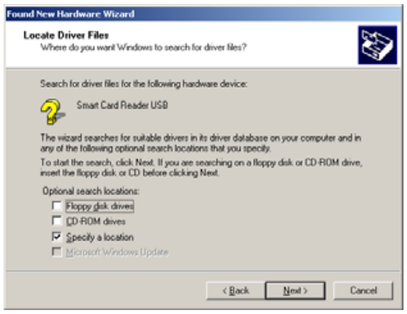
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

● Search for a suitable driver for my device (recommended)

○ Display a list of the known drivers for this device so that I can choose a specific driver

6. Now, choose “**Specify a Location”** and click “**Next”**.



Locate Driver Files

Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

Smart Card Reader USB

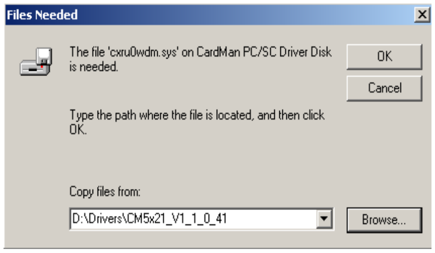
The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

☐ Floppy disk drives
☐ CD-ROM drives
☑ Specify a location
☐ Microsoft Windows Update

7. Press “**Browse”** and go to the location where you previously installed the driver package. To continue press “**OK”**.



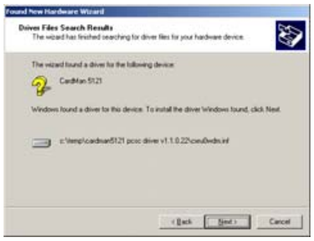
The file 'cxru0wdm.sys' on CardMan PC/SC Driver Disk is needed.

Type the path where the file is located, and then click OK.

Copy files from:

D:\Drivers\CM5x21_V1_1_0_41

8. If the driver was found in this location press “**Next”** to continue.



Driver Files Search Results

The wizard has finished searching for driver files for your hardware device.

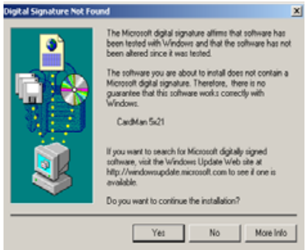
The wizard found a driver for the following device:

CardMan 5121

Windows found a driver for this device. To install the driver Windows found, click Next.

c:\temp\cardman5121 pcsc driver v1.1.0.22\cxru0wdm.inf

9. If the driver is a test driver (a BETA driver that is not digitally signed), the following dialogue will appear. It can be accepted by clicking “**Yes”**.



Digital Signature Not Found

The Microsoft digital signature affirms that software has been tested with Windows and that the software has not been altered since it was tested.

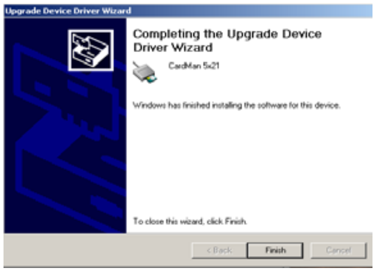
The software you are about to install does not contain a Microsoft digital signature. Therefore, there is no guarantee that this software works correctly with Windows.

CardMan 5x21

If you want to search for Microsoft digitally signed software, visit the Windows Update Web site at http://windowsupdate.microsoft.com to see if one is available.

Do you want to continue the installation?

10. The following message should appear and the green LED on the CardMan 5x21 reader should light up.



Upgrade Device Driver Wizard

Completing the Upgrade Device Driver Wizard

CardMan 5x21

Windows has finished installing the software for this device.

To close this wizard, click Finish.

If the installation was successful, the green LED on the reader will light up and the reader will be listed in the diagnostic tool as “CardMan 5x21”. Now your reader is ready for use and you can do a quick smart card system check using the OMNIKEY Diagnostic Tool described in the chapter entitled “Diagnostic Tool”.

1.1.1 Reader Name for Contact/Contactless Slot

OMNIKEY CardMan 5x21 is a “dual slot” reader. This means that from the application and smart card resource manager point of view there are two readers available, each represented by its respective reader name. “OMNIKEY CardMan 5x21 n” identifies the contact slot and “CardMan 5x21**-CL** n” stands for the contactless slot. The “n” represents a slot number 0, 1…, etc. This allows card tracking though contact and air interface, respectively.

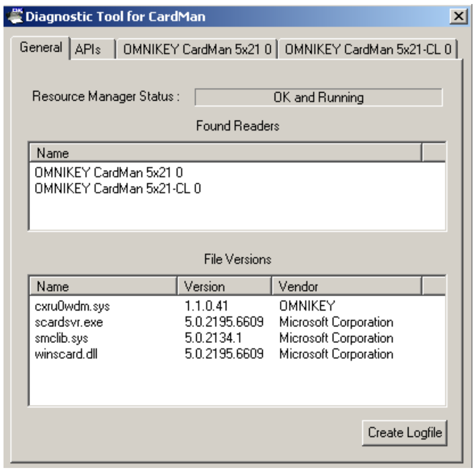
1.2 Diagnostic Tool

The OMNIKEY Diagnostic tool is a great tool for a quick test of the smart card system. It lists all available OMNIKEY readers, driver files with version information, firmware version information, and also allows the configuration of the RFID/air interface.

‘Diagnostic Tool’ can be started from the **Control Panel**.

1.2.1 Driver Version Detection

The **General** Tab shows if the ‘Resource Manager’ is running. Select this tab to find out more about version and manufacturer data of smart card system services, DLLs, and drivers.



Resource Manager Status : OK and Running

Found Readers

Name
OMNIKEY CardMan 5x21 0
OMNIKEY CardMan 5x21-CL 0

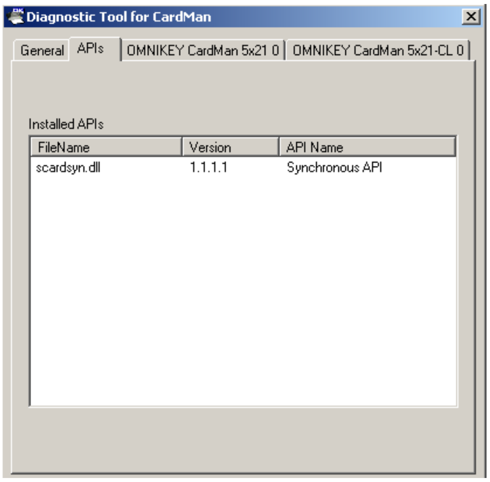
File Versions

Name	Version	Vendor
cxru0wdm.sys	1.1.0.41	OMNIKEY
scardsvr.exe	5.0.2195.6609	Microsoft Corporation
smclib.sys	5.0.2134.1	Microsoft Corporation
winscard.dll	5.0.2195.6609	Microsoft Corporation

Diagnostic Tool, General Tab

1.2.2 OMNIKEY Proprietary API Detection

The **API tab** shows the APIs that have been installed on your system, including the OMNIKEY Synchronous API.



Installed APIs

FileName	Version	API Name
scardsyn.dll	1.1.1.1	Synchronous API

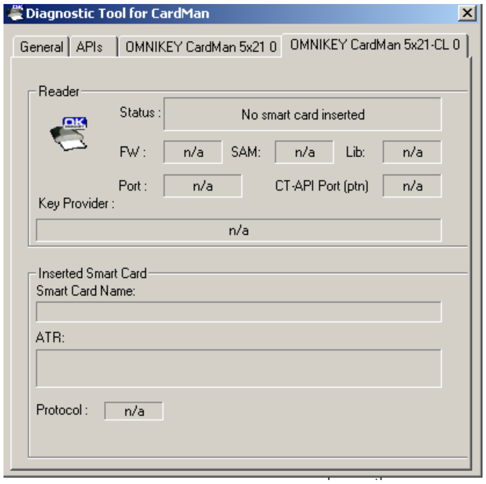
Diagnostic Tool, API Tab

1.2.3 Card and Reader Detection

The OMNIKEY Diagnostic tool creates a separate tab for each available OMNIKEY reader interface. The tabs indicate their respective reader names - the same names you will be using within the PC/SC framework.

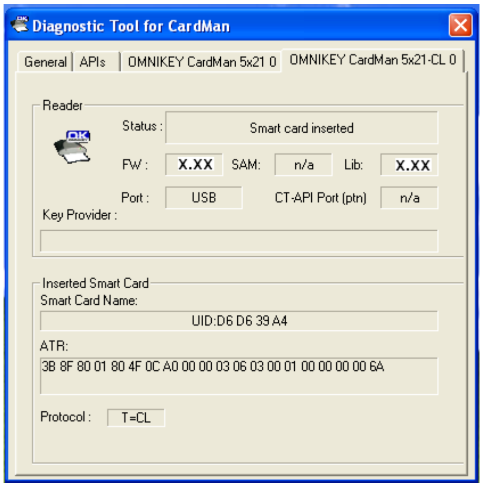
For a quick connectivity test of your contactless card, select the “OMNIKEY CardMan 5x21-CL 0” tab and place a contactless card on the reader. As soon as the card is detected, the **Status** field will switch from “no smart card inserted” to “smart card inserted” and the **ATR** field will display the card’s ATR. Please refer to the chapter about ATR for further information on how the Answer to Reset (ATR) is generated for contactless smart cards.

The Diagnostic Tool has an internal flat database that allows a quick lookup of the ATR. If it is a known card, a description will be displayed in the **Smart Card Name** field. For contactless cards the card’s unique ID (UID) will be displayed in the **Smart Card Name** field and in the **Protocol** field T=CL will be displayed.



Status : No smart card inserted
FW : n/a SAM: n/a Lib: n/a
Port : n/a CT-API Port (ptn) n/a
Key Provider : n/a
Smart Card Name:
ATR:
Protocol : n/a

Diagnostic Tool, Reader Tab



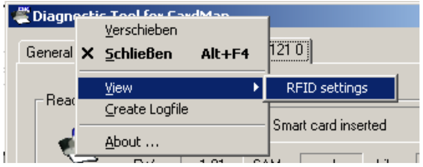
Status : Smart card inserted
FW : X.XX SAM: n/a Lib: X.XX
Port : USB CT-API Port (ptn) n/a
Key Provider :
Smart Card Name: UID:D6 D6 39 A4
ATR: 3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
Protocol : T=CL

Diagnostic Tool, Reader Tab (card inserted)

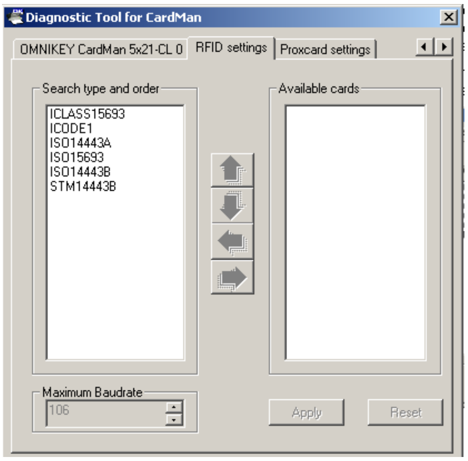
1.2.4 Card Type Detection and RFID Settings

CardMan 5x21-CL supports multiple 13.56MHz contactless standards and protocols, including but not limited to, ISO14443A, ISO14443B, ISO15694, iCLASS, I-CODE. As there is no physical card switch, the only way to get information about a card in the field is by trial and error in a predefined search order. The built-in anti-collision mechanisms ensure that, once a card is detected, it is the only card the reader interface is connected with.

The OMNIKEY Diagnostic tool has an RFID Settings tab that allows configuration of the card types the reader should look for and their respective search order. However, the RFID Settings tab must first be enabled by right-clicking on the title bar of the Diagnostic Tool window and then choosing **View->RFID settings** from the drop-down menu.



