



CardMan 5x21-CL Reader Developer's Guide

Document Version: 1.00

Abstract: This document has been released to assist the developers to develop their application using Omnikey CardMan 5x21-CL readers.

Last modified: 10.02.2006

Disclaimer: Copyright © 2004 by OMNIKEY GmbH

All Rights Reserved.

The information in this document may not be changed without express written agreement of OMNIKEY GmbH.

Table Of Contents

1 Document Information	5
1.1 Change History	5
1.2 Firmware Features	6
1.3 Further Reading	7
2 Introduction	8
2.1 Scope	8
2.2 Terms and Abbreviations	8
2.3 Reader Features	9
3 Installation	10
4 Diagnostic Tool	12
4.1 Usage	12
4.2 Card type and detection order change	13
4.3 Air interface baud rate change	14
5 ATR generation	15
6 Access synchronous cards (Memory cards)	16
6.1 Overview	17
6.2 Functions	17
6.2.1 SCardCLGetUID	17
6.2.2 SCardCLMifareLightWrite	18
6.2.3 SCardCLMifareStdAuthent	18
6.2.4 SCardCLMifareStdDecrementVal	19
6.2.5 SCardCLMifareStdIncrementVal	20
6.2.6 SCardCLMifareStdRead	21
6.2.7 SCardCLMifareStdRestoreVal	21
6.2.8 SCardCLMifareStdWrite	22
6.2.9 SCardCLWriteMifareKeyToReader	23
6.2.10 SCardCLICCTransmit	24
6.3 Accessing Mifare Application Directory (MAD)	26
7 CardMan 5x21-CL Keys	27
7.1 Different keys	27
7.2 Key container	29
7.3 Short description of the keys	30
8 Standard communication with iCLASS Card	32
8.1 APDU structure for Standard communication	32
8.2 Supported INS in the standard communication mode	33
8.2.1 Select page	34

8.2.2	Load Key	36
8.2.3	GetKeySlotInfo	37
8.2.4	Authenticate command:	38
8.2.5	Read command:	39
8.2.6	Update command:	40
8.3	Communication at Standard Mode	41
8.4	Some Remarks on the communication structure at standard mode	42
9	Secured communication with iCLASS Card	44
9.1	Preview of securities in different stages	44
9.1.1	Authenticity between the host and reader	44
9.1.2	The confidentiality of transmitted data in the USB cable	44
9.1.3	The integrity of transmitted data	44
9.1.4	The authenticity between the reader and the card	45
9.1.5	Integrity in the RF transmission	45
9.1.6	Confidentiality in the RF transmission	45
9.1.7	Read/Write session distinction	45
9.1.8	Protection against some known attacks	45
9.2	APDU structure for Secured communication	46
9.2.1	Data Header (DH):	47
9.2.2	Calculation of Signature:	47
9.2.3	Example-Proprietary data structure for one session:	48
9.3	Instructions (INS) for Secured communication	49
9.3.1	Manage Session command:	50
9.3.2	Select page command:	51
9.3.3	Load Key command:	53
9.3.4	Authenticate command:	54
9.3.5	Read command:	55
9.3.6	Update command:	56
9.3.7	GetKeySlotInfo command:	57
9.4	Communication at Secured Mode	58
9.5	Example APDUs for a Session at Secured Mode	59
10	Inter industry commands for synchronous cards	62
11	Performance	63
11.1	Supported baud rates	63
11.2	DESFire Working speed from an example application	63
Appendix A	Application Programming	65
A1	Sample project	65
A1.1	Overview	65
A1.2	Reader Related functions	66
A1.3	Mifare Card Related functions	66

A1.4	ISO 7816 - APDU	66
A1.5	iCLASS Standard Mode	66
A2	Code snippets	67
A2.1	Connect card	67
A2.2	Mifare 1K/4K Authenticate	68
A2.3	Mifare 1K/4K Read/Write	69
A2.4	Mifare 1K/4K Increment/Decrement	70
A2.5	iCLASS Select Page	70
Appendix B	Accessing of iCLASS free zones	72

1.3 Further Reading

[MIFARE] MIFARE Data sheets

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

[DESFIRE] DESFire Data sheets

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

[PCSC] PC/SC Workgroup Specifications 2.1

<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

2 Introduction

2.1 Scope

2.2 Terms and Abbreviations

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

2.3 Reader Features

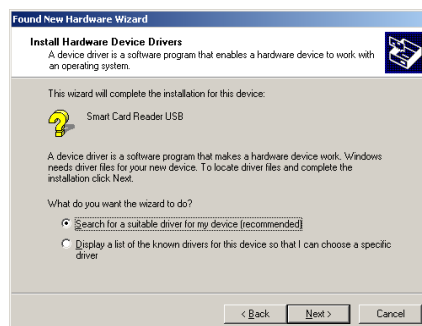
Omnikey CardMan 5x21 is a dual slot reader. Slot(reader) named “CardMan 5x21 n” is the contact slot and “CardMan 5x21-CL n” is the contact-less slot. Where n is slot number (0,1,..).

3 Installation

1. Download the latest driver installation package from <http://www.omnikey.com> for the specific reader.
2. Run this installation package and follow the instructions on screen. This package will extract the driver files to your hard drive. Please remember the location where the files are extracted.
3. Connect the reader to a USB Port of your computer.
4. The “**Found New Hardware Wizard**” will appear. To continue click “**Next**”.



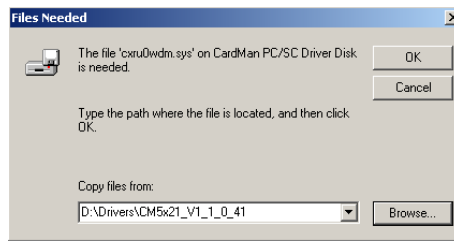
5. In the next dialog chose “**Search for a suitable driver for my device (recommended)**” and click “**Next**”.



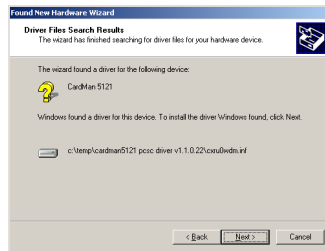
6. Now chose “**Specify a Location**” and click “**Next**”.



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



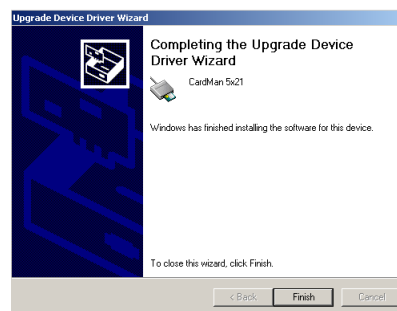
8. If the driver was found in this location press **“Next”** to continue.



9. Sometimes the following dialog will appear, telling that the driver is not digitally signed. It can be accepted by pressing **“Yes”**.



10 If everything worked fine, the following message should appear and the green LED on the reader should be lit up.



Now your reader is ready for use.

If the installation was successful, the green LED on the reader will light up and the reader is listed in the diagnostic tool.

Note: In Windows XP, sometimes a default PC/SC driver with limited functionality may be installed without appearing the “Found New Hardware Wizard”. In this case, update the driver using the Device Manager.

4 Diagnostic Tool

The Diagnostic tool is a small control panel applet. It shows all installed OMNIKEY readers, driver files with version, FW version and some related information. Using this tool the settings for card detection order may be changed.

4.1 Usage

'Diagnostic Tool' can be started from the **Control Panel**. The Tab **General** shows if the 'Resource Manager' is running, which readers are installed and active and the versions of some files, related to the drivers. The Tab **APIs** shows the installed APIs (e.g. synchronous API) and their version numbers.

