15370 Barranca Parkway
Irvine, CA 92618-2215

Contactless Smart Card Readers

DEVELOPER GUIDE

January 11, 2010

Doc Number: 5321-903, Rev A.1.20

Contents

Trademarks

HID, HID Global and OMNIKEY are the trademarks or registered trademarks of HID Global Corporation in the U.S. and other countries.

MIFARE® is a registered trademark of NXP Semiconductors
my-d ™ is a registered trademark of Infineon Technologies

Contacts

OMNIKEY brand product support
HID Global GmbH email: eusupport@hidglobal.com Fax: +49 (0) 6123 7913-28 web: http://www.hidglobal.com/omnikeyCustomerSupportForm.php http://www.hidglobal.com/omnikey

Purpose

Guide for developers for integrating contactless storage or CPU cards using OMNIKEY CardMan 5x21 and 6x21 smart card readers.

1 Contactless Reader Coverage

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

The following OMNIKEY contactless readers are covered by this document:

- OMNIKEY 5321
  Desktop Smart Card reader with contact and contactless interface, contactless interface featuring full contactless functionality as described in this developers guide.
- OMNIKEY 5321 CL
  Desktop reader in a closed housing, same functionality as OMNIKEY 5321 but contactless-only reader.
- OMNIKEY 5321 CR
  Desktop reader in a waterproof (Clean Room) closed housing, same functionality as OMNIKEY 5321 but contactless-only reader.
- OMNIKEY 6321
  Mobile Smart Card reader with SIM-sized contact and contactless interface. Contactless interface features full contactless functionality.
- OMNIKEY 6321 CLi
  Mobile Smart Card reader with contactless-only interface. Contactless interface supports iCLASS-only.
- OMNIKEY 5321 CLi
- Desktop Smart Card reader in a closed housing, with contactless-only interface. Contactless interface supports iCLASS-only.
- OMNIKEY 5325 Prox
  Desktop Smart Card reader with contact and contactless interface. Contactless interface features operating on 125 kHz (Prox). The PC/SC section of this guide applies for this reader.

All readers listed are based on the OMNIKEY 5x21 RFID chipset. Therefore this document will use the term **5x21** to reference OMNIKEY readers.

2 Getting Started

This chapter describes how to install the drivers necessary to operate the OMNIKEY 5x21 in a Windows based environment.

Note: Other operating systems, such as Linux, are also supported by the OMNIKEY 5x21.

2.1 Driver Installation

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

OMNIKEY 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 OMNIKEY 5x21 driver is necessary.

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

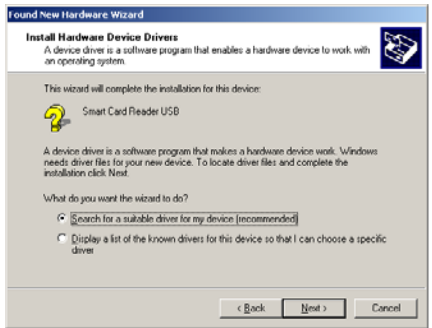
1. First, go to http://www.hidglobal.com/omnikey. Based on the appropriate reader, click the driver icon. Download the latest OMNIKEY 5x21 driver installation package for Windows.
2. Run the installation package and follow the instructions. The installation package extracts all the necessary driver files to your hard drive.

   Take note of the location to which the files were copied.

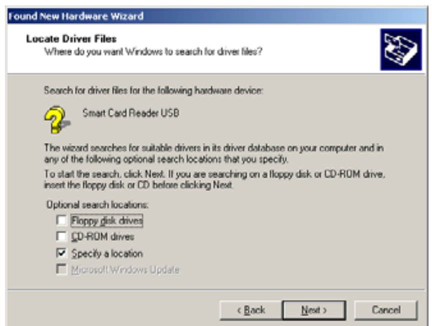
   At this time you have only extracted, not installed the driver files.
3. Connect the reader to your computers USB port.
4. The **Found New Hardware Wizard** appears. To continue driver installation, click **Next.**

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, replace the Microsoft PC/SC driver manually with the OMNIKEY proprietary PC/SC driver using the Device Manager.

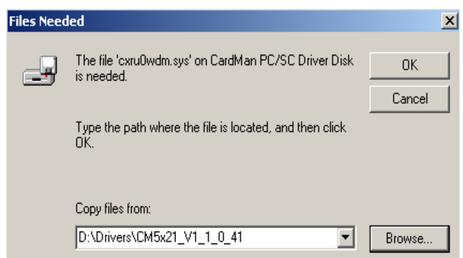
5. Select **Search for a suitable driver for my device (recommended)** and click **Next**.



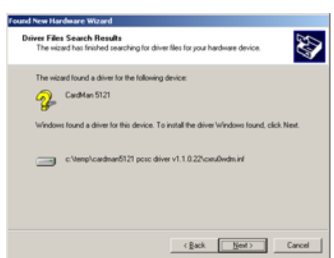
6. Then, select **Specify a Location** and click **Next**.



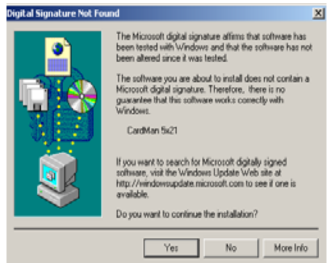
7. Click **Browse** and go to the location where you previously installed the driver package. To continue, click **OK**.



8. If the driver was found, click **Next**.



9. If the driver is a beta driver and not digitally signed, the following dialogue appears. Click **Yes**.



10. The following message appears and the green LED illuminates on the OMNIKEY 5x21 reader.

If the installation was successful, the green LED on the reader illuminates and the reader is listed in the diagnostic tool as OMNIKEY 5x21.

Your reader is ready for use. Do a quick smart card system check using the OMNIKEY Diagnostic Tool described in Diagnostic Tool, page 8.

2.1.1 Reader Name for Contact/Contactless Slot

OMNIKEY 5x21 is a dual slot reader. This means that from the application and smart card resource manager viewpoint there are two readers available, each represented by its respective reader name. **OMNIKEY 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 through the contact and air interface.

2.2 Diagnostic Tool

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

Go to http://www.hidglobal.com/omnikey > select the OMNIKEY Reader > click the driver icon to download the latest OMNIKEY Diagnostic Tool for Windows.

Start former versions of the Diagnostic Tool from the **Control Panel**.

2.2.1 Driver Version Detection

The **General** tab shows if the **Resource Manager** is running. In addition, this tab shows smart card system services version, manufacturer data, DLLs, and drivers.

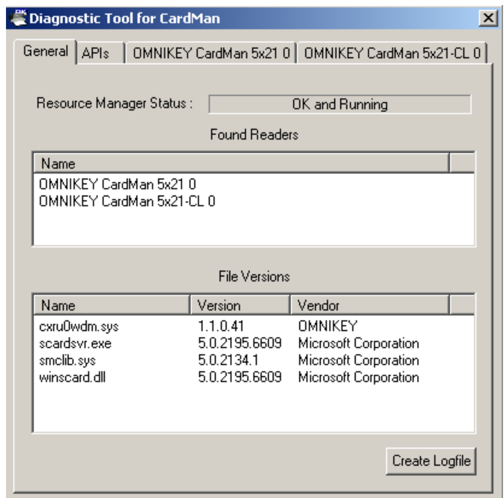


Figure 1 - Diagnostic Tool - General

2.2.2 OMNIKEY Proprietary API Detection

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

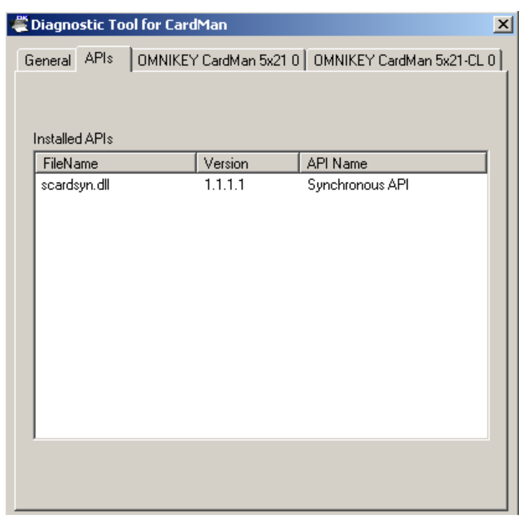


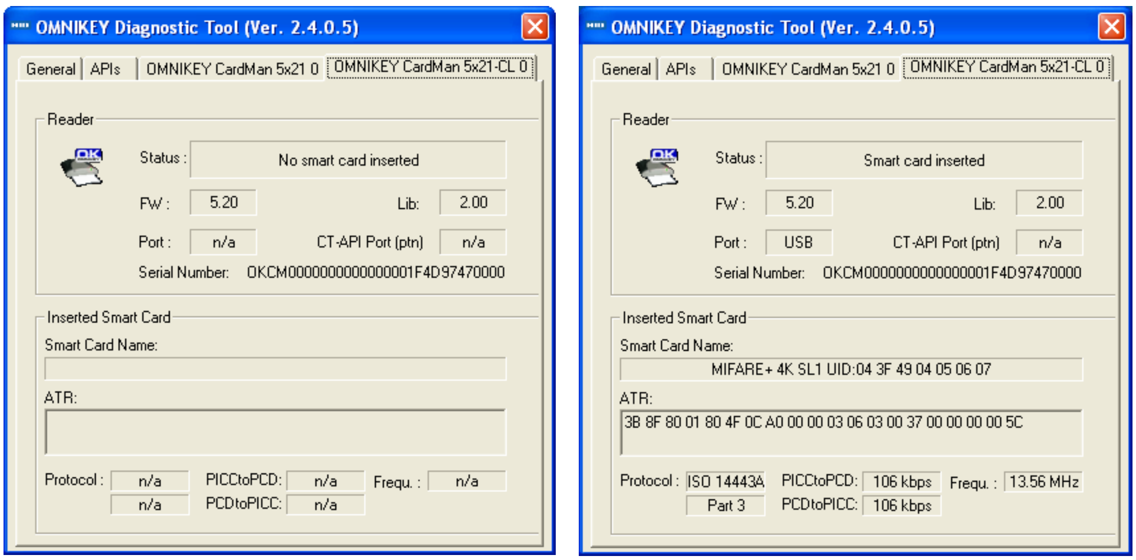
Figure 2 - Diagnostic Tool - API

2.2.3 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.



No smart card | **Smart card inserted**

Figure 3 – Diagnostic Toll - Reader

2.2.4 Card Type Detection and RFID Settings

CardMan 5x21-CL supports multiple 13.56 MHz contactless standards and protocols including ISO14443A, ISO14443B, ISO15694, iCLASS, I-CODE. Acquire information about a card within the RFID field in a predefined search order. With built-in anti-collision, once a card is detected it is the only card in which the reader is connected.

The OMNIKEY Diagnostic tool has a **RFID Settings** tab that allows configuration of the reader card and their respective search order. First enable the **RFID Settings** tab by right-clicking the title bar of the Diagnostic Tool window and then choosing **View > RFID settings** from the drop-down menu.