Diagnostic Tool, activate RFID settings



Search type and order:

ICLASS15693
ICODE1
ISO14443A
ISO15693
ISO14443B
STM14443B

Maximum Baudrate: 106

Diagnostic Tool, RFID settings Tab

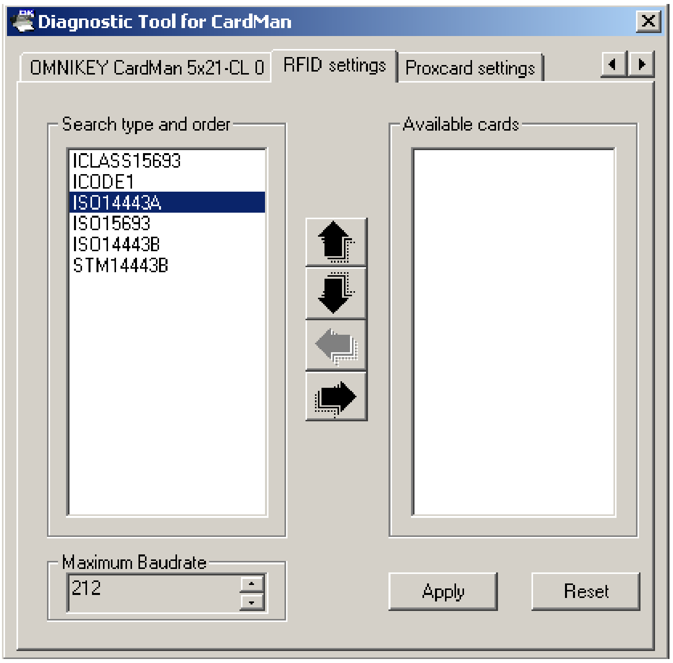
The left windowpane contains a list of all active card types. The right windowpane contains a list of all available card types that are supported by the reader but are not included in the card search. Card types from the left windowpane can be moved to the right windowpane, and vice versa, using the ◀ and ▶ buttons. With the ▲ and ▼ buttons the search order in the active list can be changed.

The setting has to be activated using the **Apply** button. The **Reset** button discards any unsaved changes.

Note: The search order is forward-looking to improve system performance. This means that the last successfully detected card type automatically moves to the top of the search order, regardless of its position within the order set on the RFID settings tab.

1.2.5 Air Interface Baud Rate Configuration

For ISO 14443 type cards the air interface transmission speed can be 106 kbps, 212 kbps, 424 kbps, or 848 kbps. By default, the contactless interface is set to 424 kbps. It can be changed to a different value through the RFID settings tab of the Diagnostic Tool.



ICLASS15693
ICODE1
ISO14443A
ISO15693
ISO14443B
STM14443B

Maximum Baudrate: 212

Diagnostic Tool, Baud Rate Change

To change the baud rate, select the card type (ISO14443A or ISO14443B) and change the ‘Maximum Baud Rate’ field. Finalize your setting by pressing the ‘Apply’ button.

2 PC/SC 2.0

With the OMNIKEY CardMan 5x21 PC/SC driver, ISO14443A/B or ISO15693 compliant contactless cards can be accessed through the same framework as ISO7816 compliant contact cards. This makes card integration a snap for any developer who is already familiar with PC/SC. Even valuable PC/SC resource manager functions, such as card tracking, are available for contactless card integration.

The Microsoft Developer Network (MSDN) Library contains valuable information and a complete documentation of the SCard API within the MSDN Platform SDK: “Security” section.

Contactless CPU cards can be accessed directly via PC/SC. For storage cards other than Mifare cards, an additional library – the OMNIKEY synchronous API – is necessary. Whether using direct PC/SC access or the OMNIKEY synchronous API, only a small set of functions is required to write your first “hello card” program.

2.1 How to Access Contactless Cards via PC/SC

The following steps provide a guideline to create your first contactless smart card application using industry standard, PC/SC compliant API function calls. The function definitions provided below are taken verbatim from the MSDN Library [MSDNLIB]. For additional descriptions of these and other PC/SC functions provided by the Microsoft Windows PC/SC smart card components, please refer directly to the MSDNLIB. The following functions are defined in the Microsoft Platform SDK (Security section).

1. **Establish Context**
   This step will initialize the PC/SC API and allocate all resources necessary for a smart card session. The SCardEstablishContext function establishes the resource manager context (the scope) within which database operations are performed.

```
LONG SCardEstablishContext( IN DWORD dwScope,
                            IN LPCVOID pvReserved1,
                            IN LPCVOID pvReserved2,
                            OUT LPSCARDCONTEXT phContext);
```

2. **Get Status Change**
   Next, check the status of the reader for card insertion, removal, or availability of the reader. SCardGetStatusChange. This function blocks execution until the current availability of the cards in a specific set of readers changes. The caller supplies a list of readers to be monitored and the maximum amount of time (in milliseconds) that it is willing to wait for an action to occur on one of the listed readers.

```
LONG SCardGetStatusChange( IN SCARDCONTEXT hContext,
                           IN DWORD dwTimeout,
                           IN OUT LPSCARD_READERSTATE rgReaderStates,
                           IN DWORD cReaders);
```

3. **List Readers**
   Then, get a list of all PC/SC readers using the SCardListReaders function. Look for "OMNIKEY CardMan 5x21-CL 0" in the returned list. If multiple CardMan 5x21 readers are connected to your system, they will be enumerated. Examples: "OMNIKEY CardMan 5x21-CL 1", "OMNIKEY CardMan 5x21-CL 2".

   Always make sure to analyze the complete string. CardMan 5x21 also has a contact interface. Look for the "-CL" in the reader mane to make sure that you are really referring to the contactless interface in the following calls.

```
LONG SCardListReaders( IN SCARDCONTEXT hContext,
                       IN LPCTSTR mszGroups,
                       OUT LPTSTR mszReaders,
                       IN OUT LPDWORD pcchReaders);
```

4. **Connect**
   Now, you can connect to the card. The SCardConnect function establishes a connection (using a specific resource manager context) between the calling application and a smart card contained by a specific reader. If no card exists in the specified reader, an error is returned.

```
LONG SCardConnect( IN SCARDCONTEXT hContext,
                   IN LPCTSTR szReader,
                   IN DWORD dwShareMode,
                   IN DWORD dwPreferredProtocols,
                   OUT LPSCARDHANDLE phCard,
                   OUT LPDWORD pdwActiveProtocol);
```

   Note: For iCLASS cards use T=0 protocol (mandatory).

5. **Exchange Data and Commands with the Card**
   Next, exchange command and data via APDUs. The SCardTransmit function sends a service request to the smart card, and expects to receive data back from the card.

```
LONG SCardTransmit( IN SCARDHANDLE hCard,
                    IN LPCSCARD_I0_REQUEST pioSendPci,
                    IN LPCBYTE pbSendBuffer,
                    IN DWORD cbSendLength,
                    IN OUT LPSCARD_IO_REQUEST pioRecvPci,
                    OUT LPBYTE pbRecvBuffer,
                    IN OUT LPDWORD pcbRecvLength);
```

   *Note: For storage cards that are not supported by PC/SC 2.0, you need to call an OMNIKEY proprietary API function such as **SCardCLICCTransmit** instead. This function exposes additional functionality of the CardMan 5x21-CL reader that is not yet defined in PC/SC standards. Other than that, you are still using the standard PC/SC framework to track cards, list readers, etc. Even the smart card handle is the same.*

6. **Disconnect**
   It is not absolutely necessary to disconnect the card after the completion of all transactions, but it is recommended. The SCardDisconnect function terminates a connection previously opened between the calling application and a smart card in the target reader.

```
LONG SCardDisconnect( IN SCARDHANDLE hCard,
                      IN DWORD dwDisposition);
```

7. **Release**
   This step ensures that all system resources are being released. The SCardReleaseContext function closes an established resource manager context, freeing any resources allocated under that context.

```
LONG SCardReleaseContext( IN SCARDCONTEXT hContext);
```

2.2 ATR Generation

Unlike contact cards, contactless cards don't generate an ATR. They generate an Answer to Select (ATS) instead. To make contactless cards available within the PC/SC framework, CardMan 5x21 generates a PC/SC compliant ATR according to PC/SC v2.01 "Interoperability Specification for ICCs and Personal Computer Systems" [PCSC 2.01].

The documents can be downloaded from the PC/SC Workgroup at the following web address: http://www.pcscworkgroup.com/specifications/specdownload.php

2.2.1 CPU Cards

Contactless smart cards (cards with a CPU) expose their ATS or information bytes via ATR mapping according to PCSC 2.01, Part 3: Requirements for PC-Connected Interface Devices, 3.1.3.2.3.1, Table 3.5.

2.2.2 Storage Cards

The ATR of storage cards (i.e. cards without a CPU) is composed as described in PCSC 2.01, Part 3: Requirements for PC-Connected Interface Devices, 3.1.3.2.3.1, Table 3.6. In order to allow the host application to identify a storage and card type properly, its standard and card name is mapped according to the Part 3 Supplemental Document of PCSC 2.01.

Note: The Registered Application Provider Identifier (RID) returned by the CardMan 5x21 for storage cards (cards without a CPU) is A0 00 00 06 0A, indicating a PC/SC compliant ATR generation.

3 Accessing Asynchronous Cards

Asynchronous cards are cards with CPU or memory cards that are accessible via standard PC/SC using Microsoft's library "winscard.dll". This type of cards supports at least one of the asynchronous protocols T=0 or T=1. The Microsoft Platform SDK contains PC/SC sample code for Visual C/C++ and Visual Basic.

No additional libraries or third-party software components are necessary to integrate contactless CPU cards.

3.1 DESFire Card

According to [DESFIRE], DESFire cards can be accessed via ISO7816-4 compliant framed APDU commands (ISO7816-4 framing).

3.1.1 Example: Write Card Data via ISO 7816-4 Framed APDU

Command Syntax

CLA	INS	P1	P2	Lc	File No.	Offset	Length	Data	Le
‘90’	‘3D’	‘00’	‘00’	‘xx’	‘xx’	‘xxxxxx’	‘xxxxxx’	‘xx’ … ‘xx’	‘00’

Lc = 7+ DataLength; Le=0 (no other values accepted)

Response Syntax

Response Data	SW1	SW2
empty	‘xx’	‘xx’

Status Codes

SW1	SW2	Description
'90'	'00'	success
'91’	'xx	error (refer to DESFire data sheet)

3.1.2 Example: Read Card Data via ISO 7816-4 Framed APDU

Command Syntax

CLA	INS	P1	P2	Lc	File No.	Offset	Length	Data	Le
‘90’	‘3D’	‘00’	‘00’	‘07’	‘xx’	‘xxxxxx’	‘LLLLLL’	empty	‘00’

Le=0 (no other values accepted)

Response Syntax

Response Data	SW1	SW2
‘xx’ ... ‘xx’ (‘LLLLLL’ bytes)	‘xx’	‘xx’

Status Codes

SW1	SW2	Description
'90'	'00'	success
'91’	'xx	error (refer to DESFire data sheet)

4 Accessing Synchronous Cards (Storage Cards)

OMNIKEY provides two ways to integrate contactless storage cards. They can either be accessed through OMNIKEY's proprietary synchronous API library, or, for cards such as Mifare cards, directly via PC/SC 2.0 compliant function calls. Storage cards that are not supported directly through PC/SC 2.0 compliant APDU exchanges, can only be accessed via OMNIKEY proprietary synchronous API.

The synchronous API for Windows systems resides in a DLL named "scardsyn.dll". It can be downloaded from OMNIKEY's website. The download also contains sample code for Mifare and iCLASS cards. For further information about this API, refer to the help file "cmsync.hlp" available in the c:\omnikey\hlp folder after installation of the synchronous API with default settings.

The OMNIKEY Synchronous API is typically used whenever a card has not yet found its way into the PC/SC 2.0 standard. Currently, only Mifare cards can be integrated via PC/SC 2.0 compliant APDu.

	Card can be integrated via	
	PC/SC 2.0 compliant APDUs	**OMNIKEY Synchronous API**
Mifare	yes	yes
iCLASS	no	yes

No special drivers are required for a PC/SC 2.0 compliant card integration on Windows or Linux. OMNIKEY's latest drivers provide seamless cross-platform support allowing industry standard-compliant contactless card integration.