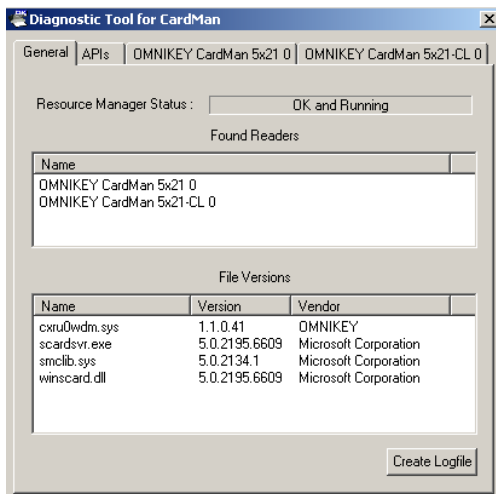


Figure 1: Diagnostic Tool, General Tab

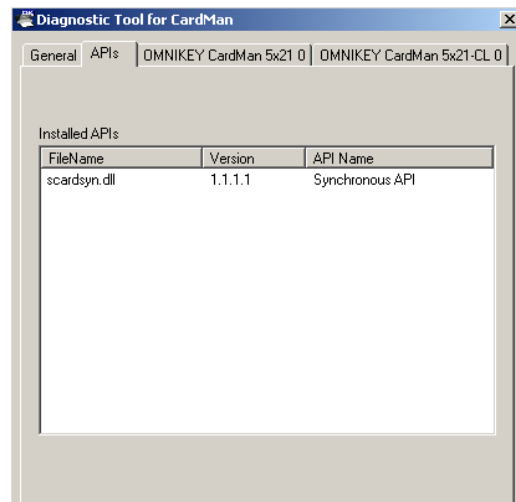


Figure 2: Diagnostic Tool, API Tab

Every installed and active reader has its own tab:

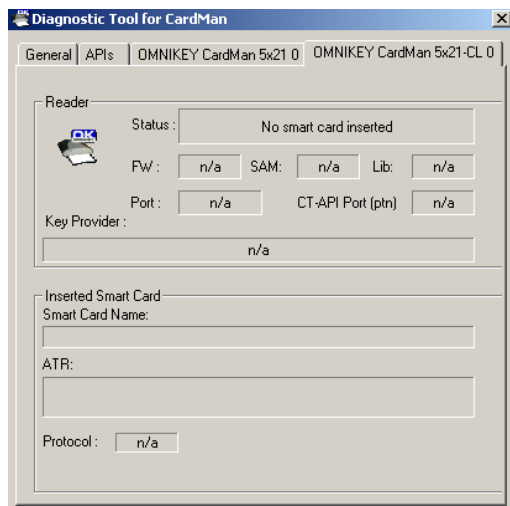


Figure 3: Diagnostic Tool, Reader Tab

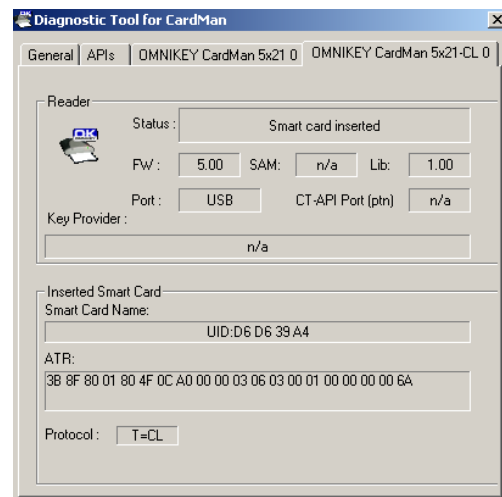


Figure 4: Diagnostic Tool, Reader Tab (Card inserted)

The field **Status** shows if a card is inserted or not and if it is responding. The field **Smart Card Name** gives the name of the smart card and its UID. The fields **ATR** and **Protocol** give the ATR of the card and the Protocol (e.g. T=CL).

4.2 Card type and detection order change

For contactless cards there are many different standards (e.g. ISO14443A, ISO14443B, ISO15694, iCLASS, I-CODE,...), which requires special activation routines and anti-collision algorithms. As there is no physical card switch and information about inserted contact-less card type, the reader is always activating different routines and performing anti-collision for all supported types in a sequence. So detection of a card could take anytime from minimum to maximum time needed for a complete sequence.

To increase the card detection speed, it is possible to deactivate some activation sequences in the reader. For example the detection of a card may be up to six times faster with only one activation sequence instead of six. Additionally the order in which the activation sequences are done can be set.

To activate the RFID settings dialog, use the right mouse button on the title bar of the Diagnostic Tool window to show the system menu. In this menu choose **View->RFID settings**. A new Tab will appear.

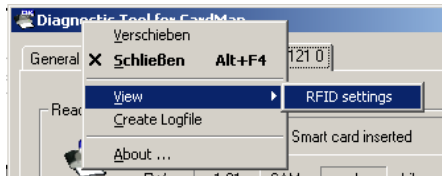


Figure 5: Diagnostic Tool, activate RFID settings

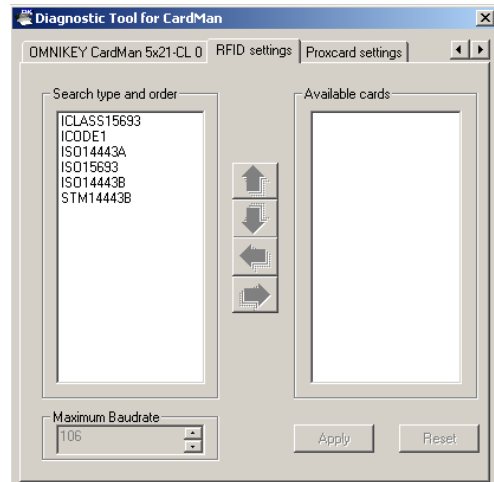






Figure 6: Diagnostic Tool, RFID settings Tab

In this Tab the card types to detect can be chosen. In the left list are the active card types (If the list is empty, all card types are activated). You can choose card types from the right list and put them in the left list using the button . To deactivate card types, choose them from the left list and press the button . With the button  and  the order of the card types in the active list can be changed.

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

Note: The reader always looks for the last active card type until a card of the newly activated card type is detected. E.g. ISO14443A was the only active card type, which is changed to be ISO14443B only active. Now ISO14443A cards are detected until the first ISO14443B card is presented to the reader.

4.3 Air interface baud rate change

For ISO 14443 type card the air interface could be different e.g. 106 kbps, 212 kbps, 424 kbps and 847 kbps. By default the air interface is 424 kbps for ISO 14443 card, but it can be changed from the diagnostic tool.

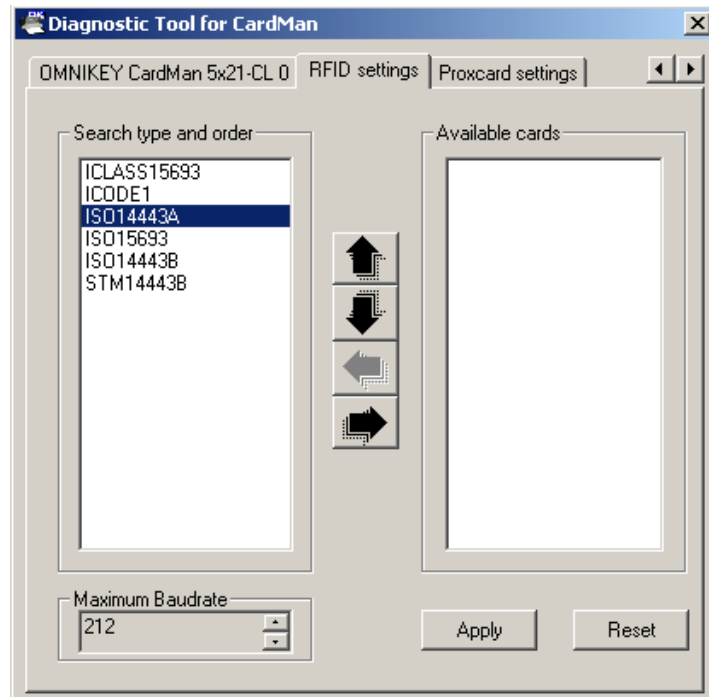


Figure 7: Diagnostic Tool, Baud rate change

To change the baud rate select the card type (ISO14443A or ISO14443B) and change the 'Maximum Baudrate' field. Finalize your setting by pressing 'Apply' button

5 ATR generation

ATR Generation according to PCSC part 3 v2.1

6 Access synchronous cards (Memory cards)

For the access of synchronous cards we provide the synchronous API. ScardCL is a part of this API, which deals with contactless memory cards. For the time being, the complete functionality of the following RF interface cards is supported by the **scardsyn.dll** shared library:

- Mifare Standard 1K
- Mifare Standard 4K
- Mifare Ultra Light

For adequate understanding of the card's functionality please refer to the Mifare Data sheet.