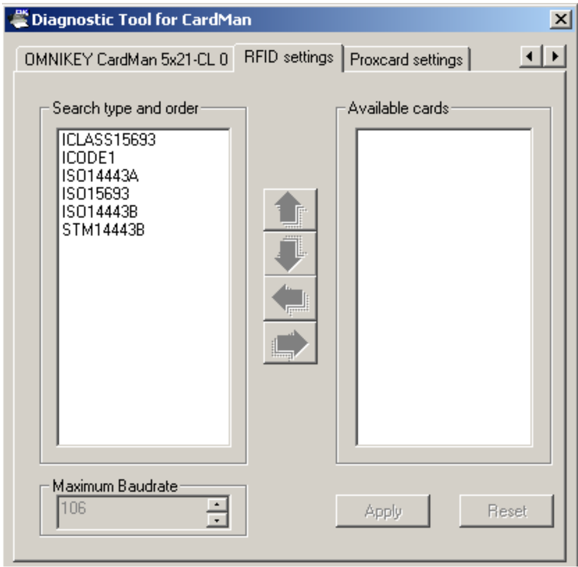


Figure 4 – Diagnostic Tool – RFID Settings

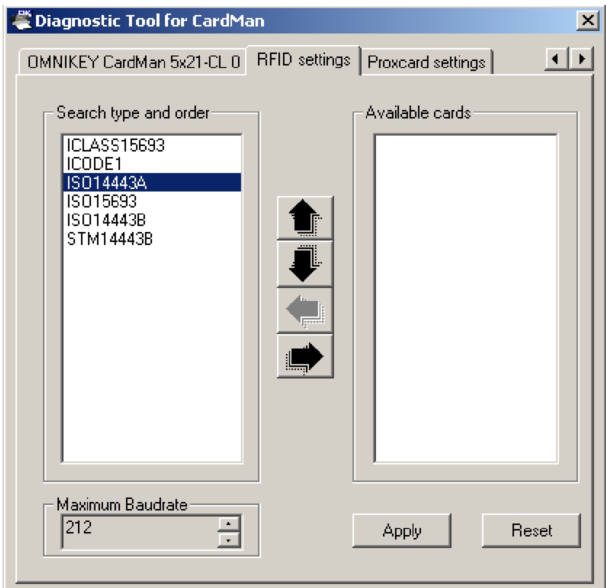
The left pane contains a list of active card types. The right pane contains a list of available card types that are supported by the reader but are not included in the card search. Move card types from the left to the right pane using the ◄ and ► buttons. Change the search order with the ▲ and ▼ buttons.

Activate this setting using the **Apply** button. The **Reset** button discards any unsaved changes.

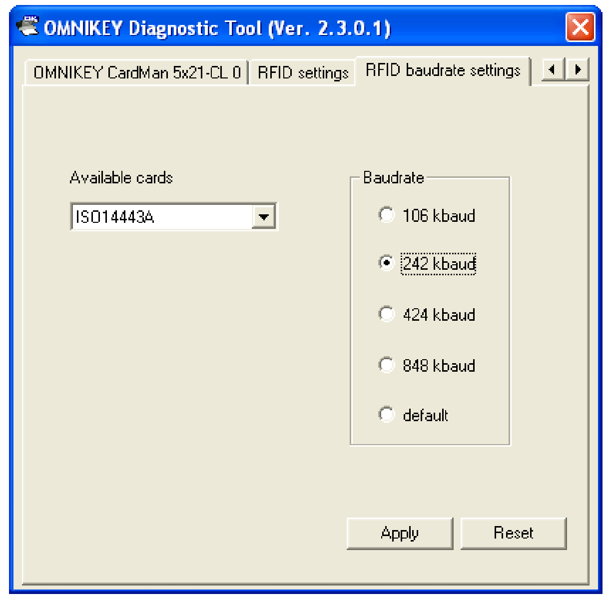
Note: The search order is forward-looking to improve system performance. The last successfully detected card type automatically moves to the top of the search order, regardless of its position within the order set on the **RFID settings** tab.

2.2.5 Air Interface Baud Rate Configuration

For ISO 14443 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. Change the interface transmission speed to a different value through the Diagnostic Tool **RFID settings** tab.



Baud Rate Change (old)



Baud Rate Change (new)

Figure 5 – 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, click **Apply**.

3 PC / SC 2.0

With the OMNIKEY 5x21 PC/SC driver, access ISO14443A/B or ISO15693 contactless cards through the same framework as ISO7816 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.

See http://msdn.microsoft.com/en-us/library/ms953432.aspx.

Access contactless CPU cards directly through PC/SC. For storage cards other than MIFARE, 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 are required to write your first **hello card** program.

	Integrate your card through:	
	PC/SC 2.0 compliant APDU's	**OMNIKEY Synchronous API**
MIFARE	YES	YES
iCLASS	NO	YES
LRI64	YES	NO

3.1 How to Access Contactless Cards through 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 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, refer directly to the MSDN Library. See http://msdn.microsoft.com/en-us/library/ms953432.aspx.

1. **Establish Context**

   This step initializes the PC/SC API and allocates all resources necessary for a smart card session. The **`SCardEstablishContext`** function establishes the resource manager context (scope) within which database operations is performed.

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

2. **Get Status Change**

   Check the status of the reader for card insertion, removal, or availability of the reader. This **`SCardGetStatusChange`** function blocks execution until the current availability of the cards in a specific set of readers change. The caller supplies a list of monitored readers and the maximum wait time (in milliseconds) 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**

   Gets a list of all PC/SC readers using the **`SCardListReaders`** function. Look for **OMNIKEY CardMan 5x21-CL 0** in the returned list. If multiple OMNIKEY 5x21 readers are connected to your system, they will be enumerated.

   Example: OMNIKEY CardMan 5x21-CL 1, and OMNIKEY CardMan 5x21-CL 2.

   Analyze the complete string. CardMan 5x21 also has a contact interface. Look for **-CL** in the reader name to ensure you are 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**

   Exchange command and data through APDUs. The **`SCardTransmit`** function sends a service request to the smart card, expecting 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 unsupported PC/SC 2.0 storage cards, call an OMNIKEY proprietary API function such as **`SCardCLICCTransmit`** instead. This function exposes additional functionality of the OMNIKEY 5x21-CL reader that is not yet defined in PC/SC standards. Otherwise, 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 all system resources are released. The **`SCardReleaseContext`** function closes an established resource manager context, freeing any resources allocated under that context.

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

3.2 ATR Generation

Unlike contact cards, contactless cards do not generate an ATR. Instead, they generate an Answer to Select (ATS). To make contactless cards available within the PC/SC framework, OMNIKEY 5x21 generates a PC/SC compliant ATR according to PC/SC v2.01.

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

3.2.1 CPU Cards

Contactless smart cards (cards with a CPU) expose their ATS or information bytes through ATR mapping according to PC/SC 2.01 - **Part 3: Requirements for PC-Connected Interface Devices, section 3.1.3.2.3.1 Contactless Smart Cards, Table 3.5**.

3.2.2 Storage Cards

The ATR of storage cards (for example, cards without a CPU) is composed as described in PC/SC 2.01 - **Part 3: Requirements for PC-Connected Interface Devices, section 3.1.3.2.3.2 Contactless Storage Cards, Table 3.6**. For the host application to identify a storage and card type properly, its standard and card name is mapped according to PC/SC 2.01 - **Part 3: Requirements for PC-Connected Interface Devices - Supplemental Document**.

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

4 Accessing Asynchronous Cards

Asynchronous cards contain a CPU or are memory cards accessible through standard PC/SC using Microsoft's library **winscard.dll**. This type of card 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.

4.1 MIFARE DESFire Card

MIFARE DESFire cards are accessed through ISO7816-4 compliant framed APDU commands (ISO7816-4 framing).

New versions of MIFARE DESFire cards (EV1) support extended APDU commands. For this the driver must switch to DESFire native mode. This native mode is not **default** for the OMNIKEY 5x21. For proper protocol settings use the following registry key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CardMan\RFID

DesfireNative=0x00000001

Note: Restart the OMNIKEY 5x21 driver after changing the registry key (disconnect and reconnect the reader).

4.1.1 Example: Write Card Data through 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 the MIFARE DESFire data sheet)

4.1.2 Example: Read Card Data through ISO 7816-4 Framed APDU

Command Syntax

CLA	INS	P1	P2	Lc	File No.	Offset	Length	Data	Le
‘90’	‘BD’	‘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 the MIFARE DESFire data sheet)

5 Accessing Synchronous Cards (Storage)

OMNIKEY provides two ways to integrate contactless storage cards. One option is OMNIKEY's proprietary synchronous API library, or for MIFARE cards, directly through PC/SC 2.0 compliant function calls. Access storage cards not supported through PC/SC 2.0 compliant APDU exchanges through OMNIKEY proprietary synchronous API.

The synchronous API for Windows systems resides in a DLL named **scardsyn.dll**. Download the Synchronous API for OMNIKEY 5x21 from www.hidglobal.com/omnikey and execute the setup **CardMan_Synchronous_API_V1_1_1_4.exe**. The setup include this DLL. The download also contains sample code for MIFARE and iCLASS cards. For 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 used whenever a card has not yet found its way into the PC/SC 2.0 standard. Currently, only MIFARE cards can be integrated through PC/SC 2.0 compliant APDU.

	Integrate Card through	
	PC/SC 2.0 compliant APDUs	**OMNIKEY Synchronous API**
MIFARE	Yes	Yes
iCLASS	No	Yes

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

5.1 MIFARE Card

OMNIKEY 5x21 supports MIFARE Mini, MIFARE 1K, MIFARE 4K and MIFARE Ultra Light cards.

The following functions are supported through PC/SC:

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
MIFARE Emulation Mode	OMNIKEY proprietary extension of PC/SC *CM_IOCTL_SET_RFID_CONTROL_FLAGS*

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.

5.1.1 MIFARE Increment (Card Command)

This command increments the value of a block, if the card and block supports 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

5.1.2 MIFARE Decrement (Card Command)

This command decrements 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

5.1.3 MIFARE Emulation Mode

By default, the OMNIKEY 5x21 driver exposes standard MIFARE storage cards through a PC/SC 2.01 compliant interface. This driver-level MIFARE emulation mode makes standard MIFARE cards available through standard APDUs even though the card itself does not support any asynchronous protocols supported directly by native PC/SC components.

Dual-interface cards work differently. Their CPU supports communication through ISO14443A part 4 (T=CL) allowing on-card MIFARE emulation rather than host-side MIFARE emulation. This means that OMNIKEY 5x21's default mode (for example, host-side MIFARE emulation) must be disabled to support the on-card MIFARE emulation of a dial-interface card.

There are two ways to switch between host-side and card-side MIFARE emulation:

1. Registry keys
2. IO controls using the PC/SC function **`ScardControl()`** as described in Appendix A2.8 MIFARE Emulation Mode (OMNIKEY Proprietary API).

The following registry keys let you switch between OMNIKEY MIFARE emulation mode (default) and on-card MIFARE emulation.

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CardMan\RFID

ControlFlags=0x00000004	OMNIKEY's host-side MIFARE emulation ON
	default
ControlFlags=0x00000000	OMNIKEY's host-side MIFARE emulation OFF
	T=CL, for on-card MIFARE emulation

Note: The OMNIKEY 5x21 driver needs to be (re)started after changing the registry keys (disconnect and reconnect the reader).

5.1.4 MIFARE Application Directory (MAD)

To access the MIFARE Application Directory (MAD), two commands are necessary – Authenticate and Read. The following steps describe how to retrieve a MAD 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.

For information about the block content see:

http://www.nxp.com/acrobat_download/other/identification/M001830.pdf

5.2 iCLASS Card

Only access the HID iCLASS cards through OMNIKEY's proprietary **`scardsyn`** API. This synchronous API contains a function that is dedicated to accessing contactless cards using the standard PC/SC card handle.

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

1. Standard mode communication
2. Secured mode communication (OMNIKEY proprietary mode)

Note: OMNIKEY OMNIKEY 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.

5.2.1 Card Access through SCardCLICCTransmit

`SCardCLICCTransmit` is the OMNIKEY proprietary function to access HID iCLASS cards through 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 above table are valid for all the commands. Command specific error codes are documented with their respective command documentation.

Note: The error code '6982' **security status not satisfied**, received during secured communication, blocks any further commands. Remove and reinsert the card to reactivate communication with the card.

5.3 ST LRI64 Support (PC/SC 2.0 add-on)

ST Microelectronics' LRI64 is a memory tag IC with 64-bit Unique ID (UID) and WORM user area. The following table lists PC/SC 2.01 compliant functions that are available for LRI64 based storage cards.

Get UID	implemented according to [PCSC 2.01]
Update Binary	
Read Binary	

This ISO15693 compliant IC is not accessible with standard driver settings. It requires the following registry key setting:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CardMan\RFID]

"ControlFlags"=dword:00000010

Refer to the [PCSC 2.01] and [LRI64] for documentation of PC/SC 2.0 compliant LRI64 card access. The following section describes usage of functions that are not already documented in [PCSC 2.01].

5.3.1 Update Binary

UpdateBinary requires block numbers within the WORM memory area (Write-Once Read-Many).