4.1 Mifare Card

CardMan 5x21 supports Mifare 1K, Mifare 4K and Mifare Ultra Light cards. The following functions are supported:

getUID	implemented according to [PCSC 2.01]
LoadKey	
Authenticate	
Verify	
Update Binary	
Read Binary	
Increment	OMNIKEY proprietary extension of PC/SC
Decrement	OMNIKEY proprietary extension of PC/SC

Please refer to the [PCSC 2.01] and [MIFARE] for documentation of PC/SC 2.0 compliant Mifare card access. The following section only describes usage of functions that are not already documented in [PCSC 2.01]. They are part of an OMNIKEY proprietary extension of PC/SC.

4.1.1 Mifare Increment (Card Command)

This command will increment the value of a block, if the card and block support this functionality:

Command Syntax

CLA	‘FF’
INS	‘D4’
P1	MSB of block address
P2	LSB of block address
LC	1
Data Field	one byte value indicating block increment
Le	empty

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success
'65'	‘81’	memory failure (unsuccessful increment)
‘69’	‘81’	incompatible command
‘69’	‘82’	security status not satisfied
‘69’	‘86’	command not allowed
‘6A’	‘81’	function not supported
‘6A’	‘82’	invalid block address

4.1.2 Mifare Decrement (Card Command)

This command will decrement the value of a block, if the card and block support this functionality:

Command Syntax

CLA	‘FF’
INS	‘D8’
P1	MSB of block address
P2	LSB of block address
LC	1
Data Field	one byte value indicating block decrement
Le	empty

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success
'65'	‘81’	memory failure (unsuccessful decrement)
‘69’	‘81’	incompatible command
‘69’	‘82’	security status not satisfied
‘69’	‘86’	command not allowed
‘6A’	‘81’	function not supported
‘6A’	‘82’	invalid block address

4.1.3 Mifare Application Directory (MAD)

To access the Mifare Application Directory (MAD), two commands are necessary – Authenticate and Read. The following steps describe how a MAD can be retrieved from a Mifare card:

1. Authenticate block 3 with the Public key ‘A0A1A2A3A4A5’ and authentication mode A.
2. Read Block 3.
3. Read Block 2.
4. Read Block 1.

4.2 iCLASS Card

HID iCLASS cards can only be accessed through OMNIKEY's proprietary "scardsyn" API. This synchronous API contains a function that is dedicated to accessing contactless cards using the PC/SC card handle.

OMNIKEY CardMan 5x21-CL exposes all iClass functions necessary to access any of the application areas on an iClass card. Two modes of communication between the card and the application are supported:

- Standard mode communication
- Secured mode communication (OMNIKEY proprietary mode)

*Note: OMNIKEY CardMan 5x21 does not allow **WRITE access to the HID application** (1st application on page 0). For **READ access to the HID application**, secured communication (available for firmware version 5.00 and greater) is mandatory*.

4.2.1 Card Access via SCardCLICCTransmit

SCardCLICCTransmit is the OMNIKEY proprietary function to access HID iCLASS cards via the OMNIKEY synchronous API. It supports both, standard and secure communication modes and is defined as follows:

```
OKERR ENTRY SCardCLICCTransmit (
      IN SCARDHANDLE ulHandleCard,
      IN PUCHAR      pucSendData,
      IN ULONG       ulSendDataBufLen,
      IN OUT PUCHAR  pucReceivedData,
      IN OUT PULONG  pulReceivedDataBufLen );
```

Parameter	Description
ulHandleCard	handle to the card, provided from the PC/SC "smart card resource manager" after connecting to the card with SCardConnect
pucSendData	buffer for data sent to the reader/card, typically a command APDU
ulSendDataBufLen	length of the data to be sent
pucReceivedData	buffer for data received from reader/card, typically data and status
pulReceivedDataBufLen	before the call: length (in bytes) of the receive buffer after the call: number of bytes actually received

Command Syntax

CLA	INS	P1	P2	Lc	Input Data or Datagram***	Le
‘8x’	‘xx’	‘xx’	‘xx’	‘xx’	‘xx’ ... ‘xx’ (Lc bytes)	‘xx’

Response Syntax

Response Data or Datagram***	SW1	SW2
‘xx’ .. ‘xx’ (Le or max bytes)	‘xx’	‘xx’

Status Codes

SW1	SW2	Description
'90'	'00'	success
'64'	'00'	card execution error
'67'	'00'	wrong length
'68'	'00'	invalid class (CLA) byte
'69'	'82'	security status not satisfied. This can include wrong data structure, wrong keys, incorrect padding.
'6A'	'81'	invalid instruction (INS) byte
'6B'	'00'	wrong parameter P1 or P2

The error codes defined in the table above are valid for all the commands. Command specific error codes are documented with their respective command documentation.

Note: Error code ‘6982’ “security status not satisfied”, received during any secured communication, blocks any further commands and requires that the card be removed and re-inserted to reactivate communication with the card.

5 CardMan 5x21-CL Keys

OMNIKEY CardMan 5x21-CL has a set of built-in cryptographic keys, some of which are implemented in volatile memory others in non-volatile memory.

5.1 Key Numbering Scheme

Cryptographic keys are referenced by a unique key number between 0x00 and 0xFE. Each key number refers to a key of pre-defined length for a specific card type. For cards such as Mifare and iCLASS, multiple key numbers are reserved.

The OMNIKEY key number is used to determine key usage, key length, and to map the reader key to the third party card key.

Examples:

CardMan Key number '0A' refers to the 6 byte Mifare key 10, K_{MIF10}

CardMan Key number '24' refers to the 8 byte iCLASS Default key for application 1 on page 1

Refer to [MIFARE] and [ICLASS] for detailed documentation of these third-party keys and contact your card manufacturer in case you need information about any key values.

Keys Numbers and Key Names

Key Number	Key Name	Key Length	Key Type	Memory Type
	6-byte (Mifare) keys			
'00' to '1F'	K_{MIF0} (Mifare Key 0) to K_{MIF31} (Mifare Key 31)	6 bytes	Card Key	Non-volatile memory
	8-byte (iClass) keys			
'20'	K_{IAMC} (Any Inside Application Master key)	8 bytes	Card Key	Non-volatile memory
'21'	K_{MDC} HID Master Key (K_{MD0}, Kd for application 1 of page 0 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
'22'	RFU (previously used for HID Master Key K_{MDO})	8 bytes	Card Key	Non-volatile memory
'23'	K_{MC0} (Default Master Key for application 2 of page 0 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
'24'	K_{MD1} (Default Master Key for application 1 of page 1 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
'25'	K_{MC1} (Default Master Key for application 2 of page 1 of iCLASS card)	8 bytes	Card Key	Non-volatile memory

‘26’	K_{MD2} (Default Master Key for application 1 of page 2 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘27’	K_{MC2} (Default Master Key for application 2 of page 2 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘28’	K_{MD3} (Default Master Key for application 1 of page 3 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘29’	K_{MC3} (Default Master Key for application 2 of page 3 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘2A’	K_{MD4} (Default Master Key for application 1 of page 4 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘2B’	K_{MC4} (Default Master Key for application 2) of page 4 of iCLASS card	8 bytes	Card Key	Non-volatile memory
‘2C’	K_{MD5} (Default Master Key for application 1 of page 5 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘2D’	K_{MC5} (Default Master Key for application 2 of page 5 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘2E’	K_{MD6} (Default Master Key for application 1 of page 6 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘2F’	K_{MC6} (Default Master Key for application 2 of page 6 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘30’	K_{MD7} (Default Master Key for application 1 of page 7 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘31’	K_{MC7} (Default Master Key for application 2 of page 7 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
‘32’	K_{MTD} (Master Transport Key for application 1 of iCLASS card, key stored at chip production)	8 bytes	Card Key	Non volatile memory
‘33’	K_{MTC} (Master Transport Key for application 1 of iCLASS card, key stored at chip production))	8 bytes	Card Key	Non-volatile memory
‘34’..’7f’	RFU			
16-byte keys				
‘80’	K_{CUR} (Custom read key)	16 bytes	Reader Key	Non-volatile memory
‘81’	K_{CUW} (Custom write Key)	16 bytes	Reader Key	Non-volatile memory

‘82’	K_{ENC} (Card data encryption key)	16 bytes	Card Key	Non-volatile memory
24- byte keys				
‘B0’..’CF’	RFU			
32-byte keys				
‘D0’..’DF’	RFU			
0xF0 to 0xFF are volatile keys				
0xF0	K_{VAK} (volatile application key)	8 bytes	Card Key	Volatile memory
‘F1’...‘FF’	RFU			

Note: CardMan 5x21 firmware version 5.00 is the first to support all keys listed above. Readers with firmware version 1.03 and 1.04 only support key numbers 0x20 and 0xF0.

Key number 0x21 to Key number 0x31 (except 0x22) are the default keys for iCLASS cards. Key number 0x32 and 0x33 are the default transport keys for Inside cards.
Keys 0x21 and 0x22 are stored in the reader. The remaining non-volatile keys 0x23 to 0x33 are stored in the registry.

Key 0x21 can’t be updated. Updates of key 0x22 are RFU and currently not supported.

5.2 Key Container and Key Slots

The CardMan 5x21-CL key container is organized in fixed-length key slots. These key slots allow easy usage of cryptographic keys. It is thus not necessary that the host application knows anything about the physical storage location. Keys can be loaded into a key container just by referring to a key slot and a key number. Key access and usage are managed by the reader firmware. For security purposes, keys can only be used and updated, but they can never be read. As an additional security measure, keys are diversified with two 16-byte secret keys before being committed to a key container.

Key slot properties are available for advanced users. This feature is designed to ensure proper use of a single key in case there are more keys than key slots.

Key Container of CardMan 5x21-CL Reader

Key Slot (KS) Number	KS Length	Default Stored Key Name	Default Stored Key Number	Remarks
‘00’	**12**	K_{MIF0}	**’00’**	No key slot information is available for these key slots. Retrieving information will return SW1SW2 “6300”.
….	**12**	**-------**	**----**	
‘1F’	**12**	K_{MIF31}	**’1F’**	
‘20’	**16**	K_{CUR}	**’80’**	Key slot information is available.
’21’	**16**	K_{CUW}	**’81’**	
’22’	**16**	K_{ENC}	**’82’**	
’23’	**08**	K_{IAMC}	**’20’**	
’24’	**08**	K_{MDO}	**’22’**	

’25’	**08**	**K_{MDC}**	**’21’**	
’26’	**08**	**K_{VAK}**	**’F0’**	No key slot information is available for these key slots. Retrieving information will return SW1SW2 “6300”.
’27’	**08**	**K_{MC0}**	**’23’**	Key slot information is available.
’28’	**08**	**K_{MD1}**	**’24’**	
’29’	**08**	**K_{MC1}**	**’25’**	
’2A’	**08**	**K_{MD2}**	**’26’**	
’2B’	**08**	**K_{MC2}**	**’27’**	
’2C’	**08**	**K_{MD3}**	**’28’**	
’2D’	**08**	**K_{MC3}**	**’29’**	
’2E’	**08**	**K_{MD4}**	**’2A’**	
’2F’	**08**	**K_{MC4}**	**’2B’**	
’30’	**08**	**K_{MD5}**	**’2C’**	
’31’	**08**	**K_{MC5}**	**’2D’**	
’32’	**08**	**K_{MD6}**	**’2E’**	
’33’	**08**	**K_{MC6}**	**’2F’**	
’34’	**08**	**K_{MD7}**	**’30’**	
’35’	**08**	**K_{MC7}**	**’31’**	
’36’	**08**	**K_{MTD}**	**’32**	
’37’	**08**	**K_{MTC}**	**’33’**	

5.3 Key Update Rules

The following table lists update rules for keys being used by the reader system. Key updates relate to keys residing in the OMNIKEY reader. Those keys are used for authentication of the reader to the card or to encrypt data written to the card.