The following functions are available in the SCardCL module within the library:

- **SCardCLGetUID** may be used to get the uid or snr of present card in RF-field.
- **SCardCLMifareStdAuthent**
- **SCardCLMifareStdRead**
- **SCardCLMifareStdWrite**
- **SCardCLMifareLightWrite**
- **SCardCLMifareStdIncrementVal**
- **SCardCLMifareStdDecrementVal**
- **SCardCLMifareStdRestoreVal**
- **SCardCLWriteMifareKeyToReader**
- **SCardCLICCTransmit**

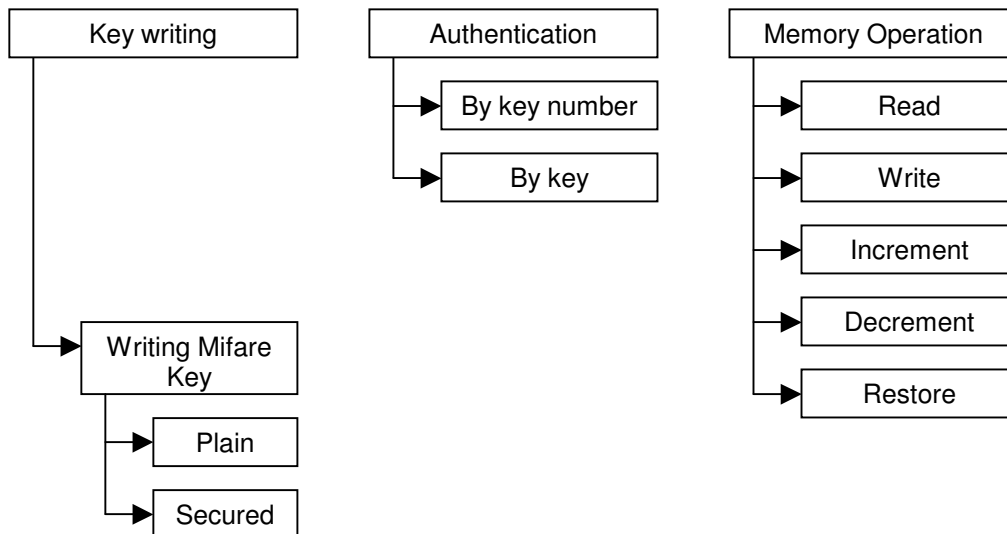
The functions **SCardCLWriteMifareKeyToReader** are for writing the Mifare keys to the reader. This function does not care either the card is present, connected or not. The other functions are completely card related.

The function **SCardCLICCTransmit** is the gateway to communicate between the synchronous card and the application.

Currently it supports only the iCLASS card. So in this document SCardCLICCTransmit function dedicates to the iCLASS standard and secured mode communications.

6.1 Overview

The functionality for Mifare cards, implemented in Omnikey CardMan 5x21-CL reader can be classified as follows:



6.2 Functions

In the following chapters, the functions of ScardCL are described.

6.2.1 SCardCLGetUID

```

OKERR ENTRY SCardCLGetUID
(
    IN SCARDHANDLE ulHandleCard,
    IN OUT PCHAR pucUID,
    IN ULONG ulUIDBufLen,
    IN OUT PULONG pulnByteUID
);
  
```

The function SCardCLGetUID gives the UID and number of bytes in the UID of the contactless card present in the rf field. For cards do not following the T = CL (no ATS), the ATR field also filled with the UID. UID starts from the 4th byte of ATR. One must know the number of valid bytes of UID for the specific card.

The following parameters need to be provided:

Parameter	Type	Description
UIHandleCard		Handle provided from the "smart card resource manager" after connecting the card (SCardConnect)
PucUID		Unique Identifier of the contactless card for ISO 14443A card, the last byte is BCC (UID CLn check byte, calculated as exclusive-or over the 4 previous bytes), actual UID will be without the last byte.
UIUIDBufLen		The length of the ucUID buffer
PulnByteUID		Number of bytes in the UID

6.2.2 SCardCLMifareLightWrite

```
OKERR ENTRY SCardCLMifareLightWrite
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PUCCHAR pucMifareDataWrite,
    IN ULONG ulMifareDataWriteBufLen
);
```

The function SCardCLMifareLightWrite writes the block of the Mifare Ultra Light card only, it does not need authentication. Four bytes data can be written to the specified block. Block 0, 1 and first two bytes of block 2 cannot be written. Other two bytes of block 2 and block 3 needs extra care, as they are OTP and LOCK bytes.

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare ultra Light card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The Mifare block number which is to be written, it is 2 to 15.
pucMifareDataWrite		A pointer to the 4 byte buffer, which data to be written
ulMifareDataWriteBufLen		The length of the data buffer, it must be 4

6.2.3 SCardCLMifareStdAuthent

```
OKERR ENTRY SCardCLMifareStdAuthent
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN UCHAR ucMifareAuthMode,
    IN UCHAR ucMifareAccessType,
    IN UCHAR ucMifareKeyNr,
    IN PUCCHAR pucMifareKey,
    IN ULONG ulMifareKeyLen
);
```

The function SCardCLMifareStdAuthent authenticates the block of Mifare 1K or Mifare 4K Cards. The authentication can be done in two ways either supplying the key or the Key number which is already in the reader. The ways can be selected by using ucMifareAccessType. If it is MIFARE_KEY_INPUT then ucMifareKeyNr has no role, on the other hand if it is MIFARE_KEYNR_INPUT then pucMifareKey and ulMifareKeyLen have no effect.

A virgin Mifare card may have the following default key

Key A: FFFFFFFFFF, Key B: FFFFFFFFFF

Key A: A0A1A2A3A4A5, Key B: B0B1B2B3B4B5

The Mifare cards supplied with the developer's kit are virgin Mifare cards.

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The block number which is to be authenticated, for Mifare 1K it will be 0 to 63, and for Mifare 4K it will be 0 to 255.
ucMifareAuthMode		This tells the reader the key you want to use is Mifare Key A or Mifare Key B, For Key A this parameter must be set to MIFARE_AUTHENT1A and for Key B to MIFARE_AUTHENT1B
ucMifareAccessType		The ways of supplying key or providing the key number
ucMifareKeyNr		If key number is to be supplied, then valid key number, make sure that the key has been written in the specified KeyNr using function SCardCLWriteMifareStdKeyToRC
pucMifareKey		A pointer to the six byte Mifare key, if intended to supply key
ulMifareKeyLen		If key is supplying then length of the key, it is always 6

6.2.4 SCardCLMifareStdDecrementVal

```
OKERR ENTRY SCardCLMifareStdDecrementVal
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PUCCHAR pucMifareDecrementValue,
    IN ULONG ulMifareDecrementValueBufLen
);
```

The function SCardCLMifareStdIncrementVal decrements the value block's content of Mifare 1K and Mifare 4K cards. The block must be authenticated by calling the function SCardCLMifareStdAuthent prior to calling SCardCLMifareStdDecrementVal, if the block is not authenticated it will return ERROR_ACCESS_DENIED. Sector trailer block and block 0 cannot be Decrementated. If the block is not Value block the decrement will also fail. One can be sure of the block either value block or not by simply reading the block and checking the data storage according to Mifare Value Block definition,(Please see the Mifare Data Sheet).

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare 1K or Mifare 4K card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The Mifare value block number which is to be decremented. for Mifare 1K it can be 1 to 63 and for Mifare 4K it can be 1 to 255, except the sector trailer block and unless it is not value block.
pucMifareDecrementValue		A pointer to the 4 byte buffer, by which the content of the block will be decremented.
ulMifareDecrementValueBufLen		Size of the Buffer, it must be 4.

6.2.5 SCardCLMifareStdIncrementVal

OKERR ENTRY SCardCLMifareStdIncrementVal

```
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PUCCHAR pucMifareIncrementValue,
    IN ULONG ulMifareIncrementValueBufLen
);
```

The function SCardCLMifareStdIncrementVal increments the value block's content of Mifare 1K and Mifare 4K cards. The block must be authenticated by calling the function SCardCLMifareStdAuthent prior to calling SCardCLMifareStdIncrementVal, if the block is not authenticated it will return ERROR_ACCESS_DENIED. Sector trailer block and block 0 cannot be Incremented. If the block is not value block the increment will also fail and card will leave current state, should be reconnected the card. One can be sure of the block either value block or not by simply reading the block and checking the data storage according to Mifare Value Block definition,(Please see the Mifare Data Sheet).