Examples:

Write '121314' to block '0D' (decimal 12):
Command APDU: 'FFD6000D03121314'
Response APDU: '9000'

Attempt to write '101112 to block '0A' (10 decimal):
Command APDU: 'FFD6000A03101112'
Response APDU: '6282'

For blocks 10 and 11 this works out fine, however, because we previously wrote to block 12, the card responds with '6282' **End of file reached before writing Lc bytes**. After the first write access to block 12 only read operations are supported.

The following APDU attempts to write to block 7:
Command APDU: 'FFD6000701FF'
Response APDU: '6581'

The card responds with '6581' **Memory failure (unsuccessful writing)** because this is a UID byte - write access to the UID area is always locked.

5.3.2 Read Binary

The ReadBinary command is available for all blocks of the LRI64 chip.

Examples:

Reading all 15 blocks from 0 to 14

Command APDU: 'FFB0000000'

Response APDE: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx9000'

Attempt to read 16 blocks

Command APDU: 'FFB0000010'

Response APDE: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx6282'

The response is '6282' or **End of file reached before reading expected number of bytes.** Even though the warning '6282' is returned, all bytes from block 0 up to block 14 are read correctly.

Read blocks 10 and 11 (2 bytes)

Command APDU: 'FFB0000A02'

Response APDE: 'xxxx9000'

Attempt to read an invalid block number:

Command APDU: 'FFB0000F01'

Response APDE: '6A82'

The response is the error code '6A82' because block number 15 does not exist.

5.4 ISO15693-3 Memory Card Support

For detailed information about supported ISO15693 Tags please refer to chapter 10 Reading ISO15693.

READ BINARY and **UPDATE BINARY** is compliant to PS/SC2.01 (see chapter 3 PC/SC 2.0).

6 Communication with MIFARE Plus

Depending on the card security level the reader activate the MIFARE Plus card in the ISO 14443A Layer 3 or in the ISO 14443A Layer 4 (T=CL).

Security Level	Protocol Type
MIFARE Plus SL 0	ISO 14443 A – 4
MIFARE Plus SL 1	ISO 14443 A – 3
MIFARE Plus SL 2	ISO 14443 A – 3
MIFARE Plus SL 3	ISO 14443 A – 4

Note : The OMNIKEY synchronous API do not support the new MIFARE Plus cards e.g. SL1 cards. The command set from PC/SC 2.01 part 3 must be used. The MIFARE functions from the sample application „contactlessdemoVC“ and „contactlessdemoVB“ do not work with MIFARE Plus cards.

6.1 ISO 14443 A – part 4 card communication

If the card is activated in protocol layer 4, then the application can communicate with the MIFARE Plus card by calling SCardTransmit. The card command will be transferred directly to the MIFARE Plus card by using the T=CL protocol layer. The T=CL protocol layer is done by the driver. The application can use this type of communication for all card commands in SL0 and SL3. For MIFARE Plus details refer to the MIFARE Plus data sheet from NXP.

The application can execute the card provisioning in security level 0 or the AES authentication in security level 3 by direct transferring of the MIFARE Plus commands.

6.2 ISO 14443 A – part 3 card communication

If the card is activated in protocol layer 3, then the application can not use the direct card communication. For this type of communication an transparent transmission cannel to the card is necessary. Currently is an amendment proposal for PC/SC specification part 3 (HID, NXP) for this type of communication in discussion with the PC/SC work group.

Because the standardization is not concluded, the OMNIKEY 5x21 reader provide an HID proprietary transparent cannel. In this cannel the application can communicate with generic card commands (clause 6.3, 6.4 and 6.5).

6.3 Open Generic Session

Stop the driver activity for card tracking and initialize the generic command session. Take the card control to the application

Table 1 INIT GENERIC SESSION Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Init Session	0xFF	0xA0	0x00	0x07	0x03	0x01 0x00 0x01	-

Table 2 INIT GENERIC SESSION Command Output

Data Out
SW1 SW2 = 0x9000

At first the application must send the following APDU with SCardTransmit

Send FFA0000703010001

Receive 9000

6.4 Generic Card Commands

Write the Mifare+ command in an transparent cannel to the card. The Application send the Generic Card Command APDU with SCardTransmit.

Table 3 GENERIC CARD COMMAND APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Card Command	0xFF	0xA0	0x00	0x05	6+n	01 00 F3 00 00 64 + Mifare+ command	00

Preamble	Mifare+ card command	Explanation
01 00 F3 00 00 64	E1 81	ISO14443-3 RATS
01 00 F3 00 00 64	0A 01 70 02 90 00	ISO14443-4 First Authentication

Do never change the red labeled preamble.

The green labeled data field is the PCB and CID. The application is responsible for the correct usage of the Protocol Control Byte (PCB) 0000 1010. The green labeled bit 0 is the block number (see ISO14443-4 clause 7.5.3 Block numbering rules).

Table 4 GENERIC CARD COMMAND Output

Data Out					
RF Controller Status	Mifare+ card answer			SW1 SW2	
Byte1 Byte 2	Byte 3 … n-2			Byte n-1 Byte n	
00 00	[PCB+CID]	SC	Data	0x9000	successful
	[0A 01]	90	[XX XX … XX]	0x6400	no card answer (TimeOut)

The green labeled PCB, CID filed is only available if the card is switched to ISO14443-4. The data field can be empty. The status code in this sample is the success code.

Sample for Mifare+ commands with the GENERIC INTERFACE Command APDU.

Sample for switching to ISO14443 part 4 (RATS):
Send FFA00005080100F3000064E08100
Receive 00000C757784024D46505F454E479000

Sample for first authentication:
Send FFA000050C0100F30000640A017002900000
Receive 00000A0190XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX9000

Sample for SL1 authentication:
Send FFA00005090100F300006476049000
Receive 000090XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX9000

6.5 Close Generic Session

Continue the driver activity for card tracking and close the generic command session. Take the card control from the application to the driver.

Table 5 CLOSE GENERIC SESSION Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Close Session	0xFF	0xA0	0x00	0x07	0x03	0x01 0x00 0x02	-

Table 6 INIT GENERIC SESSION Command Output

Data Out
SW1 SW2 = 0x9000

After the generic interface session the session must be closed. Do never forgot this step.

The application must send the following APDU with SCardTransmit:

Send FFA0000703010002
Receive 9000

7 CardMan 5x21-CL Keys

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

7.1 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 on Book 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

Key Number	Key Name	Key Length	Key Type	Memory Type
'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'	K_{MD0B1} (Default Master Key for application 1 of page 0 on Book 1 of iCLASS card)	8 bytes	Card Key	Non-volatile memory
'35'..'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			

Key Number	Key Name	Key Length	Key Type	Memory Type
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: OMNIKEY 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 cannot be updated. Updates of key 0x22 are RFU and currently not supported.

7.2 Key Container and 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 not necessary that the host application knows anything about the physical storage location. Load keys into a key container 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’	
’38’	08	K_{MD0B1}	’34’	

7.3 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}, and K_{MD0B1}	'23' to '31' and '34'	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.

8 Standard Communication with iCLASS Card

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

8.1 APDU Structure for Standard Communication

iCLASS cards are supported through 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 (through pucSendData)

CLA	INS	P1	P2	Lc	Data in	Le
'80'	'xx'	'xx'	'xx'	'xx'	'xx' … 'xx'	'xx'

Response APDU (through pucReceivedData)

Data out	SW2	SW1
'xx' … 'xx'	'xx'	'xx'

8.2 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

8.2.1 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.

Select the appropriate page and book before authentication to an iCLASS card application for performing read/write access. 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 1st book 1: for 2nd book on iCLASS 32KS	0	Page number 0-7		

Page Selection Examples:

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

Reference section 5.2.1-Card Access through 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 is helpful to call **Select Page** anyway, in case you need to retrieve the card serial number, configuration block, or application issuer data.

8.2.2 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. OMNIKEY 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: Only load a key in volatile memory 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 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

Reference section 5.2.1-Card Access through SCardCLICCTransmit for additional status words common to all iCLASS access functions.

8.2.3 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. Easily load key slots with keys by referring to the key number (for example, 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 section 7.2 Key Container and 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

Reference section 5.2.1-Card Access through 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 7.1-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

8.2.4 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 7.1-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

Reference section 5.2.1-Card Access through SCardCLICCTransmit for additional status words common to all iCLASS access functions.

8.2.5 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 information about available blocks reference [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'

Reference section 5.2.1-Card Access through 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'.

8.2.6 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 reference [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

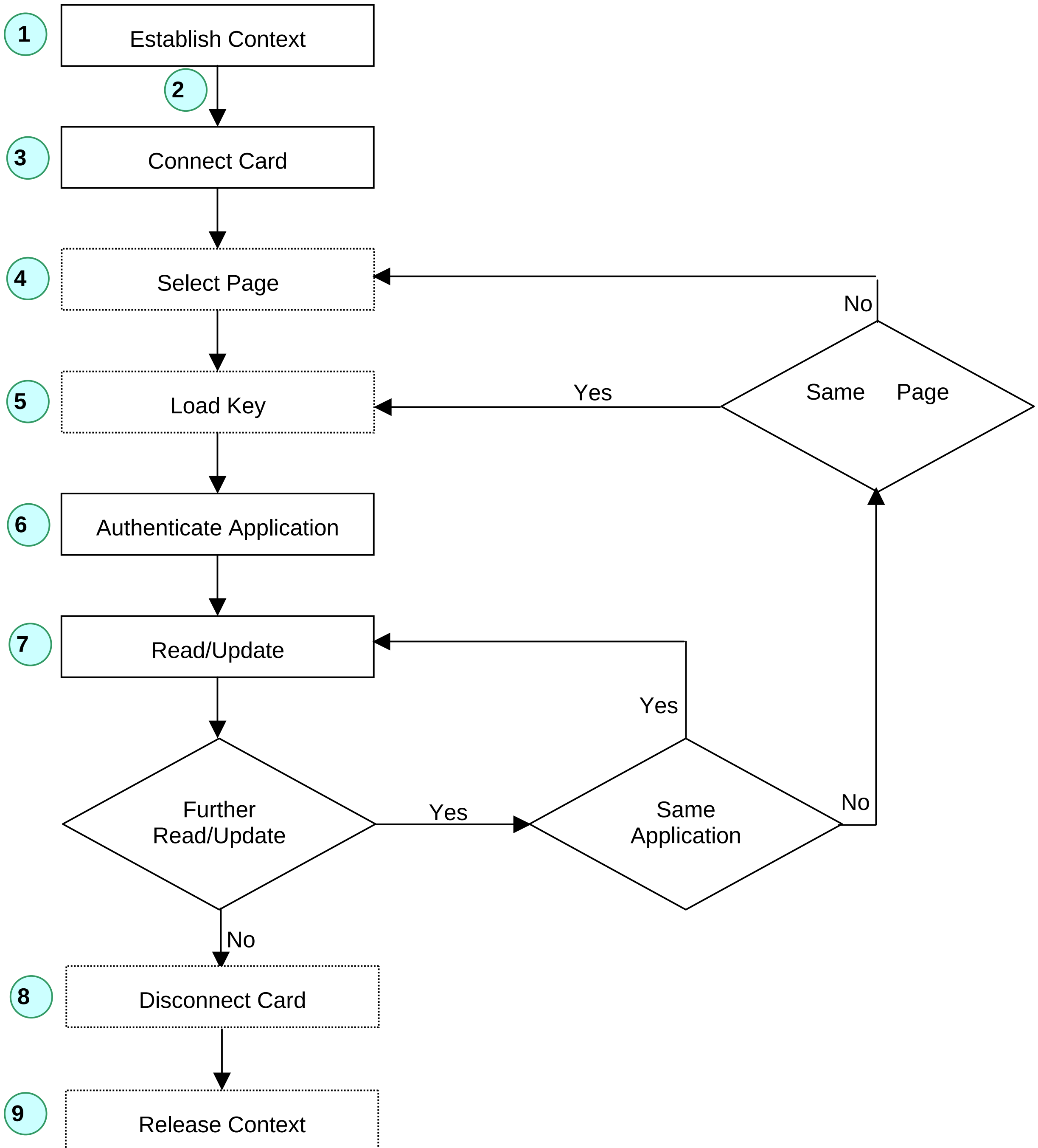
Response Syntax

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

Reference section 5.2.1-Card Access through SCardCLICCTransmit for additional status words common to all iCLASS access functions.