Key Name	**Key Number**	**Key Update Rule**	**Description**
K_{MIF0} to **K_{MIF31}**	’00’ to ‘1F’	Always	6-byte Mifare keys can be loaded/updated by using the ***SCardCLWriteMifareKeyToReader*** function of synchronous API. A key sent to reader may be plain or 3-DES encrypted with the K_{CUR} or K_{CUW}
K_{IAMC}	‘20’	Standard Mode: - Always Secured Mode: - Read session - Write session	8-byte iCLASS key to authenticate any iCLASS application. The default value for this key is the Inside contactless card transport key Kd0 (authenticates to application 1 on page 0).
K_{MDC}	’21’	Never	Authenticates the reader to the HID application of an iCLASS card for read access. This authentication

			requires secure mode operation. Write access to the HID application is not allowed.
K_{MDO}	’22’	Never	RFU
K_{CUR}	’80’	Secured mode: - read session - write session	Authenticates the reader to establish a secured session. Grants the application read access. This key can also be used to encrypt the Mifare key in ***SCardCLWriteMifareKeyToReader function***.
K_{CUW}	’81’	Secured mode: - read session	Authenticates the reader to establish a secured session. Grants the application read-only access. This key can also be used to encrypt the Mifare key in ***SCardCLWriteMifareKeyToReader function***.
K_{ENC}	’82’	Secured mode: - read session - write session	Encrypts data written to the card or decrypts data read from the card. Requires read/update INS bits to be set accordingly. If INS bits are set for DES, the first 8 bytes of **K_{ENC}** are used. For 3-DES operations, all 16 bytes are used.
K_{VAK}	’F0’	Standard Mode: - Always Secured Mode: - Read session - Write session	Authenticates any application on the iCLASS card. The sequence is as follows: Load K_{VAK} with the 8-byte value, Authenticate with K_{VAK} …………………………………………………………… Load K_{VAK} with new 8-byte value, Authenticate with $K_{VAK.}$
K_{MC0} **to** **K_{MC7}** **K_{MD1}** to **$K_{MD7,}$**	’23‘ to ’31’	Never	iCLASS default keys for free memory zones. May be used to authenticate to any non-HID application on an iCLASS card. This allows quick evaluation of iCLASS cards without knowledge of the default keys.
K_{MTD} - **$K_{MTC,}$**	’32’ ’33’	Never	iCLASS transport keys set by the card manufacturer.

6 Standard Communication with iCLASS Card

Standard communication means that there is no authentication of the host application (i.e. Windows program) to the CardMan 5x21-CL. Unless the card itself has a built-in mechanisms for confidential communication, the channel between host and reader is unprotected, exposing the connecting USB cable to eavesdropping.

6.1 APDU Structure for Standard Communication

iCLASS cards are supported via ISO7816 compliant APDU exchange. Command and response APDUs are exchanged through the OMNIKEY proprietary API function SCardCLICCTransmit residing in the OMNIKEY synchronous API.

Command APDU (via pucSendData)

CLA	INS	P1	P2	Lc	Data in	Le
‘80’	‘xx’	‘xx’	‘xx’	‘xx’	‘xx’ … ‘xx’	‘xx’

Response APDU (via pucReceivedData)

Data out	SW2	SW1
‘xx’ … ‘xx’	‘xx’	‘xx’

6.2 Commands Available in Standard Communication Mode

Card commands are referred to by their respective instruction (INS) byte as part of a command APDU sent by SCardCLICCTransmit. The following table lists all INS values supported by the OMNIKEY CardMan 5x21-CL reader in standard communication mode.

List of Supported INS bytes (APDU Commend Set)

Instruction (INS)	Description	Command Type
‘82’	Load Key	reader command
‘C4’	GetKeySlotInfo	reader command
‘A6’	Select Page	card command
‘88’	Authenticate	card command
‘B0’	Read	card command
‘D6’	Update	card command

6.2.1 Select Page (Card Command)

iCLASS comes with various card configurations. Every iCLASS card has at least one page (page 0). Cards such as the iCLASS 2x8KS, provide additional pages 1 to 7. In addition to pages, iCLASS cards also have books. To select a certain memory block on an iCLASS card, you need to know its book number, page number, and block number.

It is necessary to select the appropriate page and book before any authentication to an iCLASS card application for read/write access can be performed. In the context of iCLASS cards, an application area and memory area are synonymous.

Currently, only cards with more than 16 kbit of total memory capacity have an additional book. The following section describes parameters of the Select Page command.

Command Syntax

CLA	‘80’
INS	‘A6’
P1	‘00’: Select the only page of iCLASS 2KS or single page of 16KS ’01’: Select page of multi-page iCLASS 16KS (8x2KS) or 32KS
P2	Specifies whether data is requested from the card ’00’: no data requested ’04’: request for 8-byte card serial number ’08’: request for 8-byte configuration block data ’0C’: request for 8-byte application issuer data

LC	for P1=’00’: standard mode: empty; secured mode: ‘00’
	for P1=’01’: ‘01’

Data Field	for P1=’00’: empty for P1=’01’: book number and page number according to format below

Le	for P2=’00’: empty
	for P2>’00’: ’00’ or ‘08’

Data Field Format for Page Number & Book Selection

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	Book number 0: for 1^{st} book 1: for 2^{nd} book on iCLASS 32KS	0	Page number 0-7		

Examples for Page Selection:

Data Field	Description
‘03’	select page 3 of an iCLASS 8x2KS card
‘03’	select page 3 of book 0 of an iCLASS 32KS (book 0: 8x2KS) card
‘13’	select page 3 of book 1 of an iCLASS 32KS (book 1: 8x2KS) card
‘10’	select book 1 (16KS) of an iCLASS 32KS
‘00’	select book 0 (16KS) of an iCLASS 32KS

Response Syntax

Data Field		empty or 8 byte card response, in case of a previous request for such data
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘62’	‘83’	requested page number does not exist
‘6C’	‘xx’	wrong length Le. xx returns the number of data available

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Note: If the application resides on page 0 of an 8x2KS iCLASS card or on the single page of an iCLASS 16KS or iCLASS 2KS card, the Select Page command is not necessary. It can still be helpful to call Select Page anyway, in case you need to retrieve the card serial number, configuration block, or application issuer data.

6.2.2 Load Key

Load Key command loads an iCLASS card key and stores it in reader memory, thus preparing the reader for subsequent card authentication commands. CardMan 5x21 can only store one such key at a time.

Command Syntax

CLA	‘80’: standard mode operation ’84’: secured mode operation
INS	‘82’
P1	‘xx’ specifies key location according to byte format below
P2	‘xx’ key number (see Key Numbering Scheme)
LC	‘08’
Data Field	8 byte key
Le	empty

P1 - Format for Key Location

b7	b6	b5	b4	b3	b2	b1	b0	Description
x								0: card key 1: reader key
	x							0: plain transmission 1: secured transmission (not available)
		x						0: key loaded in volatile memory 1: key loaded in non-volatile memory.
			x					0: RFU (non-zero value returns error)
				0	0	0	0	b0..b3 must be set to 0

Note: A key in volatile memory only needs to be loaded once during any given card session. Unless you need to authenticate to any additional application with a different key, you can use the stored key throughout the session for more than just one authentication.

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘63’	‘00’	no further information given (warning)
‘63’	‘81’	loading/updating is not allowed
‘63’	‘82’	card key not supported
‘63’	‘83’	reader key not supported
‘63’	‘84’	plaintext transmission not supported
‘63’	‘85’	secured transmission not supported
‘63’	‘86’	volatile memory is not available
‘63’	‘87’	non-volatile memory is not available
‘63’	‘88’	key number not valid
‘63’	‘89’	key length is not correct

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

6.2.3 GetKeySlotInfo (Reader Command)

The GetKeySlotInfo reader command provides access to key slot status information.

OMNIKEY CardMan 5x21-CL provides a set of predefined key slots in the key container. Key slots can easily be loaded with keys by referring to the key number (i.e. key reference) rather than loading the actual 8 byte key by value. The slot for key storage is automatically determined by the reader system.

Command Syntax

CLA	‘80’: standard mode operation ’84’: secured mode operation
INS	‘C4’
P1	‘00’
P2	‘xx’ key slot number (see chapter 5.2 Key Container and Key Slots)
LC	standard mode: empty; secured mode: ‘00’
Data Field	8 byte key
Le	‘00’ or ‘02’

Response Syntax

Data Field		2 byte key information see “Key Information” and “Key Access Option” below
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘63’	‘00’	no further information given (warning)
‘63’	‘01’	key slot does not contain valid key or empty key slot
‘62’	‘83’	requested key slot does not exist
‘6C’	‘xx’	more data available than requested; xx returns available data size

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Key Information (contained in Data Field)

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
RFU						Key Access Option		key number according to 5.1-Key Numbering Scheme ’FF’ means “empty key slot”							

Key Access Option (contained in b9,b8 of Data Field)

b9	**b8**	**Key Access Option**
0	0	key can be loaded for any plaintext and secured transmission.
0	1	key can only be loaded in OMNIKEY proprietary secured mode
1	0	key can never be loaded

1	1	RFU

6.2.4 Authenticate (Card Command)

The Authenticate command authenticates the reader system to the card application of the selected page. For iCLASS authentication, this command requires previous page selection.

Command Syntax

CLA	‘80’: standard mode operation ’84’: secured mode operation
INS	‘88’
P1	‘xx’ key type: ’00’: Inside Contactless or iCLASS debit key Kd (i.e. application 1) ’01’: Inside Contactless or iCLASS credit key Kc (i.e. application 2) ’60’: Mifare Key A ’61’: Mifare Key B ’FF’: key type unknown or not necessary all other values: RFU
P2	‘xx’ key number (see chapter 5.1-Key Numbering Scheme)
LC	length of address iCLASS: standard mode: empty; secured mode: ‘00’ other cards: ‘01’ or ‘02’ (max 2 address bytes supported)
Data Field	iCLASS: empty other cards: one or two byte address
Le	empty

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘63’	‘00’	no further information given (warning)
‘69’	‘83’	authentication cannot be done
‘69’	‘84’	reference key not useable
‘69’	‘88’	key number not valid

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

6.2.5 Read (Card Command)

The Read command reads a data block from the given block address. For the iCLASS card, only eight bytes can be read at a time. For further information about available blocks refer to [HID_ICLASS]. This command requires previous page selection and, depending on the iCLASS card configuration, authentication to the iCLASS application.

Command Syntax

CLA	‘80’: standard mode operation ’84’: secured mode operation
INS	‘B0’
P1	MSB of block number
P2	LSB of block number
LC	standard mode: empty; secured mode: ‘00’
Data Field	empty
Le	‘00’ or ‘08’ ’20’: if supported by card, up to 32 bytes can be returned

Response Syntax

Data Field		8 byte block returned from the card (iCLASS) 32 bytes returned if card supports it
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘62’	‘81’	part of returned data may be corrupted
‘62’	‘82’	end of file reached before reading all requested bytes
‘69’	‘81’	command incompatible
‘69’	‘86’	command not allowed
‘6A’	‘81’	function not supported
‘6A’	‘82’	file not found or addressed block or byte does not exist
‘6C’	‘xx’	more data available than requested; xx returns available data size, typically ‘08’

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Note: Reading blocks without valid authentication or trying to read data without read permission, will set all returned data to ‘FF’.

6.2.6 Update (Card Command)

The Update command writes a data block to a given block address. For the iCLASS card, only eight bytes can be written at a time. For further information about available blocks refer to [HID_ICLASS]. This command requires previous page selection and, depending on the iCLASS card configuration, authentication to the iCLASS application.