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare 1K or Mifare 4K card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The Mifare value block number which is to be incremented. for Mifare 1K it can be 1 to 63 and for Mifare 4K it can be 1 to 255, except the sector trailer block and unless it is not value block.
pucMifareIncrementValue		A pointer to the four byte buffer by which the value will be incremented
ulMifareIncrementValueBufLen		The number of byte in the buffer, it must be 4

6.2.6 SCardCLMifareStdRead

```
OKERR ENTRY SCardCLMifareStdRead
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN OUT PUCCHAR pucMifareDataRead,
    IN ULONG ulMifareDataReadBufLen,
    IN OUT PULONG pulMifareNumOfDataRead
);
```

The function SCardCLMifareStdRead reads the block of Mifare 1K, Mifare 4K or Mifare Ultra Light Cards. If the card is Mifare Ultra Light, it does not need authentication, otherwise it must be authenticated by calling the function SCardCLMifareStdAuthent prior to calling SCardCLMifareStdRead, if the block is not authenticated it will return ERROR_ACCESS_DENIED. If the operation is successful, sixteen byte data will be available in the pucMifareDataRead buffer. In case of Mifare Ultra light the sixteen byte data will be from four blocks starting from the ulMifareBlockNr. A roll back is implemented e.g. if ulMifareBlockNr is 14, the contents of 14, 15, 0, and 1 is read. The Mifare Ultra Light has 16 blocks (0 to 15).

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The Mifare block number which is to be read, for Mifare 1K it is 0 to 63, for Mifare 4K it is 0 to 255 and for Mifare Ultra Light it is 0 to 15.
pucMifareDataRead		Pointer to the buffer allocated for data reading from the card
ulMifareDataReadBufLen		The size of the buffer, must be 16 or higher
pulMifareNumOfDataRead		It will return the number of bytes received from the card, it will be always 16 if the reading is successful

6.2.7 SCardCLMifareStdRestoreVal

```
OKERR ENTRY SCardCLMifareStdRestoreVal
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareOldBlockNr,
    IN ULONG ulMifareNewBlockNr,
    IN BOOLEAN fMifareSameSector,
    IN UCHAR ucMifareAuthModeForNewBlock,
    IN UCHAR ucMifareAccessTypeForNewBlock,
    IN UCHAR ucMifareKeyNrForNewBlock,
    IN PUCCHAR pucMifareKeyForNewBlock,
    IN ULONG ulMifareKeyLenForNewBlock
);
```

The function SCardCLMifareStdRestoreVal restores the data of one value block to another value block. If one of them are not value block, then it will fail. One can be sure by reading both blocks and checking the data storage according to Mifare Value Block definition, (Please see the Mifare Data Sheet) before calling this function. Just before calling function SCardCLMifareStdRestoreVal the source block ulMifareOldBlockNr must be authenticated, otherwise it will return ERROR_ACCESS_DENIED. If source and destination block both are in the same sector, then set fMifareSameSector TRUE otherwise FALSE, for standard value block configuration. If the Destination

block is in other sector, by setting the fMifareSameSector FALSE, the succeeding parameters has to be provided.

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare 1K or Mifare 4K card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareOldBlockNr		The source block number from where values will be transferred
ulMifareNewBlockNr		The destination block number to where the values will be taken
fMifareSameSector		Identify if the source and destination block both are in the same sector or not, if in the same sector set TRUE, else FALSE
ucMifareAuthModeForNewBlock		If different sector, the authentication mode for new sector, If Key A is used then MIFARE_AUTHENT1A, if Key B then MIFARE_AUTHENT1B
ucMifareAccessTypeForNewBlock		The ways of supplying key information, providing the key or Key number for the destination block
ucMifareKeyNrForNewBlock		If key number is to be supplied, then valid key number for destination block, Make sure that the key has been written in the specified KeyNr using function SCardCLWriteMifareStdKeyToRC
pucMifareKeyForNewBlock		A pointer to the six byte mifare key, if intended to supply key
ulMifareKeyLenForNewBlock		If key is supplying then length of the key, it is always 6

6.2.8 SCardCLMifareStdWrite

```
OKERR ENTRY SCardCLMifareStdWrite
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PUCCHAR pucMifareDataWrite,
    IN ULONG ulMifareDataWriteBufLen
);
```

The function SCardCLMifareStdWrite Writes the block of the Mifare card If the card is Mifare Ultra Light, it does not need authentication, otherwise it must be authenticated by calling the function SCardCLMifareStdAuthent prior to calling SCardCLMifareStdWrite, if the block is not authenticated it will return ERROR_ACCESS_DENIED. For Mifare 1K and Mifare 4K block 0 cannot be written. To write a sector trailer please take extra care, incorrect configuration can make permanent loss of the blocks. If the mod(ulMifareBlockNr + 1, 4) is 0, then ulMifareBlockNr is a sector trailer. For Mifare 4k this could be different for higher block numbers. Please see the Mifare Data Sheet.

If the card is Mifare Ultra light then block 0, 1 and first two bytes of block 2 cannot be written. Other two bytes of block 2 and block 3 needs extra care as they are OTP and LOCK bytes. Although 16 bytes is supplied to the cards, only first four bytes will be written in the given ulMifareBlockNr.

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle to a Mifare card, provided from the "smart card resource manager" after connecting the card (SCardConnect)
ulMifareBlockNr		The Mifare block number which is to be written, for Mifare 1K it is 1 to 63, for Mifare 4K it is 1 to 255 and for Mifare Ultra Light it is 2 to 15.
pucMifareDataWrite		The pointer to a 16 byte buffer, the data to be written
ulMifareDataWriteBufLen		The length of the data buffer, it must be 16

6.2.9 SCardCLWriteMifareKeyToReader

OKERR ENTRY SCardCLWriteMifareKeyToReader

```
(
    IN SCARDHANDLE ulHandleCard,
    IN SCARDCONTEXT hContext,
    IN PCHAR pcCardReader,
    IN ULONG ulMifareKeyNr,
    IN ULONG ulMifareKeyLen,
    IN PUCCHAR pucMifareKey,
    IN BOOLEAN fSecuredTransmission,
    IN ULONG ulEncryptionKeyNr );
```

The function SCardCLWriteMifareKeyToReader writes the Mifare keys in the reader. These keys are rewrite-able but cannot be read back. These keys will be used for Mifare Authentication, if one intends to do so. Maximum 32 keys (ulMifareKeyNr 0 to 31) can be stored. In order to write without connecting card, ulHandleCard is set to invalid e.g. 0x00000000 or 0xFFFFFFFF, and the user must provide the hContext and pcCardReader. If valid ulHandleCard is provided, hContext and pcCardReader have no role to play, card is considered connected. If fSecuredTransmission is set TRUE, then 6-byte Mifare key (2-byte padding "8000") has to be 3-DES encrypted by using one of the two 16-byte customer keys which have key number 0x80 or 0x81. One or both of the keys may be collected from Omnikey with an NDA. These customer keys can be updated in secured mode communication described in the developer's guide. *The following parameters need to be provided:*

Parameter	Type	Description
ulHandleCard		Handle, provided from the "smart card resource manager" after connecting the card (SCardConnect)
hContext		current context handle, received from SCardEstablishContext
pcCardReader		ptr to a str holding the name of the selected reader
ulMifareKeyNr		Identity of the key (it must be any value from 0 to 31, maximum 32 keys can be stored)
ulMifareKeyLen		Length of the key which will be written, must be 6 if not secured, if secured it must be 8
pucMifareKey		if not secured Six-byte Mifare key, if secured then 8 byte 3-DES encrypted key
fSecuredTransmission		The way of transmission of MifareKey from host pc to reader, secured or plain
ulEncryptionKeyNr		If the transmission is secured, then define the Encryption key Number which has been used for 3-DES encryption of MifareKey, this must be 0x80 or 0x81

6.2.10 SCardCLICCTransmit

```

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

```

OMNIKEY CardMan 5x21-CL reader supports the complete functionality of the Inside contactless card including the HID application on iCLASS cards. There are two mode of communication between the card and the application:

- Standard mode communication
- Secured mode communication

The following chapters describe the know-how of these two types of communication.

SCardCLICCTransmit is the function, establishing communication between the iCLASS card and host in both modes standard and secured.

The parameters must be in accordance to the APDU described in the following chapters.