Note: Updating without authenticating to the corresponding application returns ‘6400‘ **Card Execution Error**.

8.3 Communication in Standard Mode



9 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 through 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.

9.1 Multi-Step Approach to a Secure Card Reader System

9.1.1 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. Only initiate sessions upon successful completion of this one-step mutual authentication process.

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

9.1.2 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.

9.1.3 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.

9.1.4 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 is granted.

9.1.5 Integrity of the Radio Frequency (RF) Transmission

Data integrity of an RF transmission with an iCLASS card is enforced with a two-byte checksum (based on CRC algorithm).

9.1.6 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 is securely transmitted between the card and the reader without an eavesdropper reading the data in plaintext.

9.1.7 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.

9.1.8 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 after starting a new session.

9.2 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+ $\mathbf{Lc_{INS}}$+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.

9.2.1 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.

9.2.2 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 encrypted 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.

9.2.3 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.

9.2.4 Proprietary Host and Reader Datagram Example

Host **Reader**

Step 1: Start session

3DES (KCUR.

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

)

→

Session Key(Ks) = Rnd8 + MAC

3DES (KS,

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

)

←

Step n: Any other command

3DES (KS,

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

)

→

3DES (KS,

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

)

←

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.

9.3 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
‘24’	Update Card Key	card command

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

Notes

Secured mode and Standard Mode use different formatting of P1, bit 7 and bit 6 of the Read/Update commands (INS 0xB0 and 0xD6 respectively). Use the two LSBits of P1 to control the encryption of data read or updated.

Lc must always be transmitted in secured mode.

9.3.1 Manage Session (Reader Command)

The Manage Session command is used to start 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

Reference section 5.2.1-Card Access through SCardCLICCTransmit for additional status words common to all iCLASS access functions.

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

9.3.2 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 8.2.1-Select Page (Card Command).

9.3.3 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 8.2.2-Load Key.

9.3.4 Authenticate (Card Command)

Except for the CLA byte '84', the syntax for Authenticate in secured mode is identical to the command described in 8.2.4-Authenticate (Card Command).

9.3.5 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 8.2.5-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.

9.3.6 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 8.2.6-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.

9.3.7 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 8.2.3-GetKeySlotInfo (Reader Command).

9.3.8 Update Card Key

The Update Card Key command is used to change KC or KD.

Command Syntax

CLA	'84'
INS	'24'
P1	'00': New key for KD (application 1) '01': New key for KC (application 2) other values: RFU
P2	Key number where new key is stored.
Lc	'00': empty
Data Field	empty
Le	empty

Response Syntax

Data Field		empty
SW1	**SW2**	status word as described below
'90'	'00'	Success
'65'	'81'	Memory failure (unsuccessful writing)
'69'	'81' '86'	Command incompatible Command not allowed
'6A'	'81'	Function not supported

Reference section 5.2.1-Card Access through SCardCLICCTransmit for additional status words common to all iCLASS access functions.

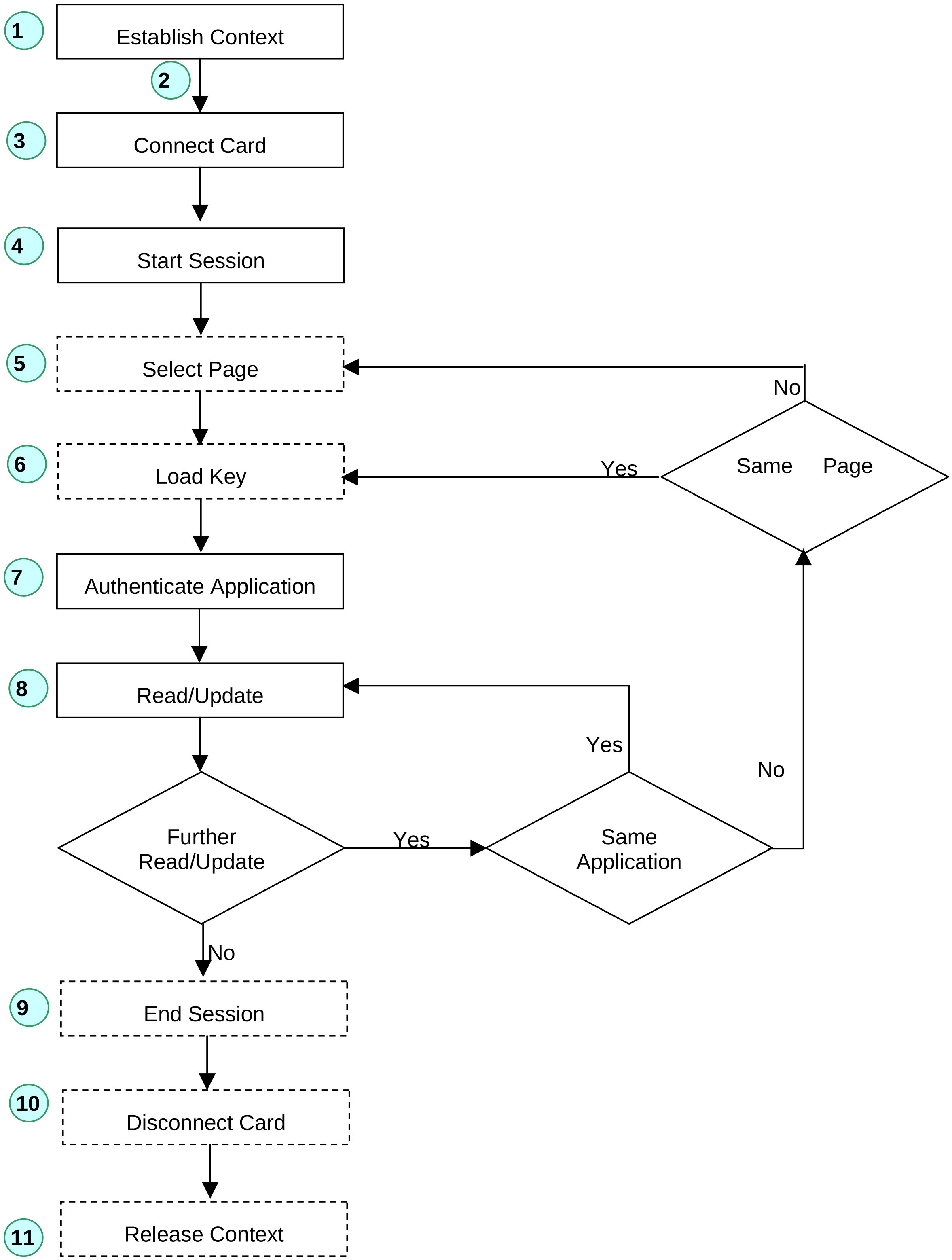
The sequences for using UpdateCardKey command are as follows:

1. If the desired change of the key is not in page 0, the page has to be selected by a **Select Page** command.
2. Load transport/old key by **Load Key** command.
3. Authenticate the card with the old key (key number as used for **Load Key** in step 2).
4. Load new key by **Load Key** command.
5. Now send the **Update CardKey** command with specific P2 (New Key number as loaded in step 4).

Note: Only update KD (application 1) after authentication with KD, and only update KC (application 2) after authentication with KC.

CAUTION: Do not write directly to address 3, 4 where KC and KD are stored, this will destroy the keys.

9.4 Communication at Secured Mode



9.5 Session at Secured Mode APDUs Example

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						SW1 SW2
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.

10 Reading ISO15693

10.1 Products

This document describes the commands for ISO 15693 support of OMNIKEY 5x21.

Applicable readers are:

OMNIKEY 5321 USB

OMNIKEY 6321 USB

OMNIKEY 5321 CL

OMNIKEY 5321 CR

Applicable drivers and operating system:

MS Windows Drivers Version 1.2.0.6

10.2 Tags

The following tags and functions are covered by this document

- iCODE (see table below)
- LRI 64
- SLC Montalbano Technology
- Texas Instruments Tag-it[1]
- Infineon (MY-D, MY-D light)[2]
- All ISO 15693-3 compliant Tags with support for functions marked as **optional**.
  (Include tag functions **Inventory**, **Stay Quiet** …etc)

Support for ICODE tags

Card Type	Chip Type	Support
ICODE 1	SL2 ICS30 01	UID, (Not ISO15693 Part3 compliant)
ICODE SLI	SL2 ICS20	Full
ICODE EPC	SL2 ICS10	Not supported
ICODE UID	SL2 ICS11	Not supported
ICODE UID-TOP	SL2 ICS12	Not supported
ICODE SLI-S / SLI-S HC	SL2 ICS53 / ICS54	Full support except GetSecurityStatus because not supported by card, for further information read the datasheet [iCODE SL2] please

[1] Tag-it Standard and Pro do only support READ BINARY, UPDATE BINARY, GET DATA PICC memory and LOCK, Applicable at MS Windows Drivers Version 1.2.0.14

[2] Applicable at MS Windows Drivers Version 1.2.0.14

10.3 Commands

10.3.1 Get Data

This Get Data command will retrieve information about the inserted command depending on the inserted card. It can be used for kind of contactless cards.

GET DATA Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Get Data	0xFF	0x30	XX	0x00	-	-	XX

P1/P2 denotation

P1	P2	Description
0x00	0x00	RFU
0x01	0x00	RFU
0x02	0x00	AFI of a ISO 15693 card is returned if supported
0x03	0x00	DSFID of a ISO 15693 card is returned if supported
0x04	0x00	PICC memory size is returned if supported
0x05	0x00	IC reference is returned if supported
0x06	0x00	EAS sequence (only for I-CODE SLI cards) is returned , Note : EAS sequence is a bit stream which is sent LSB first !!!

GET DATA Command Output

Data Out
Data + SW1 SW2

Le = 0x00, this means: Return full length of the data

SW1SW2 Examples:

	SW1	SW2	Meaning
Warning	‘62’	‘82’	End of data reached before Le bytes (Le is greater than data length).
Error	‘6A’	‘81’	Function not supported
	‘6C’	‘xx’	Wrong length (wrong number Le; ‘XX’ encodes the exact number) if Le is less than the available UID length)

10.3.2 Put Data

Use this command to write system information to a contactless card.

Put Data Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Put Data	0xFF	0x30	0x00	0x01	3 + N	See table	-

Put Data bytes

Byte 1	Byte 2	Byte 3	Byte 4..n	
Version 0x01	Flag1	Flag2	Data	

Put Data Flag denotation for version 0x01

Flag1	Flag2	
0x00	0x00	RFU
0x01	0x00	RFU
0x02	0x00	AFI of a ISO 15693 card is written if supported
0x03	0x00	DSFID of a ISO 15693 card is written if supported
0x04	0x00	RFU
0x05	0x00	RFU
0x06	0x00	EAS bit is written (for I-Code SLI) cards. Data field consists of one byte (bit 0 is the new value of the EAS bit)[3]
0x00	0x01	Stay quiet (the PICC does not answer any more any response), currently not supported

The following table introduces examples of SW1SW2 and their meaning.

Put data Command Error Codes

	SW1	SW2	Meaning
Warning	'62'	'82'	Block or field is locked
	'63'	'00'	No information is given
Error	'64'	'00'	Execution error[4]
	'6A'	'81'	Function not supported
	'69'	'82'	Security status not satisfied
		'86'	Command not allowed, no ISO15693-3 chip

[3] EAS is supported by MY-D; EAS must be enabled in AFI byte (bit 2)!

[4] The chip does not support the optional ISO15693-3 command type.

10.3.3 Lock

Use this command to lock the memory area of a contactless card. [5]

Lock APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Lock	0xFF	0x30	0x00	0x02	3 + N	See table	-

Lock data bytes

Byte 1	Byte 2	Byte 3	Byte 4..n	
Version 0x01	Flag1	Flag2	Data	

Lock Flag denotation for version 0x01

Flags1	Flags2		Data1	Data2
0x00	0x00	Data field contains in 2 bytes the block number	Address (MSB)	Address (LSB)
0x01	0x00	RFU	-	-
0x02	0x00	AFI of a ISO 15693 card is locked if supported	-	-
0x03	0x00	DSFID of a ISO 15693 card is locked if supported	-	-
0x04	0x00	RFU	-	-
0x05	0x00	RFU	-	-
0x06	0x00	EAS bit (only for I-CODE SLI cards) is locked	-	-

The following table introduces SWISW2 examples.