Command Syntax

CLA	‘80’: standard mode operation ’84’: secured mode operation
INS	‘D6’
P1	MSB of block number
P2	LSB of block number
LC	‘08’ (iCLASS only allows 8 bytes per call)
Data Field	8 bytes to be written to card
Le	empty

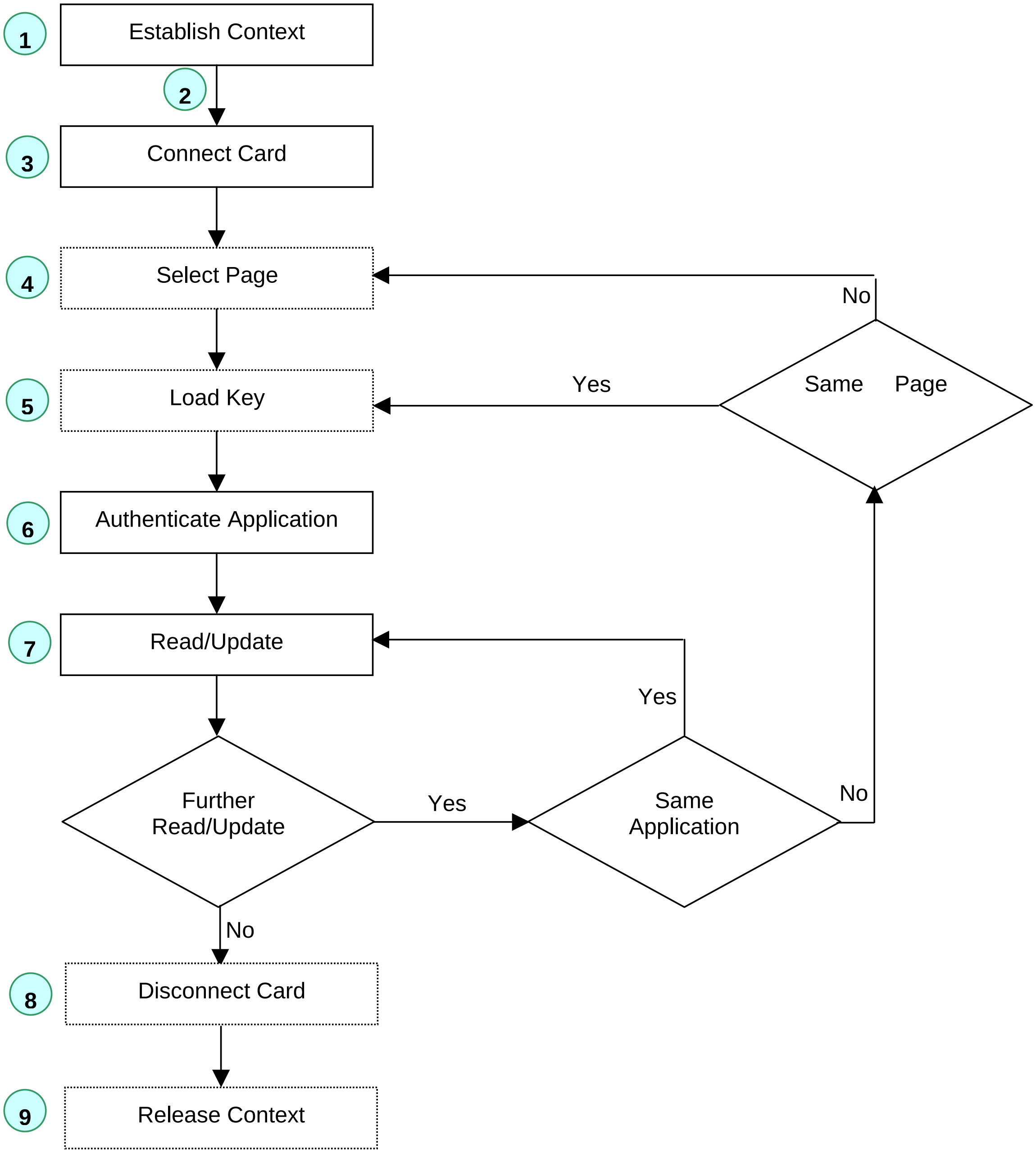
Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success
‘62’	‘82’	end of file reached before writing all Lc bytes
‘65’	‘81’	memory failure (unsuccessful writing).
‘69’	‘81’	command incompatible
‘69’	‘86’	command not allowed
‘6A’	‘81’	function not supported
‘6A’	‘82’	file not found or addressed block or byte does not exist

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Note: Trying to update without authenticating to the corresponding application will return ‘6400‘ “Card Execution Error”.

6.3 Communication in Standard Mode



7 Secured Communication with the iCLASS Card

For a desktop smart card reader, such as the OMNIKEY CardMan 5x21-CL, security mainly evolves from the following scenarios:

- Authenticity between the host application and the reader
- Confidentiality of data transmitted via USB cable
- Integrity of transmitted data
- Authenticity between the reader and the card
- Confidentiality and integrity of the RF transmission
- Confidentiality of data stored in cards

OMNIKEY CardMan 5x21-CL reader provides an end-to-end security scheme to fulfill the security requirements listed above.

Note: Secured mode communication requires reader firmware version 5.00 or greater.

7.1 Multi-Step Approach to a Secure Card Reader System

7.1.1 Authenticity Between Host and Reader

Authenticity between host and reader is enforced with a mutual authentication scheme that requires a 16-byte transport key (Kcur or Kcuw) and a proprietary algorithm. Sessions can only be initiated upon successful completion of this one-step mutual authentication process.

Note: This feature prevents unauthorized reader usage. Additional information about this process is available to under an NDA.

7.1.2 Confidentiality of USB Data Exchange

CardMan 5x21-CL has a built-in mechanism that protects against eavesdropping and replay attacks on USB traffic. The data transmitted through a USB cable is triple DES encrypted with the Session Key (Ks). This key is generated during the mutual authentication process. It is unique for every session. Therefore, traffic recorded in one session cannot be replayed in another session.

7.1.3 Integrity of Transmitted Data

Data transmitted between host and reader is digitally signed with an eight-byte Message Authentication Code (MAC) which is appended to the data. This is done to detect any inconsistencies that may occur due to erroneous or modified data.

7.1.4 Authenticity Between Reader and Card

iCLASS cards allow authentication of the reader system to the card. This is done by proving knowledge of a shared secret, the iCLASS card application key $\mathbf{K_{IAMC}}$ or $\mathbf{K_{MDC}}$. Applications that are protected with such a key require successful reader authentication before read/write access to card data can be granted.

7.1.5 Integrity of the Radio Frequency (RF) Transmission

Data integrity of an RF transmission with an iCLASS card is enforced with a two-byte CRC.

7.1.6 Confidentiality of the RF Transmission

The CardMan 5x21-CL supports an important feature to guarantee confidentiality: it encrypts data before writing data to the card and decrypts data read from the card. Confidentiality in this context means that data can be securely transmitted between the card and the reader without an eavesdropper being able to read the data in plaintext.

7.1.7 Authentication of the Host for Read/Write Session

CardMan 5x21-CL contains two keys K_{CUR} and K_{CUW} that are used to control access to read and write functions respectively. Initiating a reader session with K_{CUR} makes it a read-only session thus blocking functions that write to the card. Starting a session with K_{CUW} enables the reader for both read and write access.

Note: This is part of a host-to-reader authentication mechanism, not to be confused with reader-to-card authentication enforced by the card itself.

7.1.8 Protection Against Known Attacks

Replay Attacks:

> The data header contains a datagram that is different with every APDU exchange. The reader ensures that no frame is repeated.

Plain Text Attack:

> For some critical commands, there is a built-in delay to prevent a plain text attack. If there is any error in the data header or signature, the session is immediately terminated. One can commence communication only again after starting a new session.

OMNIKEY welcomes comments and suggestions regarding any additional features that might be implemented to prevent specific attacks to the card system.

7.2 APDU Structure for Secured Communication

CardMan 5x21-CL provides a unique mechanism to secure the communication channel using OMNIKEY’s proprietary cryptographic envelope which protects the transmitted data from eavesdroppers.

Secured communication requires additional steps to prepare data before sending it to the reader system and after receiving data from the reader. The underlying triple DES algorithm requires a block size that is a multiple of 8. Therefore, the datagram has a built-in padding scheme. Authenticity of the plaintext is enforced with an 8 byte signature.

Command Syntax

CLA	INS	P1	P2	Lc	Input Datagram (*sent* to the reader)	Le
‘84’	‘xx’	‘xx’	‘xx’	‘xx’	‘xx ... xx”	‘xx’

Input Datagram (sent to the reader)

	Data Header (DH)	Size of INS related data Lc_{INS}	INS related data (INSData)	Padding Bytes (PB)	Signature
3-DES{K_S, (	‘xxxxxxxx’	‘xx’	‘xx ... xx’	’80 ... 00’	‘xx ... xx’ **)}**
	4 bytes	1 byte	Lc_{INS} bytes	P bytes	8 bytes

P = number of padding bytes to satisfy (4+1+ Lc_{INS}+P) is multiple of 8.

Response Syntax

Output Datagram (*received* from the reader)	SW2	SW1
‘xx ... xx’	‘xx’	‘xx’

Output Datagram (received from the reader)

	Data Header (DH)	Size of Card Response LcR	Card Response	Padding Bytes (PB)	Signature
3-DES{K_S, (	‘xxxxxxxx’	‘xx’	‘xx ... xx’	’80 ... 00’	‘xx ... xx’ **)}**
	4 bytes	1 byte	n bytes	P bytes	8 bytes

P = number of padding bytes to satisfy (4+1+ Lc_{INS}+P) is multiple of 8.

Note: If no valid session key Ks is available due to a previous error during the ‘Start Session’ command, all datagram bytes are set to ‘00’. Therefore the host would receive ’00 ... 00’ || *SW1* || *SW2 as response from the reader.*

7.2.1 Data Header (DH)

Data Header

Byte 0	Byte 1	Byte 2	Byte 3
Host data header (HDH)		Reader data header (RDH)	

When the host system sends a Host Data Header (HDH) to the reader, the reader must acknowledge the HDH in its response by returning the 1's complement of the original HDH. This allows the host to check whether it receives data originating from the correct data header.

When the reader sends a Reader Data Header (RDH) to the host, the host must acknowledge the RDH in its next request by sending the 1's complement of the preceding RDH. This allows the reader to check whether the data sent by the host follows a previous reader response.

7.2.2 Signature Generation

The CardMan 5x21-CL signature generation is based on an 8-byte Message Authentication Code (MAC). The MAC value is calculated by taking the last 8 bytes of a DES CBC encypted data block consisting of DH, LcINSData, INSData, and padding bytes. Kcur or Kcuw are used as signing keys.

The following steps describe how padding is applied to create a data block that can be signed using a DES CBC operation:

- Append '80' to the right of the data block.
- If the resulting data block length is a multiple of eight, no further padding is required.
- Do zero ('00') padding until the data block size reaches a multiple of eight.

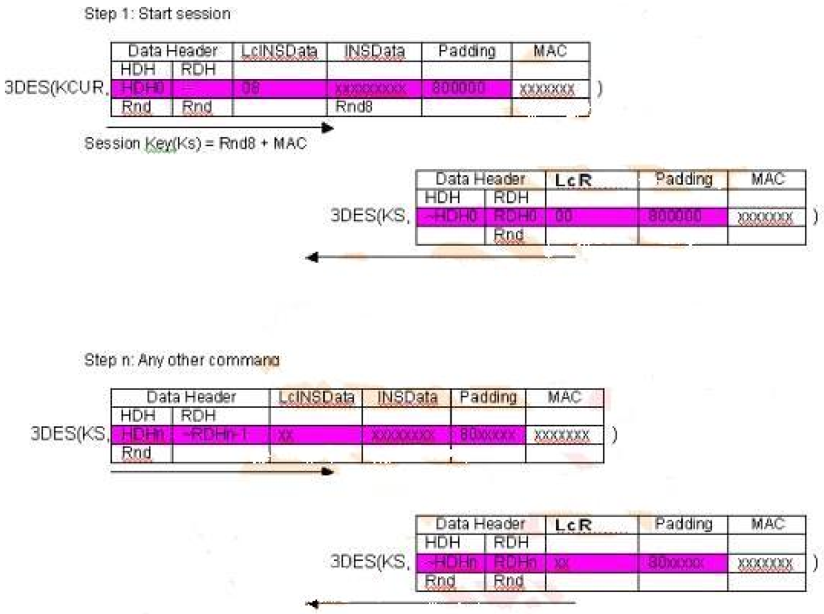
7.2.3 Session Key Generation

The session key Ks is derived from an 8-byte random number and the MAC transmitted to the reader during Start Session. For the Start Session command, LcINSData equals 8 (length of the random number) and INSData contains the 8-byte random number.