The following parameters need to be provided:

Parameter	Type	Description
ulHandleCard		Handle, provided from the "smart card resource manager" after connecting the card (SCardConnect)
pucSendData		Pointer to the buffer of data send to reader, this data buffer must be according to table 7-1
ulSendDataBufLen		Length of the send buffer
pucReceivedData		A buffer provided to receive the response according to table 7.2
pulReceivedDataBufLen		Length of the provided buffer is given, it returns the actual received bytes

Table 6-1: Data gram, application to reader

CLA	INS	P1	P2	Lc	Send data gram ***	Le
0x8X	XX	XX	XX	XX	xxxxxxxxxxxxxxxxxxxxxxxxxxxx (Lc bytes)	xx

Table 6-2: Data gram, reader to application

Response data gram ***	SW1	SW2
XXXXXXXX	XX	XX

*** In the secured communication mode, this Send data gram and Response data gram will be in Omnikey proprietary format according to section 9-2. In the standard mode just the 'data in' and 'data out'.

Table 6-3: Common status codes

	SW1	SW2	Meaning
No Error	'90'	'00'	Success
Error	'64'	'00'	Card execution error
	'67'	'00'	Wrong length
	'68'	'00'	Class byte is not correct
	'69'	'82'	Security Status not satisfied Includes wrong data structure, wrong keys, incorrect padding.
	'6A'	'81'	INS not supported
	'6B'	'00'	Wrong parameter P1-P2

The error code defined in Table 7-3 is valid for all the commands. Moreover the command specific error codes have been introduced in every command sections.

Note: If there is a '6982' error occurred in the secured communication, the system denies processing of any further commands due to security reason. In this case removing the card from the field is necessary.

The access to the HID Application is *only* allowed in the secured mode communication with specially HW configured readers. HID Application is read only, can never be written to this application.

Secured mode communication is supported by the FW version 500 0r more.

6.3 Accessing Mifare Application Directory (MAD)

Right now, there is no function in the API, which can present the complete MAD. But the MAD can be accessed using the **SCardCLMifareStdAuthent** and **SCardCLMifareStdRead**. To access the complete MAD follow the steps:

- Authenticate block 3 with the Public key and authentication mode A
- Read Block 3
- Read Block 2
- Read Block 1

The public key for MAD is "A0A1A2A3A4A5"

For complete understanding of MAD one may collect the M001824.pdf document from Philips.

7 CardMan 5x21-CL Keys

Some keys with unique key name and key number (0x00 to 0xFE) have been introduced in the reader to support the security offered by different cards and the reader itself.

7.1 Different keys

The key number inherently fixes the key length. That means Key 0x00 will be always 6 bytes, Key 0x20 always 8 bytes and so on. 0xFF is not a valid key. Key number 0x23 to 0x31 are the keys for iCLASS free zones.

Table 7-1: Different keys of CardMan 5x21-CL reader

Key Number.	Key Name	Key Length	Key Type	Memory Type
0x00 to 0x1F are for all 6-byte Mifare key				
0x00 to 0x1F	K _{MIF0} (Mifare Key 0) to K _{MIF31} (Mifare Key 31)	6 bytes	Card Key	Non volatile memory
0x20 to 0x7F are for all 8-byte key				
0x20	K _{IAMC} (Any Inside Application Master key)	8 bytes	Card Key	Non volatile memory
0x21	K _{MDC} HID Master Key Current; (K _{MDO} , K _d for application 1 of page 0 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x22	K _{MDO} (Old HID Master Key, internally used, currently not used, RFU)	8 bytes	Card Key	Non volatile memory
0x23	K _{MC0} (Default Master Key for application 2 of page 0 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x24	K _{MD1} (Default Master Key for application 1 of page 1 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x25	K _{MC1} (Default Master Key for application 2 of page 1 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x26	K _{MD2} (Default Master Key for application 1 of page 2 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x27	K _{MC2} (Default Master Key for application 2 of page 2 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x28	K _{MD3} (Default Master Key for application 1 of page 3 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x29	K _{MC3} (Default Master Key for application 2 of page 3 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x2A	K _{MD4} (Default Master Key for application 1 of page 4 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x2B	K _{MC4} (Default Master Key for application 2) of page 4 of iCLASS card	8 bytes	Card Key	Non volatile memory

0x2C	K _{MD5} (Default Master Key for application 1 of page 5 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x2D	K _{MC5} (Default Master Key for application 2 of page 5 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x2E	K _{MD6} (Default Master Key for application 1 of page 6 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x2F	K _{MC6} (Default Master Key for application 2 of page 6 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x30	K _{MD7} (Default Master Key for application 1 of page 7 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x31	K _{MC7} (Default Master Key for application 2 of page 7 of iCLASS card)	8 bytes	Card Key	Non volatile memory
0x32	K _{MTD} (Master Transport Key for application 1 of iCLASS card, key stored at chip production)	8 bytes	Card Key	Non volatile memory
0x33	K _{MTC} (Master Transport Key for application 1 of iCLASS card, key stored at chip production))	8 bytes	Card Key	Non volatile memory
0x80 to 0xAF are for all 16-byte key.				
0x80	K _{CUR} (Customer read key)	16 bytes	Reader Key	Non volatile memory
0x81	K _{CUW} (Customer write Key)	16 bytes	Reader Key	Non volatile memory
0x82	K _{ENC} (Card data encryption key)	16 bytes	Card Key	Non volatile memory
0xB0 to 0xCF are 24- byte key				
0xD0 to 0xDF are 32-Byte key				
0xF0 to 0xFF are volatile key				
0xF0	K _{VAK} (volatile application key)	8 bytes	Card Key	Volatile memory

Note: Key number 0x21 to Key number 0x31 (except 0x22) are the default keys for an iCLASS card.

Key number 0x32 and 0x33 are the default transport key of Inside industry configuration. They are fixed in the reader cannot be upgraded, can be used only in the authentication of the application.

FW version 500 or more support all the keys. FW version 103 and 104 support only key number 0x20 and 0xF0

7.2 Key container

CardMan 5x21-CL reader offers a key container organized by slots with fixed length. These slots store the pre-defined keys. To make the design simpler, the controlling of key storages in the slots has been managed by the reader internally. User just needs to know the key number to load or use a key. The reader manages the key retrieval from or key insertion in the container by itself. Reading the content of the container is protected. To make a full-proof security, the keys are diversified using two 16-byte secrets before storing them in the container.

Table 7-2: Key container of CardMan 5x21-CL reader

Key Slot (KS) Number	KS Length	Default Stored Key Name	Default Stored Key Number	Remarks
0x00	12	K _{MIF0}	0x00	There will be no key slot information available for these key slots. Retrieving information will return SW1SW2 "6300".
....	12	-----	----	
0x1F	12	K _{MIF31}	0x1F	
0x20	16	K _{CUR}	0x80	Key slot information is available.
0x21	16	K _{CUW}	0x81	
0x22	16	K _{ENC}	0x82	
0x23	08	K _{IAMC}	0x20	
0x24	08	K _{MDO}	0x22	
0x25	08	K _{MDC}	0x21	
0x26	08	K _{VAK}	0xF0	
0x27	08	K _{MC0}	0x23	Key slot information is available.
0x28	08	K _{MD1}	0x24	
0x29	08	K _{MC1}	0x25	
0x2A	08	K _{MD2}	0x26	
0x2B	08	K _{MC2}	0x27	
0x2C	08	K _{MD3}	0x28	
0x2D	08	K _{MC3}	0x29	
0x2E	08	K _{MD4}	0x2A	
0x2F	08	K _{MC4}	0x2B	
0x30	08	K _{MD5}	0x2C	
0x31	08	K _{MC5}	0x2D	
0x32	08	K _{MD6}	0x2E	
0x33	08	K _{MC6}	0x2F	
0x34	08	K _{MD7}	0x30	

0x35	08	K _{MC7}	0x31
0x36	08	K _{MTD}	0x32
0x37	08	K _{MTC}	0x33

For the advanced user a function is provided to get the information of the key slots. Nevertheless this is an option designed for future use to ensure the uses of a slot for multiple keys if there are more keys than key slots.