Lock Command Error Codes

	SW1	SW2	Meaning
Warning	'62'	'82'	Block or field already locked
	'63'	'00'	No information is given
Error	'6A'	'81'	Function not supported
	'69'	'82'	Security status not satisfied
		'86'	Command not allowed, no ISO15693-3 chip

[5] Command is not supported by MY-D light; to set and get security you can use the generic command. Reference the Infineon MY-D light specification and OK5x21_ISO15693_GenericCardCommands.doc

10.3.4 Get Security Status

Use this command to retrieve the security status of some memory area of a contactless card.[6]

Get Security Status Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Get Security Status	0xFF	0x30	0x00	0x03	3 + N	See table	XX

Get Security Status data bytes

Byte 1	Byte 2	Byte 3	Byte 4..n	
Version 0x01	Flag1	Flag2	Data	

Get Security Status Flag denotation for version 0x01

Flag1	Flag2		Data1	Data2
0x00	0x00	Block	Address (MSB)	Address (LSB)
0x01	0x00	RFU	-	-
0x02	0x00	AFI (only supported for MY-D, not MY-D light)	-	-
0x03	0x00	DSFID (currently not supported)	-	-
0x04	0x00	RFU	-	-
0x05	0x00	RFU	-	-
0x06	0x00	EAS (not supported by I·CODE-SLI)	-	-

Le codes the number of bytes for which the security status should be retrieved.

[6] Command is not supported by MY-D light; to set and get security you can use the generic command. Reference the Infineon MY-D light specification and OK5x21_ISO15693_GenericCardCommands.doc

For each address/block number/page number, retrieved is one byte with the security status.

I·CODE SLI	Data 1, Data 2 contains the block number (0 – 27). Each block has 4 bytes.
LRI 64	Data 1, Data 2 contains the block number (0 – 14). Each block has 1 bytes.
SLC Montalbano Technology	Data 1, Data 2 contains the block number (0 – 63). Each block has 8 bytes.
MIFARE 1k	Data1, Data2 contains the block number (0 - ((16 * 4) –1)) **Note**: MIFARE 1k has 16 sectors. Each sector has 4 blocks. Each block has 8 bytes. (Get Security Status currently not supported)
MIFARE 4k	Data1, Data2 contains the block number (0 - ((32 * 4 + 16*4) –1)) **Note**: MIFARE 4k has 32 sectors which have 4 blocks and 16 sectors which have 16 blocks. Each block has 8 bytes. (Get Security Status currently not supported)
MIFARE Ultra light	Data1, Data 2 contains the page number (0 – 15). Each page has 4 bytes. (Get Security Status currently not supported)
MIFARE Mini	Data1, Data2 contains the block number (0 - ((5 * 4) –1)) **Note**: MIFARE Mini has 5 sectors . Each sector has 4 blocks. Each block has 8 bytes. (Get Security Status currently not supported)
MY-D	Data 1, Data 2 contains the block number. (SRF55V10P: 0 – 247, SRF55V02P: 0 – 55) Each block has 4 bytes.

The following describes the security status byte.

Type of card	B7	B6	B5	B4	B3	B2	B1	B0
ISO15693-3 compliant chip	x	x	x	x	x	x	x	Write access bit
MIFARE 1K	x	x	x	x	x	C1	C2	C3
MIFARE 4K	x	x	x	x	x	C1	C2	C3
MIFARE Ultra light	x	x	x	x	x	x	x	Lock bit
MIFARE Mini	x	x	x	x	x	C1	C2	C3

X ….. no meaning

The following table describes examples of SW1SW2 and their description:

Get Security Status Error Codes

	SW1	SW2	Description
Warning	'63'	'00'	No information is given
Error	'64'	'00'	Execution error[7]
	'6A'	'81'	Function not supported
	'69'	'82'	Security status not satisfied
		'86'	Command not allowed, no ISO15693-3 chip

[7] The chip does not support the optional ISO15693-3 command type.

10.3.5 Read Binary Command

If the Le field contains only bytes set to '00', then all the bytes until the end of the file shall be read within the limit of 256 for a short Le field or 65 536 for an extended Le field[8].

Read Binary Command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Read Binary	0xFF	0xB0	Address MSB	Address LSB	-	-	XX

Read Binary Command Output

Data Out
Data + SW1 SW2

Read Binary Command Error Codes

	SW1	SW2	Meaning
Warning	'62'	'81'	Part of returned data may be corrupted.
		'82'	End of file reached before reading expected number of bytes.
Error	'69'	'81'	Command incompatible.
		'82'	Security status not satisfied.
		'86'	Command not allowed.
	'6A'	'81'	Function not supported.
		'82'	File not found / Addressed block or byte does not exist.
	'6C'	'XX'	Wrong length (wrong number Le; 'XX' is the exact number).

Le must be a multiple of the block size !

[8] Currently are extended APDU's only supported for Texas Instruments Tag-it and Infineon MY-D.

10.3.6 Update Binary Command

The Lc field contains the length of the field **Data in** field. For a short Lc field the data length is 1 <= Lc < 256 and for a extended Lc field[9] the data length is 1 <= N_C < 65 536.

Update Binary Command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Read Binary	0xFF	0xD6	Address MSB	Address LSB	XX	Data	-

Update Binary Command Output

Data Out
SW1 SW2

Update Binary Command Error Codes

	SW1	SW2	Meaning
Warning	'62'	'81'	A part of the returned data may be corrupted.
		'82'	End of file reached before writing Lc bytes.
Error	'65'	'81'	Memory failure (unsuccessful writing).
	'69'	'81'	Command incompatible.
		'82'	Security status not satisfied.
		'86'	Command not allowed.
	'6A'	'81'	Function not supported.
		'82'	File not found / Addressed block or byte does not exist.

Lc must be a multiple of the block size!

[9] Currently are extended APDU's only supported for Texas Instruments Tag-it and Infineon MY-D.

10.3.7 Update Single Byte Command

Use this command to write a single byte within a block. Currently, this command is only supported for Infineon MY-D.

Update Single Byte Command APDU

Command	Class	INS	P1	P2	Lc	Data In	Le
Put Data	0xFF	0xD7	0x00	0x00	6	See table	-

Update Single Byte Data bytes

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Version 0x01	Block Address MSB	Block Address LSB	Offset within Block MSB	Offset within Block LSB	Data to be written

The offset must be less than block size.

Update Single Byte Command Output

Data Out
SW1 SW2

Update Single Byte Command Error Codes

	SW1	SW2	Meaning
Error	'65'	'81'	Memory failure (unsuccessful writing).
	'69'	'82'	Security status not satisfied.
	'6A'	'81'	Function not supported.
		'82'	File not found / Addressed block or byte does not exist.

11 OMNIKEY 5321 PAY Application Interface

The OMNIKEY 5321 PAY has an EMVCo Contactless Level 1 Type Approval. The application interface (API) is compliant to PC/SC 2.01.

11.1 *PayPass*™ card transactions

For card detection the application can use the SCCardStatusChange() function. If an PAY card is present the SCARD_STATE_MUTE status flag must be checked. If the SCARD_STATE_MUTE status flag is 1 then the PCD has found more than one card in the operation volume (collision detected). If this status flag is zero, then the application can continue with the card transactions, e.g. Select PPSE (Proximity Payment System Environment).

For transactions with an PAY Card the application use the SCacrdTransmit() function.

See Appendix A2.10 EMVCo Contactless Level 2 Transactions.

If all transactions are complete the application must disconnect the card with the dwDisposition value SCARD_UNPOWER_CARD. This is necessary for the correct card removal procedure of the PCD.

11.2 LED and Buzzer control

For LED and buzzer control the device provide an PC/SC IO-C0ntrol. This IO-Control can be used for the activation of OMNIKEY 5321 PAY read indication. The OMNIKEY 5321 PAY can use the *PayPass*™ light/LED method for read indication and the additional tree indicators must light the sequence. An audio indication (buzzer) can be used to indicate the success tone. For more information about the light/LED read indication method, see the relevant ongoing specification from EMVCo and MasterCard® *PayPass*™.

Table 7: Parameter for IO-Control SIGNAL

SCardControl Parameter	Description			
dwControlCode	CM_IOCTL_SIGNAL			
lpInBuffer	+0	PPARAM_SIGNAL	UCHAR	ucCommand
			UCHAR	ucParam1
			UCHAR	ucParam2
			UCHAR	ucRFU[10]
nInBufferSize	>= 3			
lpOutBuffer	Empty			
nOutBufferSize	>= 0			
lpBytesReturned	0			

Table 8: Summary of SIGNAL Commands

Command	Value	Description
PAYPASS_SIGNAL	0x20	Complete *PayPass*™ Audio and Visual Sequence
PAYPASS_SIGNAL_MAINLED	0x21	Control of Main LED
PAYPASS_SIGNAL_ADDLED	0x22	Control of additional *PayPass*™ LED 2-4
ACOUSTIC_SIGNAL_BEEPER_ON	0x10	Control of *PayPass*™ Audio Tone ON
ACOUSTIC_SIGNAL_BEEPER_OFF	0x11	Control of *PayPass*™ Audio Tone OFF

11.2.1 SIGNAL Command – PayPass Signal

This command clone the *PayPass*™ Audio and Visual Sequence for the event "card read complete successful", according to **MasterCard® *PayPass*™ *Terminal Implementation Guide***.

Table 9: Parameter for SIGNAL Command – PayPass Signal

Parameter	Description			
lpInBuffer	Command	Param1	Param2	RFU
	20	loudness	--	--
nInBufferSize	>= 2			
lpOutBuffer	Empty			
nOutBufferSize	>= 0			
lpBytesReturned	0			

11.2.2 SIGNAL Command – PayPass Signal MAIN LED

The reader main LED (bicolour red/green) is by default under control of firmware and driver. In any cases of MasterCatrd® PayPass ™ terminal implementation an application control of this LED is required. With this comman the application can assume the LED control.

Table 10: Parameter for SIGNAL Command – PayPass Signal MAIN LED

Parameter	Description				
lpInBuffer	Command	Param1	Param2	Param3	RFU
	21	00 – CCID ESC command 01 – USB Pipe Control	LED status	00 – by default 03 – application controlled	--
nInBufferSize	>= 4				
lpOutBuffer	Empty				
nOutBufferSize	>= 0				
lpBytesReturned	0				

For LED control before receiving the PICC answer the application must use Param1 = 01 as, USB Pipe Control Command.

Param2 is coded as 0000 00xx (bit 2…7 is RFU)

Summary of Param2

LED status	Value	Description
Bit 0	1	bicolour **green** LED on
	0	bicolour **green** LED off
Bit 1	1	bicolour **red** LED on
	0	bicolour **red** LED off

For details see the code snippet in Appendix A2.12 PayPass™ Signal MAIN LED

11.2.3 SIGNAL Command – PayPass Signal Additional LEDs

For represent the status of the contactless payment application the MasterCatrd® PayPass ™ terminal implementation require three additioal LEDs for visual indication, e.g. *contactless application process was completed successfully*. This three LEDs are exclusive for the application. Driver and firmware du not use thes tree additional LEDs.

Table 11: Parameter for SIGNAL Command – PayPass Signal Additional LEDs

Parameter	Description				
	Command	Param1	Param2	Param3	RFU
lpInBuffer	22	00 – CCID ESC command 01 – USB Pipe Control	LED status	--	--
nInBufferSize	>= 3				
lpOutBuffer	Empty				
nOutBufferSize	>= 0				
lpBytesReturned	0				

Param2 is coded as 0000 0xxx (bit 3…7 is RFU)

Summary of Param2

LED status	Value	Description
Bit 0	1	green LED2 on
	0	green LED2 off
Bit 1	1	green LED3 on
	0	green LED3 off
Bit 2	1	green LED4 on
	0	green LED4 off

For details see the code snippet in Appendix A2.13 PayPass™ Signal Additional LEDs

11.2.4 SIGNAL Command – PayPass Signal Tone

No commad parameters are required. The command code 0x10 (ACOUSTIC_SIGNAL_BEEPER_ON) turn on the buzzer and the command code 0x11 (ACOUSTIC_SIGNAL_BEEPER_OFF) turn off the buzzer. See the Table 8: Summary of SIGNAL Commands.