All secured communication calls following a successful session key negotiation are 3DES encrypted with Ks.

7.2.4 Example: Proprietary Datagram Exchange between Host and Reader

Host **Reader**



Step 1: Start session

3DES(KCUR,

Data Header		LcINSData	INSData	Padding	MAC
HDH	RDH				
HDH0	--	08	xxxxxxxxx	800000	xxxxxxx
Rnd	Rnd		Rnd8		

)

Session Key(Ks) = Rnd8 + MAC

3DES(KS,

Data Header		LcR	Padding	MAC
HDH	RDH			
~HDH0	RDH0	00	800000	xxxxxxx
	Rnd			

)

Step n: Any other command

3DES(KS,

Data Header		LcINSData	INSData	Padding	MAC
HDH	RDH				
HDHn	~RDHn-1	xx	xxxxxxxx	80xxxxx	xxxxxxx
Rnd					

)

3DES(KS,

Data Header		LcR	Padding	MAC
HDH	RDH			
~HDHn	RDHn	xx	80xxxxx	xxxxxxx
Rnd	Rnd			

)

Figure: Proprietary data structure scenario

Note: This is a read-only session because K_{CUR} was used in the start session command. If K_{CUW} were used to start the session, both read and write operations would be allowed. The HID application is always read-only.

7.3 Instructions (INS) for Secured Communication

Card commands are referred to by their respective instruction (INS) byte as part of a command APDU sent by SCardCLICCTransmit. CardMan 5x21-CL with firmware version 5.00 or greater supports the following secured mode instructions:

List of INS bytes for Secured Communication

Instruction (INS)	Description	Command Type
‘C4’	GetKeySlotInfo	reader command
‘72’	Manage Session	reader command
‘82’	Load Key	reader command
‘A6’	Select Page	card command
‘88’	Authenticate	card command
‘B0’	Read	card command
‘D6’	Update	card command

In the following sections the command structure is described. LcINS and INSData are part of the OMNIKEY proprietary structure.

Notes
1- cured mode’ and ‘Standard Mode’ use different formatting of P1, bit 7 and bit 6 of the Read/Update commands (INS 0xB0 and 0xD6 respectively). The two LSBits of P1 are used to control the encryption of data read or updated.
2- Lc must always be transmitted in secured mode.

Manage Session (Reader Command)

The Manage Session command is used to either start a session or end a session.

Command Syntax

CLA	‘84’	
INS	‘72’	
P1	‘00’: start session ’01’: end session other values: RFU	
P2	P1 = ‘00’ (start session)	P1 = ’01’ (end session)
	‘00’: start read only session ’01’: start read/write session	‘00’
Lc	‘08’: challenge size	‘00’
Data Field	8-byte random number (challenge)	empty
Le	empty	

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
‘90’	‘00’	success

Please refer to 4.2.1-Card Access via SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Note: A session will be automatically ended if the card is removed.

7.3.1 Select Page (Card Command)

Except for the CLA byte ‘84’, the syntax for Select Page in secured mode is identical to the command described in 6.2.1-Select Page (Card Command).

7.3.2 Load Key (Reader Command)

Except for the CLA byte ‘84’, the syntax for Load Key in secured mode is identical to the Load Command described in 6.2.2-Load Key.

7.3.3 Authenticate (Card Command)

Except for the CLA byte ‘84’, the syntax for Authenticate in secured mode is identical to the command described in 6.2.4-Authenticate (Card Command).

7.3.4 Read (Card Command)

Except for the CLA byte ‘84’, and the additional formatting rules for P1 described below, the syntax for the Read command in secured mode is identical to the command described in 6.2.5-Read (Card Command).

P1 Formatting for Secured Mode

b7	b6	b5 – b0	Description
0	0	Block Nr. MSB	Plain
0	1		DES Encryption
1	0		Triple DES Encryption
1	1		RFU

Data needs to be decrypted with the K_{ENC} to get the plaintext data.

7.3.5 Update (Card Command)

Except for the CLA byte ‘84’, and additional formatting of P1 described below, the syntax for the Update command in secured mode is identical with the command described in 6.2.6-Update (Card Command).

P1 Formatting for Secured Mode

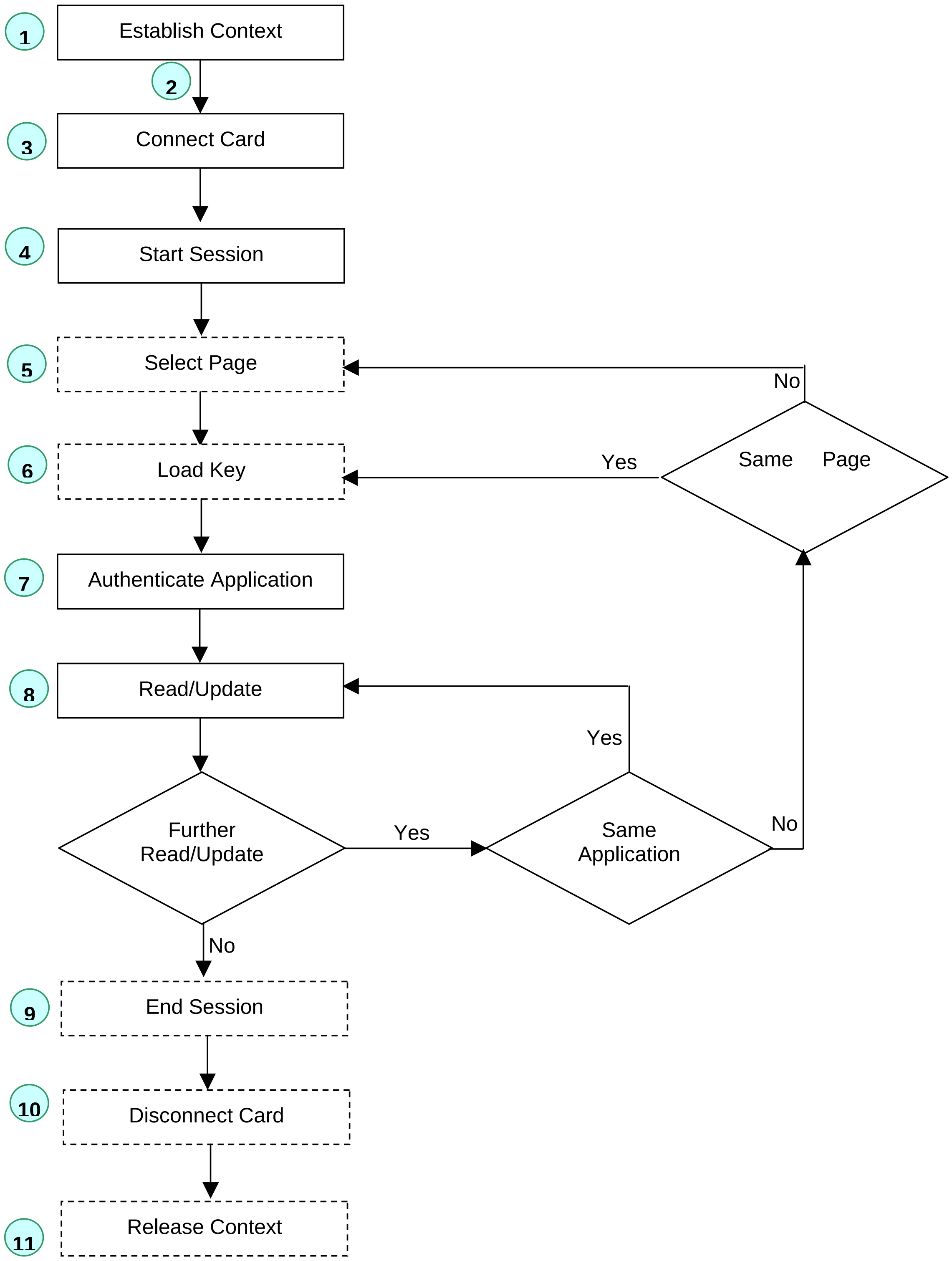
b7	b6	b5 – b0	Description
0	0	Block Nr. MSB	Plain
0	1		DES Encryption
1	0		Triple DES Encryption
1	1		RFU

Data is encrypted with K_{ENC} before storing it on the card.

7.3.6 GetKeySlotInfo (Reader Command)

Except for the CLA byte ‘84’, the syntax for 7.3.7 GetKeySlotInfo in secured mode is identical to the command described in 6.2.3-GetKeySlotInfo (Reader Command).

7.4 Communication at Secured Mode



7.5 Example APDUs for a Session at Secured Mode

K_{CUR} = ‘A0A1A2A3A4A5A6A7A8A9AAABACADAEAF’, Read-only session

Host **Reader**

1. Start Session

CLA	INS	P1	P2	Lc	OMNIKEY Proprietary Input Datagram (sent to reader) CLEAR					
‘84’	‘72’	‘00’	‘00’	‘18’	‘1422’	‘9D2B’	‘08’	‘4A895F20C2D30B5E’	‘800000’	‘9E5052819C5A8D3C’
					HDH (Rnd)	RDH (Rnd)	LcINS	Rnd8 (INSData)	Padding	Signature
					DH					MAC
					‘FD274CE840FA9AD139E4FC2923653A88743CB5986DB4F7A0’					
					OMNIKEY Proprietary Input datagram (sent to reader) ENCIPHERED					

Signature = DESEn {(A0A1A2A3A4A5A6A7),(14229D2B084A895F20C2D30B5E800000)}

= 8A8D430D608714FE9E5052819C5A8D3C

9E5052819C5A8D3C (last eight bytes of DES encryption)

Enciphered datagram = 3-DESEn{
(A0A1A2A3A4A5A6A7A8A9AAABACADAEAF),
(14229D2B084A895F20C2D30B5E8000009E5052819C5A8D3C) }

= FD274CE840FA9AD139E4FC2923653A88743CB5986DB4F7A0 (24 byte input datagram)

SessionKey (K_S) = Rnd8 + MAC = 4A895F20C2D30B5E9E5052819C5A8D3C

OMNIKEY Proprietary Output Datagram (received from reader)					SW1SW2
A04B84A4DE515FD8A9D40DFFE703FBF1					9000
‘EBDD’	E00C	00	800000	E367401E2DA8FACB	
~HDH	RDH(Rnd)	LcR	Padding	Signature	
DH				MAC	

3-DESDec{(4A895F20C2D30B5E9E5052819C5A8D3C),(A04B84A4DE515FD8A9D40DFFE703FBF1) }

= EBDDE00C00800000E367401E2DA8FACB

Signature = DESEn{(4A895F20C2D30B5E),(EBDDE00C00800000) }

= E367401E2DA8FACB

Note: An open source library to accomplish all security protocols introduced in the secured communication mode is available from OMNIKEY upon request.

2. Authenticate HID Application

CLA	INS	P1	P2	Lc	OMNIKEY Proprietary Send Datagram				
84	88	00	21	10	B3F1	1FF3	00	800000	B50318C9E871191A
					HDH (Rnd)	~RDH	LcINS	Padding	Signature
					DH				MAC
					B5FD83E756CA03DE54FBEA5546E8867D				
					Proprietary Data				

Signature = DESEn{(4A895F20C2D30B5E),(B3F11FF300800000)}
= B50318C9E871191A

Proprietary Data = 3-DESEn{(4A895F20C2D30B5E9E5052819C5A8D3C),(B3F11FF300800000B50318C9E871191A) }
= B5FD83E756CA03DE54FBEA5546E8867D

OMNIKEY Proprietary Response Datagram					SW1SW2
78A10C4FCC7EBC2C516354A56C4C7818					9000
4C0E	7D55	00	800000	D2D0B0B4E34EBDBE	
~HDH	RDH(Rnd)	LcR	Padding	Signature	
DH				MAC	

3-DESDec{(4A895F20C2D30B5E9E5052819C5A8D3C), (78A10C4FCC7EBC2C516354A56C4C7818) }
= 4C0E7D5500800000D2D0B0B4E34EBDBE

Signature = DESEn{(4A895F20C2D30B5E),(4C0E7D5500800000) }
= D2D0B0B4E34EBDBE

Note: An open source library to accomplish all security protocols introduced in the secured communication mode is available from OMNIKEY upon request.