7.3 Short description of the keys

Table 7-3: Short description of the different keys

Key Name	Key Number	Updating criteria	Description
K _{MIF0} to K _{MIF31}	0x00 to 0x1F		These keys can be loaded/ upgraded by using SCardCLWriteMifareKeyToReader function of synchronous API in. The key sent to reader may be plain or 3-DES encrypted with the K _{CUR} or K _{CUW}
K _{IAMC}	0x20	Always, Standard Mode Secured Mode - Read session - Write session	This key can be loaded with any 8-byte value and can be used to authenticate any application of the iCLASS card. Default reader has here the transport Kd0. (Key for application 1 (of page 0)) of Inside contact-less card.
K _{MDC}	0x21	Never	This key is used to authenticate the HID application of the iCLASS card. The authentication of the application using this key is only allowed in secured mode. If the application is authenticated with this key, writing to the application is not allowed.
K _{MDO}	0x22	Never	Reader Internal, RFU
K _{CUR}	0x80	In the secured mode's read/write session only	Authentication of reader (to established a secured session) by using this key gives the application read-only right. This key can also be used to encrypt the Mifare key in SCardCLWriteMifareKeyToReader function .
K _{CUW}	0x81	In the secured mode's read/write session only	Authentication of reader (to established a secured session) by using this key gives the application read/write right. This key can also be used to encrypt the Mifare key in SCardCLWriteMifareKeyToReader function .
K _{ENC}	0x82	In the secured mode's read/write session only	It is used to encrypt the data will be written to the card or is used to decrypt the data read from the card if the specific bits are set in the read or update INS. If the bits are set for DES, first 8-byte is used, If the bits are set for 3-DES, all 16-byte is used.
K _{VAK}	0xF0	Always, Standard Mode Secured Mode - Read session - Write session	This key can be used to authenticate any application of 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 -} K _{MC7,}	0x23 to 0x31	Never	These are the iCLASS default keys for free zones. May be used to authenticate the applications of iCLASS card other

K _{MD1} - K _{MD7} ,			than the HID application.
K _{MTD} - K _{MTC} ,	0x32 0x33	Never	These are the iCLASS transport keys. The keys are set by the chip manufacturer.

8 Standard communication with iCLASS Card

This type of communication does not provide any authentication, confidentiality and integrity between the host and reader. The security in the reader to card communication as well as the card data integrity and confidentiality depend on the card technology.

8.1 APDU structure for Standard communication

The supported APDU is standard ISO7816-4 APDU.

Table 8-1: APDU application to reader

CLA	INS	P1	P2	Lc	Data in	Le
0x80	XX	XX	XX	XX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	xx

Table 8-2: APDU, reader to application

Data out	SW2	SW1
XXXXXXX	xx	xx

Please note that, APDU application to reader is the pucSendData and the APDU reader to application will be in the pucReceivedData of the synchronous function SCardCLICCTransmit.

8.2 Supported INS in the standard communication mode

The following instruction (INS) commands are supported by Omnikey CardMan 5x21-CL reader in the standard communication mode.

Table 8-3: Instructions

Instruction	Description
0xA6	Select Page
0x82	Load Key
0xC4	GetKeySlotInfo
0x88	Authenticate
0xB0	Read
0xD6	Update

8.2.1 Select page

For the iCLASS 2x8KS type card the required page of the card must be selected at first, if it is other than page 0. For other type of cards this command is not necessary to perform, but could be performed with the correct combination of P1, P2, Lc, Le to retrieve the supported information according to P2.

Table 8-4: Select page command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Select page	0x80	0xA6	xx	xx	xx	xxxx	xx

Table 8-5: Data Out of Select page command

Data Out	
xx	SW1 SW2

Lc: Absent for encoding Nc = 0, present for encoding Nc > 0

Data in: Absent or Page number (according to P1)

Le: If any data field is requested according to P2 and (Le = 0x00 or Le= 0x08) then 8-byte data is returned according to P2.

Table 8-6: P1 of Select page Command

b7	b6	b5	b4	b3	b2	b1	b0	Meaning	Command data field
0	0	0	0	0	0	0	0	Select the only page of iCLASS 2KS or single page of 16KS	Absent
0	0	0	0	0	0	0	1	Select page of multi-page iCLASS 16KS (8x2KS) or 32KS	Page number **
Other values are RFU									

** The data field (1 byte page number) is as follows:

Table 8-7: Page number

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	Book number	0	Page number 0-7		

For iCLASS 16KS (8x2KS) or first book (8x2KS or 16KS) of iCLASS 32KS, the book number is always 0. For second book, Book number is 1.

An example chart is as follows:

Table 8-8: Example of page number

Item	Page number
Select page 3 of an iCLASS 8x2KS card	0x03
Select page 3 of book 0 of an iCLASS 32KS (book 0: 8x2KS) card	0x03
Select page 3 of book 1 of an iCLASS 32KS (book 1: 8x2KS) card	0x13
Select book 1 (16KS) of an iCLASS 32KS	0x10
Select book 0 (16KS) of an iCLASS 32KS	0x00

Table 8-9: P2 of Select page Command

b7	b6	b5	b4	b3	b2	b1	b0	Meaning
0	0	0	0	0	0	0	0	Return nothing
0	0	0	0	0	1	0	0	Return 8-byte Card serial number
0	0	0	0	1	0	0	0	Return 8-byte configuration block data
0	0	0	0	1	1	0	0	Return 8-byte application issuer data
Other values are RFU								

Table 8-10: SW1SW2 for Select page command

	SW1	SW2	Meaning
Error	62	83	Requested page number does not exist
	6C	XX	Wrong length Le. XX returns the number of data available

8.2.2 Load Key

Load Key command loads the K_{IAMC} in the reader memory. If 'Authenticate' command wants to use the K_{IAMC} , it must be in the reader key container or in the non-volatile memory. The volatile key may be used only for the succeeding Authentication. Right now only one inside application master key (K_{IAMC}) can be stored in the reader.

Table 8-11: Load Key command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Load Key	0x80	0x82	Key Structure	Key number	Key Length	Key	-

Table 8-12: Data Out

Data Out	
-	SW1 SW2

Table 8-13: Definition of P1 of Load Key command APDU

b7	b6	b5	b4	b3	b2	b1	b0	Description
x								0 card key, 1 Reader key
	X							0:Plain transmission, 1:Secured transmission
		x						If 0, the keys are loaded in the IFDs volatile memory If 1, the keys are loaded in the IFDs nonvolatile memory.
			x					RFU (set 0, else return error)
				0000				Not Valid (set all 0, else return error)

Note: b6 must be set to 0 as no encryption of key only

P2 (Key Number): The key number is according to table 8-1.

The following table introduces some examples of SW1SW2 and their meaning.

Table 8-14: Load Keys command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
Error	'63'	'81'	Loading/Updating is not allowed
		'82'	Card key not supported
		'83'	Reader key not supported
		'84'	Plain transmission not supported
		'85'	Secured transmission not supported
		'86'	Volatile memory is not available
		'87'	Non volatile memory is not available
		'88'	Key number not valid
	'89'	Key length is not correct	

8.2.3 GetKeySlotInfo

Omniquey CardMan 5x21-CL reader has some predefined Key Slots in the key container. The user can only load key by using the key number. IFD decides where to store (in which slot) the key. But user can get the status of the key slots by using this GetKeySlotInfo command.

Table 8-15: GetKeySlotInfo Command

Command	Class	INS	P1	P2	Lc	Data in	Le
GetKeySlotInfo	0x80	0xC4	0x00	KeySlot number according to table 8-2	Absent	Absent	xx

Le:

If Le = 0x00 or Le = 0x02, 2-byte information is returned else error ('6C02') is returned.

Table 8-16: GetKeySlotInfo Command Response

Data Out	
	SW1 SW2

Reader Response will be 2 Bytes according to the following table:

Table 8-17: Key Information Byte

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
RFU						Access Type		Key Number according to table 8-1							

Note: Key number 0xFF, means the slot is empty.

Table 8-18: Access option

b9	b8	Access Option
0	0	Can be loaded in any type of transmission e.g. plain or secured
0	1	Allowed to load only in Omniquey proprietary secured mode
1	0	Loading is never allowed
1	1	RFU

Table 8-19: GetKeySlotInfo command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is available
	'63'	'01'	Key slot does not contain valid key/empty
Error	62	83	Requested Key slot does not exist
	6C	XX	Wrong length Le < requested data, XX returns the number of data available

8.2.4 Authenticate command:

Authenticate command authenticates the application of the selected page.