For details see the code snippet in Appendix A2.14 PayPass™ Signal Tone

11.3 Switch-over the operating mode

The OMNIKEY 5321 PAY require the EMVCo Level 1 PDC processing. After the driver is installed, the PCD (Proximity Coupling Device) do this by default. The Reader can also be used in standard ISO mode. For dynamic changing between RFID-ISO mode and EMVCo L1 mode the driver support an IO-Control, described in this chapter. See also the code snipped in A2.11 Set RFID operating mode.

Note: The operating volume is optimized for EMVCo L1. This is not compliant to the requirements of ISO / ICAO.

Table 12: Parameter for IO-Control Set RFID Operation Mode

SCardControl Parameter	Description			
dwControlCode	CM_IOCTL_SET_OPERATION_MODE			
lpInBuffer	+0	bOperationMode	0x10	OPERATION_MODE_RFID_ISO
			0x11	OPERATION_MODE_RFID_PAYPASS
nInBufferSize	>= 1			
lpOutBuffer	Empty			
nOutBufferSize	>= 0			
lpBytesReturned	0			

If the reader is switched to ISO mode, the complete functionality of an standard OMNIKEY 5x21 can be used.

Note: Currently the EMVCo type approval is confined to the firmware version 1.75. This firmware version do not support the read and write operations of iClass cards.

For an static usage in ISO mode the reader behavior can also switched with the following registry entry:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CardMan\CardInterface]

"ContactlessDefault"=dword:00000000

12 CardMan 5125 Registry Settings

The following registry entry specifies the used card format. See Figure 6 - Registry Editor.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CardMan]
ProxFormat=dword:000000ff
```

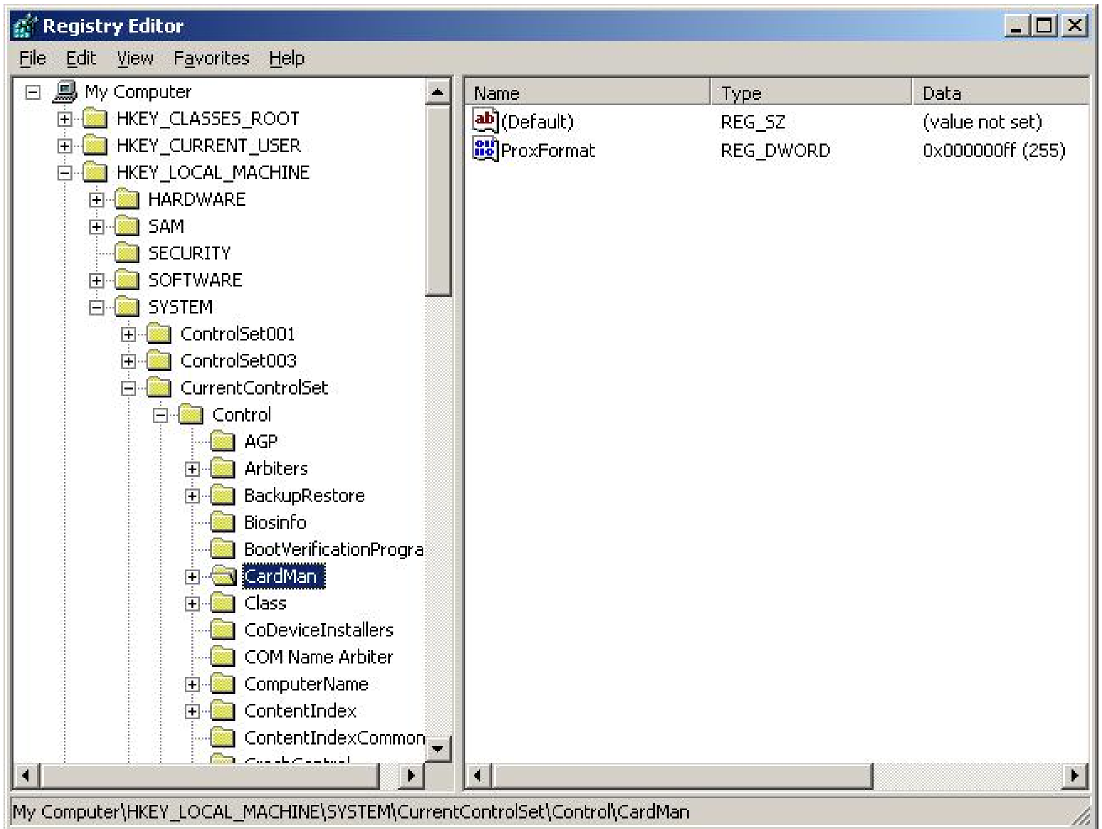


Figure 6 - Registry Editor

The following table shows allowed registry key entries. If there is a void value, the driver works like "ProxFormat"=dword:00000000 is entered. (= no decoding)

Prox Format Value Decimal	Card Format	Data Content
0	Wiegand Raw	-
1	H10301	26 bit (FAC+CN)
2	H10302	37 bit (CN)
4	H10304	37 bit (FAC+CN)
20	H10320	32 bit clock/data card
100	Corp 1000	35 bit (CIC+CN)
254	AUTO	Automatic mode
255	CUSTOMER	Customer defined

12.1 Legend / Additional Information

FAC …………….. Facility Code

CN …………….. Card Number

CIC …………….. Customer Identifier Code

Get detailed information about the card formats from www.hidglobal.com.

The structure of the decoded ATR depends on the card format and the used registry key. The next table shows in which way the card information is mapped into the ATR depending to the **ProxFormat** value.

CARD			Registry Key (hex format)	Decoded ATR
FORMAT	**FC / CIC**	**CN**		
H10301	AAAA	BBBBBBB	01	3B 06 01 AA AA BB BB BB
H10302	-	BBBBBBBBBBBB	02	3B 07 02 BB BB BB BB BB BB
H10304	AAAAA	BBBBBB	04	3B 07 04 0A AA AA BB BB BB
H10320	-	BBBBBBBB	14	3B 05 14 BB BB BB BB
Corp 1k	AAAA	BBBBBBBB	64	3B 07 64 AA AA BB BB BB BB

12.2 Automatic Mode

If the value of the **ProxFormat** key is set to 254 (0xfe), the detection of the card format and the conversation of the ATR is done automatically by the driver. The function of the automatic mode is restricted because of many different card formats.

Example: The only difference between the two 37bit formats H10302 and H10304 is that H10304 contains a facility code in the ATR and the H10302 not. Therefore, it is impossible to differentiate the two formats on the basis of the ATR.

The automatic mode supports and decodes the following formats correctly:

H10301	H10302
H10320	Corp 1000

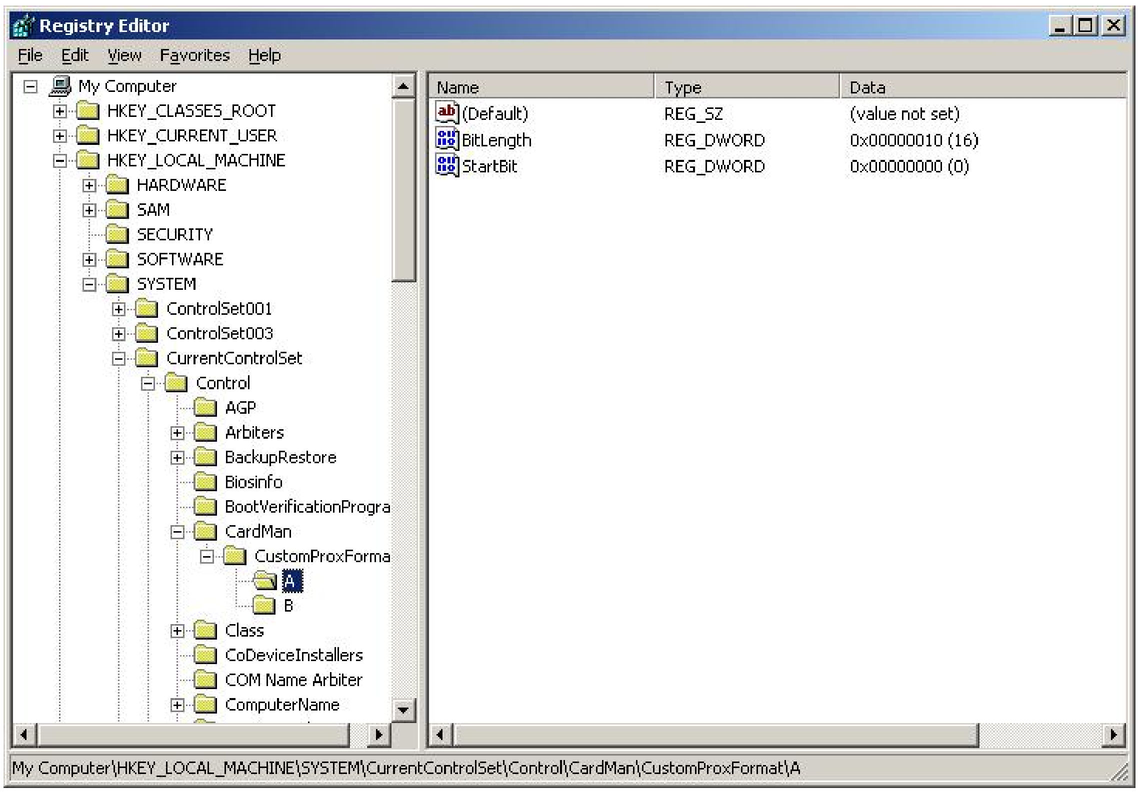
12.3 Windows Custom Mode

Because of many different card formats, the driver provides an option to decode the ATR through custom settings. To enable this function set the **ProxFormat** key to 255 (0xff). This chapter explains in which way the driver is decoding the ATR by setting additional registry keys.

Split the bit-data-stream into up to 15 data-fields. Each data field is labeled with a letter (A, B …) and is defined with a StartBit and the BitLength.

The StartBit specifies the position in the bit-data-stream, starting with the LSb. In the case, the first bit of the data-field is the LSb, while the value of StartBit must be 0.

Example:



The data-fields are converted into BCD format next and mapped into the ATR in the sequence A B C …. etc.

12.3.1 H10301 format example

26 bit code (24 information bits)

Card		w/o RegKey	ProxFormat = 255
FC=1	CN=12345	3B 05 00 02 02 60 73	3B 06 01 00 01 01 23 45

```
[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat]

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\A]
“StartBit”=dword00000011
“BitLength”=dword:00000008

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\B]
“StartBit”=dword00000001
“BitLength”=dword:00000010
```

```
PAAAAAAAABBBBBBBBBBBBBBBBP
10000000100110000000111001 1
```

P Parity Bit
A Facility Code (FAC)
B Card Number (CN)

Both data fields converted into BCD format:

```
          00000001 ->      00 01        (4 digits, defined by H10301 format)
0011000000111001 -> 01 23 45      (6 digits, defined by H10301 format)
```

12.3.2 Example: H10302 format

37bit code (35 information bits, CN)

Card	w/o RegKey	ProxFormat = 255
CN=1	3B 06 00 00 00 00 00 02	3B 07 02 00 00 00 00 00 01

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat]

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\A]

“StartBit”=dword:00000001

“BitLength”=dword:000000023

PAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAP

0000000000000000000000000000000000010

P Parity Bit

A Card Number (CN)

Data field converted into BCD format:

00000000000000000000000000000000001 -> 0 00 00 00 00 01

(11 digits, defined by H10302 format)

12.3.3 H10304 Format Example

37bit code (35 information bits, FC+CN)

Card		w/o RegKey	ProxFormat = 255
FC=65535	CN=524287	3B 06 00 0F FF FF FF FF	3B 07 04 06 55 35 52 42 87

```
[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat]

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\A]
“StartBit”=dword00000014
“BitLength”=dword:00000010

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\B]
“StartBit”=dword00000001
“BitLength”=dword:00000013
```

```
PAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBP
0111111111111111111111111111111111111
```

P Parity Bit
A Facility Code (FAC)
B Card Number (CN)

Both data fields converted into BCD format:

```
1111111111111111    ->    6 55 35 (5 digits, defined by H10304 format)
1111111111111111111 ->   52 42 87 (6 digits, defined by H10304 format)
```

12.3.4 Corp 1000 Format Example

35bit code (32 information bits, FC+CN)

Card		w/o RegKey	ProxFormat = 255
FC=4095	CN=2	3B 06 00 03 FF E0 00 05	3B 07 64 40 95 00 00 00 02

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat]

```
[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\A]
"StartBit"=dword00000015
"BitLength"=dword:0000000C

[HKLM\SYSTEM\CurrentControlSet\Control\CardMan\CustomProxFormat\B]
"StartBit"=dword00000001
"BitLength"=dword:00000014

PPAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBP
01111111111111000000000000000000101
```

P Parity Bit

A Facility Code (FAC)

B Card Number (CN)

Both data fields converted into BCD format:

`111111111111 -> 40 95`

[4 digits, defined by Corp1000 format]

`00000000000000000010 -> 00 00 00 02`

[8 digits, defined by Corp1000 format]

12.4 Linux & Mac OS X Custom Mode

Because Linux and Mac OS X based operating systems do not have a registry, the Prox specific settings have to be done in an extra file.

This file is named **cmrfid.ini** and it is copied to the /etc/ directory (the directory is the same on Mac OS X and Linux platform) during driver installation. Edit this file with any text editor.

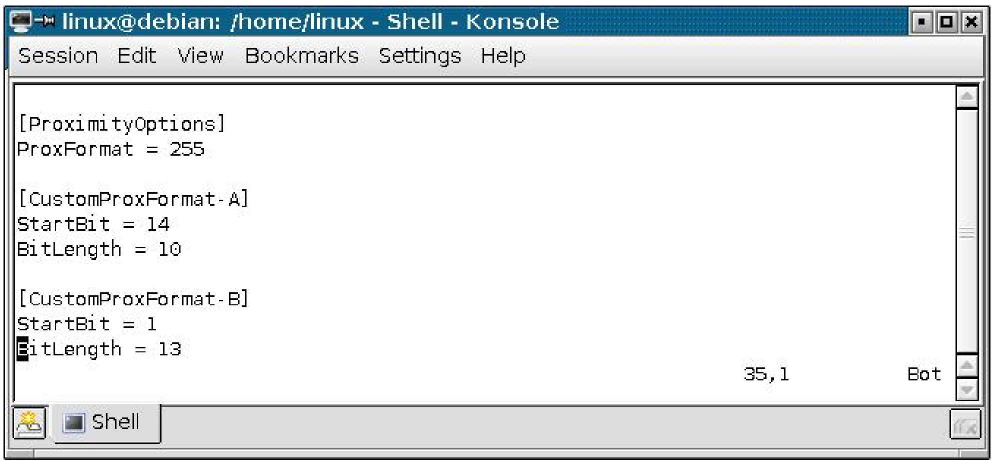
Note: You need root permissions to edit this file.

By default the **ProxFormat** mode is set to automatic:

[ProximityOptions]

ProxFormat = 254 (0xfe)

To enable the custom mode function on Linux, set the **ProxFormat** key to 255 (0xff). Additionally, add entries regarding the needs of the used card. In the following image, a H10304 format card was added.



First the ProxFormat value in the [ProximityOptions] section has to be changed to 255 (custom mode):

[ProximityOptions]

; ProxFormat = 254 (= automatic mode)

ProxFormat = 255

Then, add the entries for the format options. For Windows based operating systems, create registry keys as mentioned in the sections above. For Linux you need to create additional format sections for the used card (H10301 in this example):

[CustomProxFormat-A]

StartBit = 11

BitLength = 8

[CustomProxFormat-B]

StartBit = 1

BitLength = 10

For any other card reference the different examples above. The settings are the same for Windows; the only difference is that the settings have to be done in the **cmrfid.ini** file.

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.hidglobal.com/omnikey.

If you choose the default installation settings, sample code is found in: **`c:\omnikey\samples\contactlessdemovc`**.

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

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

Note: Integrate MIFARE cards through non-proprietary, PC/SC 2.0 compliant function calls.

A1.1 Overview

From the **Connected Reader** list (top-left corner), select the reader. The list contains all readers available to the smart card resource manager. When a card is inserted, displayed are the **ATR**, **UID** and **Card Name** fields. From the **Reader Related Function** frame, select the functions with or without a card in the RF field.

Only use the **MIFARE Functions using Sync API** frame when a MIFARE card is in the field. Use the **ISO 7816/iCLASS/PCSC 2.01** frame for APDU exchange with a CPU card (asynchronous card) in the field.

Each processed command produces output in the output log. Clear the log with the **Refresh Output Screen** button. The return status of the last executed function is shown in the **Last Operation Status** frame.

Close the application with the **Exit** button.

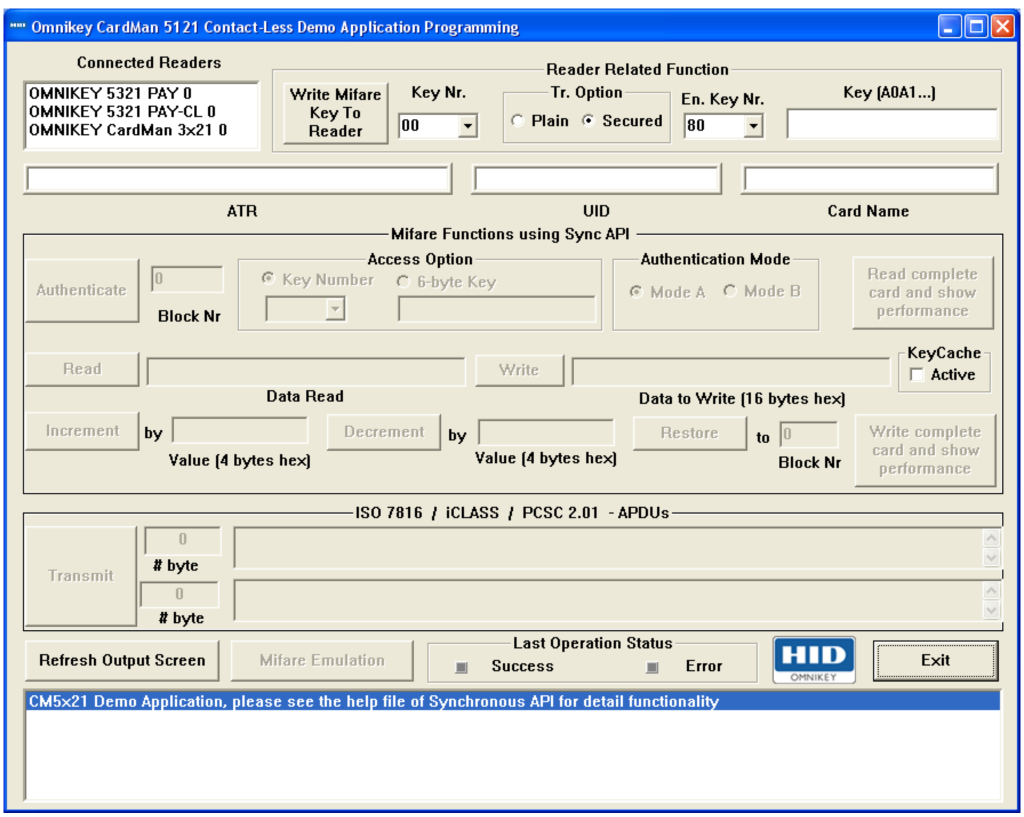


Figure 7: Sample Program Screen

A1.2 Reader Related Functions

Reader related functions do not require a card in the field.

To store a MIFARE key, complete the following:

- Define a key number to determine where to store the key.
- Select plain or secured as the mode of the key transmission. 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 Functions Using Synchronous API

Before using the **MIFARE Functions using Sync API**, authenticate the card. (MIFARE UltraLight does not need authentication).

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

- In the field **Block Nr***,* enter the authentication block number.
- In the field **Access Option** choose to supply a key number or plain key.
- In the field **Authentication Mode** choose **Mode A** or **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 as 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, see section 8-Standard Communication with iCLASS Card. This is an easy way of experimenting with the available functions.

A2 Code Examples

This section lists coding examples 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) currently connected to the card through the air interface. Use the UID as the card serial number. The UID is available for every ISO 14443 A/B or ISO 15693 compliant cards. It does not matter whether the card is a CPU or storage card. This makes GetUID the ideal candidate for Hello Card type applications. If you do not have access to application keys, the UID serves as a valuable identifier allowing 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 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 demonstrates how to authenticate 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 MIFARE Emulation Mode (OMNIKEY Proprietary API)

With the following code switch the MIFARE Emulation Mode on and off.

```
#define CM_IOCTL_SET_RFID_CONTROL_FLAGS       SCARD_CTL_CODE(3213)

  DWORD dwActiveProtocol;
  DWORD dwControlFlag;

  BYTE  InBuffer[16];
  BYTE  OutBuffer[16];
  DWORD dwInBufferSize ;
  DWORD dwOutBufferSize;
  DWORD dwBytesReturned;
  DWORD *Mask           = (DWORD *)InBuffer;
  DWORD *Value          = (DWORD *)InBuffer+1;
  DWORD dwControlCode   = CM_IOCTL_SET_RFID_CONTROL_FLAGS;



  memset(InBuffer, 0x00, sizeof(InBuffer));
  memset(OutBuffer, 0x00, sizeof(OutBuffer));

  *Mask         = 0x00000004;
  *Value                = dwControlFlag & *Mask;
  dwInBufferSize  = 8;
  dwOutBufferSize = 0;
  dwBytesReturned = 0;

  SCard_Status = SCardControl(hCard,
                              dwControlCode,
                              (LPCVOID)InBuffer,
                              dwInBufferSize,
                              (LPVOID)OutBuffer,
                              dwOutBufferSize,
                              &dwBytesReturned);

  if (SCard_Status == SCARD_S_SUCCESS)
  {
        if(dwControlFlag)
                sprintf(szText,"MIFARE\t");
          else
                sprintf(szText,"T=CL\t");
  }
  else
  {
        sprintf(szText,"IO Cntrol error\r");
  }

  // The card is disconnected after changing the MIFARE emulation mode
  do
  {
        sReaderState.szReader = szReaderName;
        sReaderState.dwCurrentState = SCARD_STATE_EMPTY;
        sReaderState.dwEventState = SCARD_STATE_EMPTY;
        SCardGetStatusChange(hContext,50,&sReaderState,1);
  }
  while((sReaderState.dwEventState & SCARD_STATE_PRESENT) == 0);
```

A2.9 iCLASS Select Page (OMNIKEY Proprietary API)

The following code 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);
}
```