3. Read Block 6

CLA	INS	P1	P2	Lc	OMNIKEY Proprietary Send Datagram					Le
84	B0	00	06	10	6762	82AA	00	800000	F63AB82BED09B039	08
					HDH (Rnd)	~RDH	LcINS	Padding	Signature	
					DH				MAC	
					2FABB8F0533E742383F4FE9045142859					
					Proprietary Data					

Signature = DESEn{(4A895F20C2D30B5E),(676282AA00800000)}

= F63AB82BED09B039

Proprietary Data = 3-DESEn{(4A895F20C2D30B5E9E5052819C5A8D3C),(676282AA00800000F63AB82BED09B039) }

= 2FABB8F0533E742383F4FE9045142859

OMNIKEY Proprietary Response Datagram						SW1SW2
AA401E3D849B881044FF4D847977D9070C589338C097F163						9000
989D	2A94	08	000000000000E414	800000	3101DDB971C922FF	
~HDH	RDH(Rnd)	LcR	Response Data	Padding	Signature	
DH					MAC	

3-DESDec {(4A895F20C2D30B5E9E5052819C5A8D3C),
(AA401E3D849B881044FF4D847977D9070C589338C097F163)}

= 989D2A9408000000000000E4148000003101DDB971C922FF

Signature = DESEn{(4A895F20C2D30B5E),(989D2A9408000000000000E414800000) }

= 1CDF21DCA31BABDB3101DDB971C922FF

= 3101DDB971C922FF (last 8-byte block)

Note: An open source library to accomplish all security protocols introduced in the secured communication mode is available from OMNIKEY upon request.

Appendix A Application Programming

A1 Sample Project

The following C++ sample project is part of the synchronous API which can be downloaded from our website at www.omnikey.com

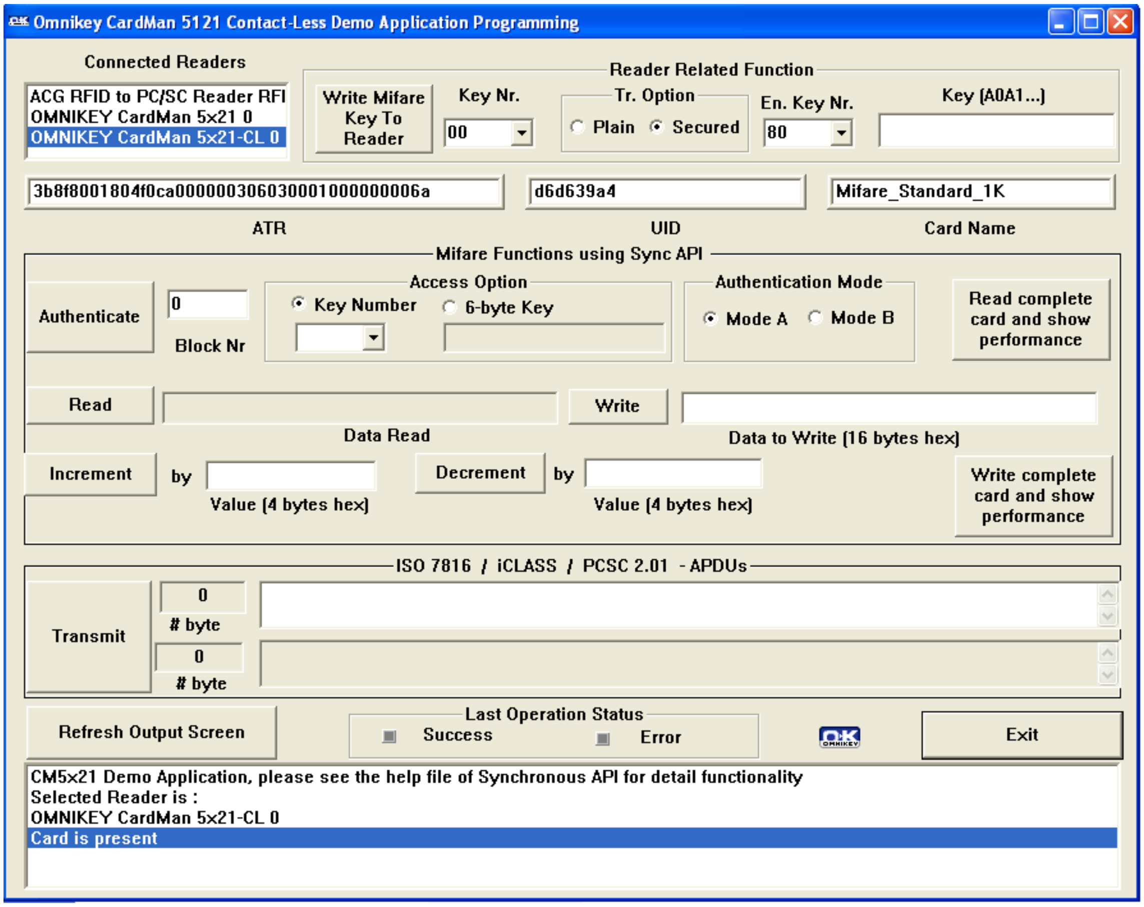
If you choose the default installation settings, sample code can be found in: c:\omnikey\samples\contactlessdemovc.

Sample code for Visual Basic is also available and can be found in c:\omnikey\samples\contactlessdemovb.

The sample uses the OMNIKEY synchronous API and demonstrates how to select a reader, connect to a card, and access either a Mifare or an iCLASS card.

Note: Mifare cards can also be integrated via non-proprietary, PC/SC 2.0 compliant function calls.

A1.1 Overview


Omnikey CardMan 5121 Contact-Less Demo Application Programming

Connected Readers:
ACG RFID to PC/SC Reader RFI
OMNIKEY CardMan 5x21 0
OMNIKEY CardMan 5x21-CL 0

Key Nr.: 00
Tr. Option: ○ Plain ● Secured
En. Key Nr.: 80

ATR: 3b8f8001804f0ca0000003060300010000000006a
UID: d6d639a4
Card Name: Mifare_Standard_1K

Block Nr: 0
Access Option: ● Key Number ○ 6-byte Key
Authentication Mode: ● Mode A ○ Mode B

Last Operation Status: ▨ Success ▨ Error

CM5x21 Demo Application, please see the help file of Synchronous API for detail functionality
Selected Reader is :
OMNIKEY CardMan 5x21-CL 0
Card is present

Reader Related Function
Write Mifare Key To Reader
Key (A0A1...)
Mifare Functions using Sync API
Authenticate
Read complete card and show performance
Read
Data Read
Write
Data to Write (16 bytes hex)
Increment
by
Value (4 bytes hex)
Decrement
by
Value (4 bytes hex)
Write complete card and show performance
ISO 7816 / iCLASS / PCSC 2.01 - APDUs
Transmit
0
# byte
0
# byte
Refresh Output Screen
Exit

Figure 1: Screenshot of Sample Program

In the list box in the top-left corner of the window you can select the reader from a drop down list that contains all readers available to the smart card resource manager. When a card is inserted, the **ATR**, **UID** and **Card Name** is displayed. The functions in the **Reader Related functions** group box can be used with or without a card in the RF field.

Functions available in the **Mifare Functions using Sync API** group can only be used when a Mifare card is in the field. The **ISO 7816/iCLASS/PCSC 2.01** group allows APDU exchange with a CPU card (asynchronous card) in the field.

Each processed command produces output in the output log on the bottom of the window. This log can be cleared with the **Refresh Output Screen** button. The return status of the last function executed is shown in the group box **Last Operation Status.**

The **Exit** button closes the application.

A1.2 Reader Related Functions

Reader related functions don't require a card in the field.

To store a MIFare key, do the following:

- Define a key number to determine the location where the key will be stored.
- Select either plain or secured as the mode of transmission of the key. For secured transmissions, use transmission key number 0x80 or 0x81.
- Enter the key in hex string format to the text field **Mifare Key**. For plain transmissions enter a 6 byte, 12 hex digit value (no spaces). For secured transmission enter an 8 byte value.
- Click on the **Write Mifare Key to Reader** button to load the key to reader memory.

A1.3 Mifare Card Related Functions Using Synchronous API

Before using any of the **Mifare Card Related functions** authentication to the card is required. (Mifare UltraLight does not need authentication).

To authenticate to a block of the card do the following:

- In the field **Block Nr***,* enter the block number of the block you want to authenticate to.
- In the **Access Option** box choose whether a key number or a plain key will be supplied.
- In the **Authentication Mode** box choose **Mode A** or **Mode B**.
- Press the **Authenticate** button.

Upon successful authentication, you can read and write data blocks and use the increment and decrement functions.

A1.4 PC/SC 2.01

Enter an APDU according to PC/SC 2.01 to access storage cards such as Mifare cards directly without using the OMNIKEY proprietary synchronous API.

A1.5 ISO 7816 - APDU

Enter an APDU for your CPU (asynchronous) card and send the APDU the same way you would an ISO7816 contact card.

A1.6 iCLASS Standard Mode

Present an iCLASS card to the reader RF field, and send APDUs directly to the card according to the *chapter 6-Standard Communication with iCLASS Card*. This will give you an easy-to-use way of experimenting with the available functions.

A2 Code Snippets

This section lists some coding samples for a PC/SC 2.01 compliant implementation.

A2.1 Getting the Card UID (PC/SC 2.01)

The following function retrieves the Unique card ID (UID) of the card currently connected via air interface. You can use the UID as the card serial number. It is available for every ISO 14443 A/B or ISO 15693 compliant card and does not matter whether it is a CPU or storage card. This makes GetUID the ideal candidate for Hello Card type applications. If you don’t have access to application keys, the UID may serve as a valuable identifier that allows card lookup on a backend database.

```
BOOLEAN GetUID(UCHAR *UID, int &sizeofUID)
{
  ucByteSend[0] = 0xFF;//CLA
  ucByteSend[1] = 0xCA;//INS
  ucByteSend[2] = 0x00;//P1
  ucByteSend[3] = 0x00;//P2
  ucByteSend[4] = 0x00;//Le
  ulnByteSend = 5;
  printf("\nRetrieving the UID..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                               ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
            ucByteReceive[dwRecvLength-2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  sizeofUID = dwRecvLength-2;
  memcpy(UID,ucByteReceive,sizeofUID);
  return TRUE;
}
```

A2.2 Loading a Mifare Key (PC/SC 2.01)

The following code-snippet loads a Mifare key to the reader. The key is stored in non-volatile memory. Once loaded, it remains available throughout the reader session.

```
BOOLEAN LoadKey(UCHAR ucKeyNr, UCHAR *ucKey, UCHAR ucKeyLength)
{
  ucByteSend[0] = 0xFF;       //CLA
  ucByteSend[1] = 0x82;       //INS
  ucByteSend[2] = 0x20;       //P1 card key, plain transmission, non-volatile memory
  ucByteSend[3] = ucKeyNr;    //P2 key number for mifare could be 0x00 to 0x31)
  ucByteSend[4] = ucKeyLength;//Lc
  memcpy(ucByteSend+5,ucKey, ucKeyLength );
  ulnByteSend = 5+ucKeyLength;
  printf("\nLoading Key to the reader..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                               ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
    ucByteReceive[dwRecvLength -2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  return TRUE;
}
```

A2.3 Mifare 1K/4K Authenticate (PC/SC 2.01)