Table 8-20: Authentication command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Authenticate	0x80	0x88	Key Type	Key Nr	Address Length	Address.	-

Table 8-21: Data out of Authenticate command

Data Out	
-	SW1 SW2

P1:

Table 8-22: Key Type

Value (b7 - b0)	Description
0x00	Inside Contactless Kd
0x01	Inside Contactless Kc
0x60	Mifare Key A
0x61	Mifare Key B
0xFF	Key Type unknown or not necessary
Other values	RFU

Note: Right now only supported for iCLASS card. Valid P1 = 0x00 or 0x01.

Key Nr.:

The card key number, which will be used for this authentication according to table 8-1

Lc: As the page has been selected in the select command and the memory authentication of iCLASS card does not need any address, so Lc and Data in must be absent. In case of other card it can be maximum 2 bytes.

Data in : As Lc Absent, Data in must be absent.

Table 8-23: Authentication command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
Error	'69'	'83'	Authentication cannot be done
		'84'	Reference key not useable
		'88'	Key number not valid

8.2.5 Read command:

Read command reads the data from the given block address.

Table 8-24: Read command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Read	0x80	0xB0	Block Nr MSB	Block Nr LSB	---	--	xx

Table 8-25: Read Command Response

Data Out	
Card response	SW1 SW2

For iCLASS card, please set P1 as 0x00 and P2 as the block number

Le:

If Le = 0x08 or 0x00, 8 bytes are returned starting from block address offset.

If Le = 0x20, 32 bytes are returned starting from block address offset (if the card supports reading 32 bytes command).

Other values of Le return error '6C08'

Table 8-26: Read Binary error codes

	SW1	SW2	Meaning
Warning	'62'	'81'	Part of returned data may be corrupted
		'82'	End of file reached before reading Ne bytes
Error	'69'	'81'	Command incompatible
		'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 Ne; 'XX' encodes the exact number)

Note: Reading the blocks without authenticating the corresponding application will return all "FF". Reading the blocks, which are not allowed to read, will return all "FF"

8.2.6 Update command:

Update command updates the given block number with the given data.

Table 8-27: Update command APDU

Command	Class	INS	P1	P2	Lc	Data in	Le
Update	0x80	0xD6	Block Nr. MSB	Block Nr. LSB	xx	xxxx	--

Table 8-28: Update Response

Data Out	
Data	SW1 SW2

For iCLASS please set P1 as 0x00 and P2 as the block number

Lc:

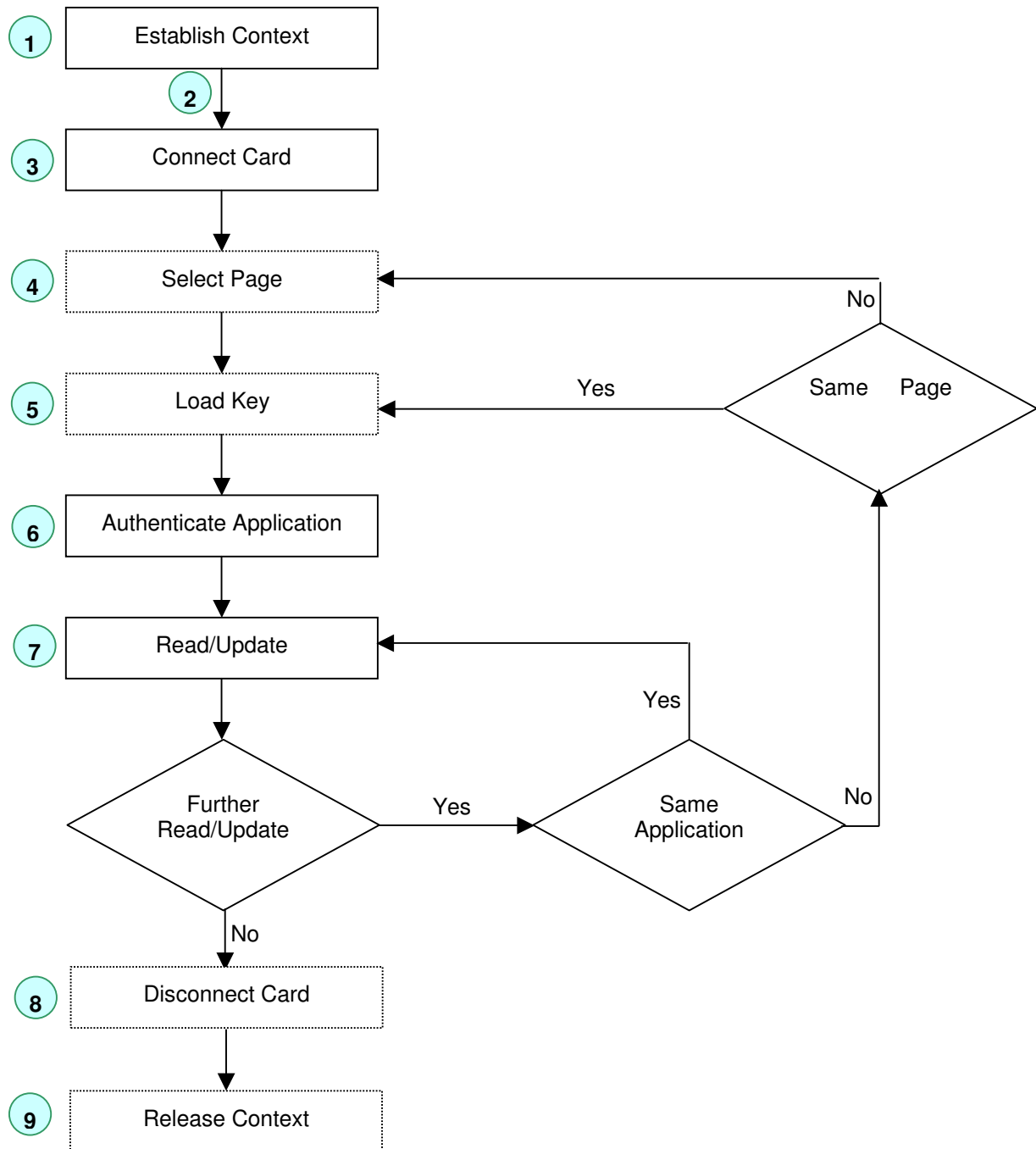
For iCLASS, if Lc \neq 8, error code is returned, as only 8 bytes can be updated once.

Table 8-29: Update Binary error codes

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

Note: Trying to update without authenticating the corresponding application will return 'Card Execution Error'.

8.3 Communication at Standard Mode



8.4 Some Remarks on the communication structure at standard mode

1. To establish Context please call the following PC/SC function

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

2. Here if necessary the status of the reader can be checked for card insertion, removal or availability of the reader by calling the following PC/SC function:

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

To see check all the connected readers the following PC/SC function may be called:

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

3. A card must be connect to communicate with that card by calling the following PC/SC function:

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

Please note that, currently for memory card e.g. iCLASS only T=0, protocol is supported

Step 4, 5, 6, 7 can be accomplished by calling the following synchronous API function with correct command specific APDU:

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

4. If the application resides in the page 0 of 8x2KS iCLASS card or in the single page iCLASS 16KS or iCLASS 2KS card, select page command is not necessary unless it is intentional to retrieve some data described in the command APDU.
5. If the key is always the same for the application, then it is not necessary to load the keys always. One can load KIAMC (key number 20) once and can use this always for the authentication.

8. It is not mandatory to disconnect the card after completion of all the transactions but preferred. A connected card can be disconnected by calling the following PC/SC function:

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

9 Secured communication with iCLASS Card

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

- The authenticity between the host application and the reader.
- The confidentiality of transmitted data in the USB cable.
- The integrity of transmitted data
- The authenticity between the reader and the card
- The confidentiality and integrity in the RF transmission.
- The confidentiality of stored data in the cards.

OMNIKEY CardMan 5x21-CL reader provides an end-to-end security by playing an enhanced role in the above-mentioned episodes.

Moreover there is a technique to allow the user only read capability to the card or both read and write.

Note: only the reader firmware version 500 and more support secured mode communication.

9.1 Preview of securities in different stages

9.1.1 Authenticity between the host and reader

A 16-byte transport key (Kcur or Kcuw) and a proprietary algorithm for one step mutual authentication will be given to the integrator provided by an NDA. With this key and specific algorithm the host can authenticate itself and can start the session.