A2.10 EMVCo Contactless Level 2 Transactions

The following code snippet shows a typical OMNIKEY 5321 PAY transaction loop.

```
SCARDCONTEXT      hContext;
SCARDHANDLE       hCard;
SCARD_READERSTATE sReaderState;
CHAR*             szReaderName;
DWORD             dwShareMode;
DWORD             dwPreferredProtocols;
DWORD             dwActiveProtocols;
UCHAR             ucByteSend[256];
DWORD             dwNByteSend;
UCHAR             abByteReceive[256];
DWORD             dwRecvLength;
DWORD             SCard_Status;

UCHAR             abSelectPPSE[20] = {0x00,0xA4,0x04,0x00,                 // CLA,INS,..
                                      0x0E,                                // Lc
                                      0x32,0x50,0x41,0x59,0x2E,0x53,0x59,  // Data field
                                      0x53,0x2E,0x44,0x44,0x46,0x30,0x31,
                                      0x00};                               // Le

// TODO: Code for PAY application

do
{
  // wait for card
  do
  {
    sReaderState.szReader = szReaderName;
    sReaderState.dwCurrentState = SCARD_STATE_EMPTY;
    sReaderState.dwEventState = SCARD_STATE_EMPTY;
    SCardGetStatusChange(hContext,30,&sReaderState,1);
    Sleep(20);
  }
  while((sReaderState.dwEventState & SCARD_STATE_PRESENT) == 0);

  if ((sReaderState.dwEventState & SCARD_STATE_MUTE) != 0)
  {
```

```
      // Card present, Collision detected

      // TODO: Code for PAY application

      // wait for remove card
      do
      {
        sReaderState.szReader = szReaderName;
        sReaderState.dwCurrentState = SCARD_STATE_PRESENT;
        sReaderState.dwEventState = SCARD_STATE_PRESENT;
        SCardGetStatusChange(hContext,30,&sReaderState,1);
        Sleep(20);
      }
      while((sReaderState.dwEventState & SCARD_STATE_EMPTY) == 0);
      continue;
    }

    // TODO: Code for PAY application

    // Connect card

    dwShareMode = SCARD_SHARE_SHARED;
    dwPreferredProtocols =  SCARD_PROTOCOL_T1;

    SCard_Status = SCardConnect( hContext,
                                 szReaderName,
                                 dwShareMode,
                                 dwPreferredProtocols,
                                 &hCard,
                                 &dwActiveProtocols );

    // TODO: Code for PAY application

    memcpy( abByteSend, abSelectPPSE, 20);
    dwNByteSend = 20;
    do
    {
      dwRecvLength = 256;
      SCard_Status = SCardTransmit ( hCard,
                                     SCARD_PCI_T1,
                                     abByteSend,
                                     dwNByteSend,
                                     NULL,
                                     abByteReceive,
                                     &dwRecvLength );

    // TODO: Code for PAY application

    }
    while( /*TODO: Code for PAY application*/ );

    // now disconnect the card
    SCard_Status = SCardDisconnect( hCard, SCARD_UNPOWER_CARD );
    // TODO: Code for PAY application

    // wait for remove card
    do
    {
      sReaderState.szReader = szReaderName;
      sReaderState.dwCurrentState = SCARD_STATE_PRESENT;
      sReaderState.dwEventState = SCARD_STATE_PRESENT;
      SCardGetStatusChange(hContext,30,&sReaderState,1);
      Sleep(20);
    }
    while( (sReaderState.dwEventState & SCARD_STATE_EMPTY) == 0 );
   // TODO: Code for PAY application
  }
  while( /*TODO: Code for PAY application*/ );
  // TODO: Code for PAY application
```

A2.11 Set RFID operating mode

The following code snipped shows a sample for setting the operating mode:

```
#define CM_IOCTL_SET_OPERATION_MODE   SCARD_CTL_CODE (3107)
#define OPERATION_MODE_RFID_ISO       0x10
#define OPERATION_MODE_RFID_PAYPASS   0x11

BYTE    InBuffer[4];
BYTE    OutBuffer[4];
DWORD   dwInBufferSize;
DWORD   dwOutBufferSize;
DWORD   dwBytesReturned;
DWORD   dwControlCode = CM_IOCTL_SET_OPERATION_MODE;

memset(InBuffer, 0x00, sizeof(InBuffer));
memset(OutBuffer, 0x00, sizeof(OutBuffer));
*InBuffer       = OPERATION_MODE_RFID_PAYPASS
dwInBufferSize  = 1;
dwOutBufferSize = 0;
dwBytesReturned = 0;

SCard_Status = SCardControl (hCard,
                             dwControlCode,
                             (LPCVOID)InBuffer,
                             dwInBufferSize,
                             (LPVOID)OutBuffer,
                             dwOutBufferSize,
                             &dwBytesReturned);
```

A2.12 *PayPass*™ Signal MAIN LED

The following code snipped shows how the reader main LED can be used under control of an application.

```
#define  CM_IOCTL_SIGNAL              SCARD_CTL_CODE (3058)
#define  PAYPASS_SIGNAL_MAINLED       0x21

BYTE     InBuffer[4];
BYTE     OutBuffer[4];
DWORD    dwInBufferSize ;
DWORD    dwOutBufferSize;
DWORD    dwBytesReturned;
DWORD    dwControlCode;
BYTE     bUSBMode    = 0x01;           // USB Pipe Control
BYTE     bReaderLEDs = 0x02;           // red LED on
BYTE     bLEDMode    = 0x03;           // application controlled

// TODO: Code for PAY application

memset(InBuffer, 0x00, sizeof(InBuffer));
memset(OutBuffer, 0x00, sizeof(OutBuffer));

dwControlCode   = CM_IOCTL_SIGNAL;
InBuffer[0]     = PAYPASS_SIGNAL_MAINLED;
InBuffer[1]     = bUSBMode;
InBuffer[2]     = (bReaderLEDs) & 0x03;
InBuffer[3]     = bLEDMode;
dwInBufferSize  = 4;
dwOutBufferSize = 0;
dwBytesReturned = 0;

SCard_Status = SCardControl( hCard,
                             dwControlCode,
                             (LPCVOID)InBuffer,
                             dwInBufferSize,
                             (LPVOID)OutBuffer,
                             dwOutBufferSize,
                             &dwBytesReturned );
```

```
if (SCard_Status != SCARD_S_SUCCESS)
{

  // TODO: Code for PAY application

}

// TODO: Code for PAY application
```

A2.13 *PayPass*™ Signal Additional LEDs

The following code snipped shows how the addtitional three LEDs can be used under control of an application.

```
#define  CM_IOCTL_SIGNAL              SCARD_CTL_CODE (3058)
#define  PAYPASS_SIGNAL_ADDLED        0x22

BYTE     InBuffer[4];
BYTE     OutBuffer[4];
DWORD    dwInBufferSize ;
DWORD    dwOutBufferSize;
DWORD    dwBytesReturned;
DWORD    dwControlCode;

BYTE      bUSBMode    = 0x01;         // USB Pipe Control
BYTE      bReaderLEDs = 0x1C;         // all additional green LEDs on

memset(InBuffer, 0x00, sizeof(InBuffer));
memset(OutBuffer, 0x00, sizeof(OutBuffer));

dwControlCode   = CM_IOCTL_SIGNAL;
InBuffer[0]     = PAYPASS_SIGNAL_ADDLED;
InBuffer[1]     = bUSBMode;
InBuffer[2]     = (bReaderLEDs >> 2) & 0x07;
dwInBufferSize  = 3;
dwOutBufferSize = 0;
dwBytesReturned = 0;

SCard_Status = SCardControl( hCard,
                             dwControlCode,
                             (LPCVOID)InBuffer,
                             dwInBufferSize,
                             (LPVOID)OutBuffer,
                             dwOutBufferSize,
                             &dwBytesReturned );

if (SCard_Status != SCARD_S_SUCCESS)
{
  // TODO: Code for PAY application

}

// TODO: Code for PAY application
```

A2.14 *PayPass*™ Signal Tone

The following code snipped shows how the buzzer can be used under control of an application.

```
#define  CM_IOCTL_SIGNAL             SCARD_CTL_CODE (3058)
#define  ACOUSTIC_SIGNAL_BEEPER_ON   0x10
#define  ACOUSTIC_SIGNAL_BEEPER_OFF  0x11

BYTE     InBuffer[4];
BYTE     OutBuffer[4];
DWORD    dwInBufferSize ;
DWORD    dwOutBufferSize;
DWORD    dwBytesReturned;
```

```
DWORD    dwControlCode;

memset(InBuffer, 0x00, sizeof(InBuffer));
memset(OutBuffer, 0x00, sizeof(OutBuffer));

dwControlCode   = CM_IOCTL_SIGNAL;
InBuffer[0]     = ACOUSTIC_SIGNAL_BEEPER_ON;
dwInBufferSize  = 1;
dwOutBufferSize = 0;
dwBytesReturned = 0;

SCard_Status = SCardControl( hCard,
                             dwControlCode,
                             (LPCVOID)InBuffer,
                             dwInBufferSize,
                             (LPVOID)OutBuffer,
                             dwOutBufferSize,
                             &dwBytesReturned );

// TODO: Code for PAY application

memset(InBuffer, 0x00, sizeof(InBuffer));
memset(OutBuffer, 0x00, sizeof(OutBuffer));

dwControlCode   = CM_IOCTL_SIGNAL;
InBuffer[0]     = ACOUSTIC_SIGNAL_BEEPER_OFF;
dwInBufferSize  = 1;
dwOutBufferSize = 0;
dwBytesReturned = 0;

SCard_Status = SCardControl( hCard,
                             dwControlCode,
                             (LPCVOID)InBuffer,
                             dwInBufferSize,
                             (LPVOID)OutBuffer,
                             dwOutBufferSize,
                             &dwBytesReturned );

// TODO: Code for PAY application
```

Appendix B - Accessing iCLASS Memory

The following describes the free zones of two typical iCLASS memory layouts.

B1.1 Memory Layout

Shown is the 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)	

Shown is the 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 iCLASS Application 2 - Assigning Space

By default, iCLASS cards have the application limit set to the last byte of its respective memory area. This means the complete memory area is reserved for application 1 and the size of application 2 is set to zero. 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.

1. Select the page you want to configure.
2. Authenticate with the selected page Kd.
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 iCLASS Read / Write Memory - 2KS, 16KS or 8x2KS page 0

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 iCLASS 8x2KS Card - Pages 1 to 7 Read / Write Memory

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 7.1Key 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 - Terms and Abbreviations

The following 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_{MDNB1}	Page N Application 1's on Book 1 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
PCD	Proximity Coupling Device
PICC	Proximity IC Card
PPSE	Proximity Payment System Environment
RDH	Reader Data Header
RSNR	Reader Serial Number

Appendix D - Version History

D1.1 Document Changes

Version	Author(s)	Date	Description
A.1.20	W Waitz	Jan. 11, 2010	MIFARE Plus, PAY API
A.1.19	S Schwab	July 17, 2009	Chapter 9, supported tags
A.1.18	S Schwab	July 16, 2009	Added footnotes for iCode SL2
A.1.17	W Waitz / L Hanna	May 13, 2009	Review to version 1.16 and error correction
A.0	L Hanna / T Muth	Feb 16, 2009	Updated to HID template
1.14	Werner Waitz	Feb 11, 2008	Extended MIFARE DESFire APDU commands
1.13	Marc Jacquinot	Aug 28, 2007	Minor edits, Reviewed recent changes.
1.12	Werner Waitz	Aug 20, 2007	Add K_{MD0B1} (Default Master Key for application 1 of page 0 on Book 1), MIFARE Emulation Mode and PC/SC 2.01 support for LRI64
1.11	Marc Jacquinot	Nov 22, 2006	Added notes: mandatory Lc in secured mode
1.10	Marc Jacquinot	Aug 18, 2006	FW 5.00, secured communication, finalized document for release
1.01	Abu Ismail	June 30, 2006	Reorganization, adding PC/SC 2.01 support
1.00	Abu Ismail	Feb 08, 2005	Initial Version

D1.2 Firmware History

FW Version	Special Features	Remarks
5.20, 1.75	MIF, MKS, IST, ISE, EMD, HSK,	iCLASS secured mode, HID application read, iCLASS High Security Key supported, EMD Suppression in firmware supported, EMVCo Contactless L1
5.10	MIF, MKS, IST, ISE	iCLASS secured mode, HID application read
5.00	MIF, MKS, IST, ISE	iCLASS secured mode, HID application read
1.03, 1.04	MIF, MKS, IST	iCLASS memory access
1.01, 1.02	MIF, MKS	
1.00	MIF	MIFARE support

D1.2.1 Synchronous Card Special Features

MIF = MIFARE Functionalities

MKS = MIFARE Key Storage

MSK = MIFARE Secured Key Loading

IST = iCLASS Standard Mode Communication

ISE = iCLASS Secured Mode Communication

EMD = Electromagnetic Disturbance

HSK = High Security Key

Appendix E - References

[MIFARE]	MIFARE Data Sheets http://www.nxp.com/acrobat_download2/other/identification/M001053_MF1ICS50_rev5_3.pdf
[MSDNLIB]	Microsoft Developer Network Library; http://msdn.microsoft.com/library/
[DESFIRE]	MIFARE DESFire Data Sheets http://www.nxp.com/documents/data_sheet/MF3ICD21_41_81_SDS.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
[LRI64]	ST Microelectronics datasheet for **LRI64**
[iCODE SL2]	ICODE SL2 Data Sheet http://www.nxp.com/acrobat_download/other/identification/SL113730.pdf