The following code snippet demonstrates how to authenticate to a Mifare card:

```
BOOLEAN Authenticate(UCHAR BlockNr, UCHAR ucKeyNr, UCHAR ucKeyType)
{
  ucByteSend[0] = 0xFF;      // CLA
  ucByteSend[1] = 0x88;      // INS
  ucByteSend[2] = 0x00;      // P1, Mifare Block Number MSB, for mifare it is always 0x00
  ucByteSend[3] = BlockNr;   // Mifare Block Number LSB
  ucByteSend[4] = ucKeyType; // P3
  ucByteSend[5] = ucKeyNr;
  ulnByteSend = 6;
  printf("\nAuthenticating ..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                             ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
            ucByteReceive[dwRecvLength-2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  return TRUE;
}
```

A2.4 Mifare 1K/4K Write (PC/SC 2.01)

```
BOOLEAN UpdateBinary(UCHAR BlockNr, UCHAR *ucDataToWrite, UCHAR ucDataLenght)
{
  ucByteSend[0] = 0xFF;//CLA
  ucByteSend[1] = 0xD6;//INS
  ucByteSend[2] = 0x00;//P1, Mifare Block Number MSB, for mifare it is always 0x00
  ucByteSend[3] = BlockNr;//Mifare Block Number LSB
  ucByteSend[4] = ucDataLenght;
  memcpy(ucByteSend+5,ucDataToWrite, ucDataLenght);
  ulnByteSend = 5+ucDataLenght;
  printf("\nUpdating Block ..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                             ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
            ucByteReceive[dwRecvLength-2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  return TRUE;
}
```

A2.5 Mifare 1K/4K Read (PC/SC 2.01)

```
BOOLEAN ReadBinary(UCHAR BlockNr, UCHAR *ucDataRead, UCHAR &ucDataLenght)
{
  ucByteSend[0] = 0xFF;//CLA
  ucByteSend[1] = 0xB0;//INS
  ucByteSend[2] = 0x00;//P1, Mifare Block Number MSB, for mifare it is always 0x00
  ucByteSend[3] = BlockNr;//Mifare Block Number LSB
  ucByteSend[4] = 0x10;//Le
  ulnByteSend = 5;
  dwRecvLength = 255;
  printf("\nReading Block ..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                               ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
           ucByteReceive[dwRecvLength-2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  ucDataLenght = (unsigned char)dwRecvLength -2;
  memcpy(ucDataRead,ucByteReceive,ucDataLenght);
  return TRUE;
}
```

A2.6 Mifare 1K/4K Increment (OMNIKEY Proprietary API)

```
BOOLEAN Increment(UCHAR BlockNr, UCHAR *ucDataTobeIncremented, UCHAR ucDataLenght)
{
  ucByteSend[0] = 0xFF;//CLA
  ucByteSend[1] = 0xD4;//INS
  ucByteSend[2] = 0x00;//P1, Mifare Block Number MSB, for mifare it is always 0x00
  ucByteSend[3] = BlockNr;//Mifare Block Number LSB
  ucByteSend[4] = ucDataLenght;
  memcpy(ucByteSend+5,ucDataTobeIncremented, ucDataLenght);
  ulnByteSend = 5+ucDataLenght;
  printf("\nIncrementing Block ..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                               ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
           ucByteReceive[dwRecvLength-2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
  }
  return TRUE;
}
```

A2.7 Mifare 1K/4K Decrement (OMNIKEY Proprietary API)

```
BOOLEAN Decrement(UCHAR BlockNr, UCHAR *ucDataTobeDecremented, UCHAR ucDataLenght)
{
  ucByteSend[0] = 0xFF;//CLA
  ucByteSend[1] = 0xD8;//INS
  ucByteSend[2] = 0x00;//P1, Mifare Block Number MSB, for mifare it is always 0x00
  ucByteSend[3] = BlockNr;//Mifare Block Number LSB
  ucByteSend[4] = ucDataLenght;
  memcpy(ucByteSend+5,ucDataTobeDecremented, ucDataLenght);
  ulnByteSend = 5+ucDataLenght;
  printf("\nDecrementing Block ..........");
  SCard_Status = SCardTransmit(hCard,SCARD_PCI_T1,ucByteSend,ulnByteSend,NULL,
                               ucByteReceive, &dwRecvLength);
  if (SCard_Status != SCARD_S_SUCCESS)
  {
    printf("\nProblem in SCardTransmit, Erro rcode = 0x%04X",SCard_Status);
    return FALSE;
  }
  if(ucByteReceive[dwRecvLength-2] != 0x90 || ucByteReceive[dwRecvLength-1] != 0x00)
  {
    printf("\nWrong return code: %02X%02X",
           ucByteReceive[dwRecvLength-   2],ucByteReceive[dwRecvLength-1]);
    return FALSE;
}
  return TRUE;
}
```

A2.8 iCLASS Select Page (OMNIKEY Proprietary API)

The following code-snippet selects page 0x01 of a 8x2KS iCLASS card and returns the card serial number:

```
//Select page 0x02 of a 8x2KS iCLASS card
UCHAR ucDataSend[7] = {0};
ULONG ulNoOfDataSend = 7;
UCHAR ucReceivedData[64] = {0};
ULONG ulNoOfDataReceived = 64;

ucDataSend [0]  = 0x80 //CLA, standard mode
ucDataSend [1]  = 0xA6 //INS
ucDataSend [2]  = 0x01 //P1
ucDataSend [3]  = 0x04 //P2, return card serial number
ucDataSend [4]  = 0x01 //Lc
ucDataSend [5]  = 0x01 //Page number
ucDataSend [6]  = 0x08 //Le

SCard_Status = SCardCLICCTransmit(hCard,ucDataSend,ulNoOfDataSend,
                                  ucReceivedData,&ulNoOfDataReceived);
if(SCard_Status!= SCARD_S_SUCCESS)
{
  printf("Error in SCardCLICCTransmit, with error code %8X", SCard_Status);
  exit(-1);
}
```

Appendix B Accessing iCLASS Memory

In the following diagrams the free zones of two typical iCLASS memory layouts is shown:

B1.1 Memory Layout

a) Memory layout of an iCLASS 2KS, iCLASS 16KS or page 0 of an iCLASS 8x2KS card:

Block Number	**Block Description (block size eight bytes)**
‘00’	card serial number
‘01’	configuration block
’02’	e-Purse
‘03’	Kd (so-called debit key, key for application 1)
’04’	Kc (so-called credit key, key for Application 2)
’05’	application issuer area
‘06’	HID application
….	
’12’	
’13’	Free zones in iCLASS 2KS, iCLASS 16KS or page 0 of iCLASS 8x2KS
….	
‘1F’(2KS) ‘FF’(16KS)	

b) Memory layout of an iCLASS 8x2KS on pages 1 to 7:

Block	Size: 8 bytes
’00’	card serial number
’01’	configuration block
’02’	e-Purse
’03’	Kd (so-called debit key, key for application 1)
’04’	Kc (so-called credit key, key for Application 2)
’05’	application issuer area
’06’	application 1 (free zones in iCLASS 8x2KS other than page 0)
….	
‘xx’	
‘xx’+1	application 2 (free zones in iCLASS 8x2KS other than page 0)
….	
‘1F’	

B1.2 Assigning Space to iCLASS Application 2

In the iCLASS default card, the application limit ‘xx’ is set the end of the memory area. This means that the complete memory area is reserved for application 1. The application limit can be set to a different block number to support an additional application. To do this, the page’s configuration block must be overwritten as follows:

1. Select the page you want to configure.
2. Authenticate with Kd of the selected page.
3. Read 8 bytes from block 0x01 – the configuration block.
4. Replace the first byte with the block number ‘xx’ of the new application limit.
5. Leave the remaining bytes of the configuration block unchanged and write all 8 bytes back to the configuration block 0x01.
6. Remove the card.

B1.3 Read/Write Memory of iCLASS 2KS, 16KS or page 0 of iCLASS 8x2KS card

1. Insert card.
2. Connect to card.
3. For secured mode: Start Session.
4. Authenticate with K_{MC0} , (P1 = 0x01, P2 = 0x23).
   If the key is not an iCLASS default key, the new key has to be loaded as K_{IAMC} or $K_{VAK\ ,}$ and in the authenticate command the key number of K_{IAMC} or K_{VAK} must be used.
5. Read/write any block (block number 0x13 to 0x1F for 2KS and 0xFF for 16KS).
6. For secured mode: End Session.
7. Disconnect from card.
8. Remove card.

B1.4 Read/Write Memory of iCLASS 8x2KS Card on Pages 1 to 7

1. Insert card.
2. Connect to card.
3. For secured mode: Start Session.
4. Select page N (N = 1 to 7).
5. Authenticate with K_{MDN} / K_{MCN} (P1 = 0x00 for $K_{MDN\ ,}$or 0x01 for K_{MCN} , P2 = K_{MDN} ,/ K_{MCN} (refer to chapter 5.1Key Numbering Scheme).
6. If the key is other than iCLASS default key, the new key has to be loaded as K_{IAMC} or $K_{VAK\ ,}$ and in the authenticate command the key number of K_{IAMC} or K_{VAK} must be used.

7. Read/write any block (block number 0x13 to 0x1F for 2KS and 0xFF for 16KS).
8. For secured mode: End Session.
9. Disconnect card.
10. Remove card.

Appendix C

C1.1 Terms and Abbreviations

The following table lists abbreviations used throughout this document.

CSNR	Card Serial Number
HDH	Host Data Header
INSData	Instruction Specific Data
K_{CUR}	Customer Read Key
K_{CUW}	Customer Write Key
K_{DOKM}	Omnikey Diversified Master Key
K_{ENC}	Card Data Encryption Key
K_{IAMC}	Any Application Master Key
K_{MCN}	Page N Application 2’s Master Key of iCLASS card
K_{MDC}	HID Master Key Current
K_{MDN}	Page N Application 1’s Master Key of iCLASS card
K_{MDO}	HID Master Key Old
K_{MTD}	ICLASS Master Transport key for application 1
K_{MTC}	ICLASS Master Transport key for application 2
K_{OKM}	OMNIKEY Master Key
K_{S}	Session Key
K_{VAK}	Any Volatile Application Master Key
Lc_{INS}	Instruction specific data (INSData) length.
LcR	Card Response data length
RDH	Reader Data Header
RSNR	Reader Serial Number

Appendix D Version History

D1.1 Document Changes

Version	***Author(s)***	***Date***	***Description***
1.00	Abu Ismail	Feb 08, 2005	Initial Version
1.01	Abu Ismail	June 30, 2006	Reorganization, adding PC/SC 2.01 support
1.10	Marc Jacquinot	Aug 18, 2006	FW 5.00, secured communication, finalized document for release
1.11	Marc Jacquinot	Nov 22, 2006	added notes: mandatory Lc in secured mode

D1.2 Firmware History

FW Version	Special Feature(s)	Remarks
1.00	MIF	Mifare support
1.01, 1.02	MIF, MKS	
1.03,1.04	MIF, MKS, IST	iCLASS memory access
5.00	MIF, MKS, IST, ISE	iClass secured mode, HID application read

Special Feature of Synchronous Cards:

MIF = MIFARE Functionalities

MKS = MIFARE Key Storage

MSK = MIFARE Secured Key Loading

IST = iCLASS Standard Mode Communication

ISE = iCLASS Secured Mode Communication

Appendix E References

[MIFARE] MIFARE Data Sheets

http://www.semiconductors.philips.com/markets/identification/datasheets/index.html - mifare

[MSDNLIB] Microsoft Developer Network Library; http://msdn.microsoft.com/library/

[DESFIRE] DESFire Data Sheets

http://www.semiconductors.philips.com/acrobat_download/other/identification/M075031.pdf

[PCSC_2.01] PC/SC Workgroup Specifications 2.01

http://www.pcscworkgroup.com/

[PICO16KS] PICOTAG and PICOCRYPT secured 16KS data sheet from the Inside Contactless

[PICO2KS] PICOTAG and PICOCRYPT secured 2KS data sheet from the Inside Contactless

[ICLASSD] iCLASS card specifications from HID.

[ISO7816-4] Information Technology Identification Cards Integrated Circuit(s) Cards with Contacts, Part 4: Inter-industry Commands for Interchange