This will protect unauthorized use of the reader

9.1.2 The confidentiality of transmitted data in the USB cable

The data transmitted in the USB cable is **triple DES** encrypted with the Session Key (Ks). This session key is generated in the authenticity of the host phase and it is always different for every session.

This will protect the communication against the sniffing attack in USB

9.1.3 The integrity of transmitted data

Every transmission contains a digital signature (8 byte MAC) in the end of the data based on the preceding data bytes.

9.1.4 The authenticity between the reader and the card

This part will be according to the Inside Contact-less proprietary algorithm. For this, every application will have a key K_{IAMC} or K_{MDC} .

9.1.5 Integrity in the RF transmission

For iCLASS card the RF transmission data required two bytes CRC.

9.1.6 Confidentiality in the RF transmission

The reader supports encryption of the data before writing to the card and decryption of them after reading.

9.1.7 Read/Write session distinction

There are two keys K_{CUR} and K_{CUW} stored in the reader. If a session is started using K_{CUR} , it will be a read only session. In this session one is not allowed to write to the card. If a session is started using K_{CUW} it will be a read/write session.

9.1.8 Protection against some known attacks

- Reply attack:

For every data gram send to the reader will have different Data Header. Therefore it will be easily detected in the reader as the frame is repeated or not.

- Plain text attack:

For some critical command a delay has been introduced in order to face the plain text attack.

If there is any error in the data header or signature the session is immediately terminated. One can do communication only again after starting a session.

(Please report other probable attack that you can imagine)

9.2 APDU structure for Secured communication

Table 9-1: APDU application to reader

CLA	INS	P1	P2	Lc	Omnikey proprietary send data-gram	Le
0x84	XX	XX	XX	XX	XXXXXXXXXXXXXXXXXXXXXXXXXXXX	XX

Table 9-2: APDU, reader to application

Omnikey proprietary response data-gram	SW2	SW1
XXXXXX	XX	XX

Omnikey proprietary data structure (Omnikey proprietary send data-gram and Omnikey Proprietary response data gram) are completely proprietary. And they are described as follows:

Table 9-3:Omnikey proprietary send data gram

Data Header Send (DH)	No of INS related data LC_{INS}	INS related data (INSData)	Padding (PB)	Signature
XXXXXX	XX	XXXX	80000000	XXXXXX
4 bytes	1 byte	n bytes	P bytes	8 bytes

3-DES{K_s, ()}

P = required number of bytes in such way that (4+1+n+P) is multiple of 8.

Table 9-4: Omnikey Proprietary response data gram

Data Header Response(DH)	No Byte Card Response (LcR)	Card Response	Padding (PB)	Signature
XXXXXX	XX	XXXXX	800000	XXXXX
4 bytes	1 byte	n bytes	P bytes	8 bytes

3-DES{K_s, ()}

P = required number of bytes in such way that (4+1+n+P) is multiple of 8.

Note: If there is an error occurred in 'Start Session' command as no valid session key has been generated all 0s is returned ending with SW1SW2.

9.2.1 Data Header (DH):

Table 9-5: Data Header

B0	B1	B2	B3
Host data header (HDH)		Reader data header (RDH)	

Whatever the HDH is sent by the host to the reader, reader sends back always the 1's complement of the HDH. Host must check if it receives the data header what it should receive.

Whatever the RDH is sent to the host by the reader, host must send back to the reader 1's complement of RDH in the next data-gram sent to the reader. Reader checks if he received the data header what he should receive.

9.2.2 Calculation of Signature:

8- byte signature is calculated by C-MACing of the preceded (DH, LcINSDData, INSDData, Padding) data bytes. For this MACing scheme first 8-byte of Ks or KCUR/KCUW is used as the DES encryption key in CBC mode.

Unless specified to the contrary, padding prior to performing a DES operation across a block of data is achieved according to the following manner:

- Append an '80' to the right of the data block.
- If the resultant data block length is a multiple of 8, no further padding is required.
- Append binary zeroes to the right of the data block until the data block length is a multiple of 8.

9.2.3 Example-Proprietary data structure for one session:

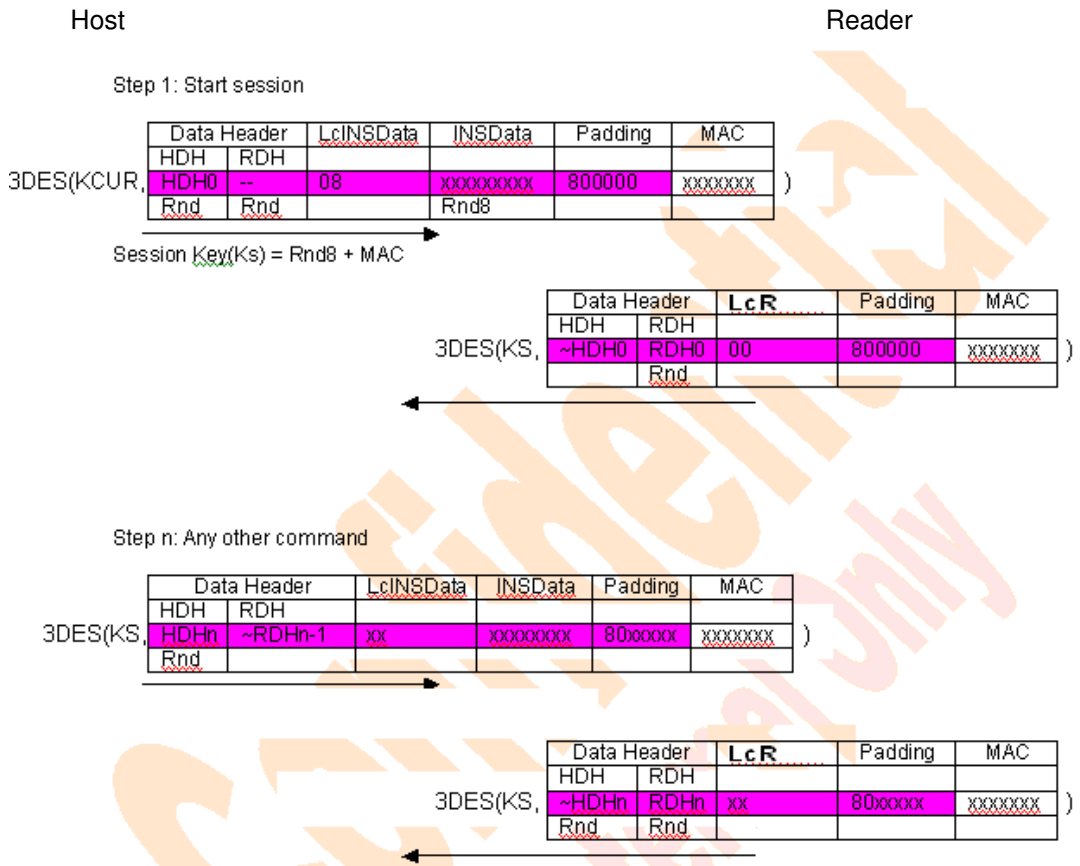


Figure: Proprietary data structure scenario

Note: This is a read only session as KCUR has been used in the starting session command. If KCUR is used in the starting session command then both read and write is allowed. The HID application is always read only.

9.3 Instructions (INS) for Secured communication

In the beginning the following Instructions code will be supported

Table 9-6: Instructions

Instruction	Description
0x72	Manage Session
0x82	Load Key
0x88	Authenticate
0xB0	Read
0xD6	Update
0xA6	Select Page
0xC4	GetKeySlotInfo

In the following sections the command structure is described, the LcINS and INSDData are the part from Omnikey proprietary structure.

Please note that, P1 has a different structure for INS 0xB0 and 0xD6 in 'Secured mode' other than in 'Standard Mode'.

9.3.1 Manage Session command:

The manage session command manages the session either start a session or end a session.

Table 9-7: Manage Session command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSDATA	Le
Manage Session	0x84	0x72	xx	xx	xx	xxxx	--

Data Out	
--	SW1 SW2

P1:

If P1 = 0x00, start session

If P1 = 0x01, end session

Other values are RFU

P2:

If P1 = 0x00

P2 = 0x00, start read only session

P2 = 0x01, start read/write session

If P1 = 0x01, P2 = 0x00

Other values are RFU

LC_{INS}: 0x08, if Start session.

: 0x00, if End Session.

INSDATA: 8 byte random number if start session command, empty if end session command.

Table 9-8: SW1SW2 for Session command

	SW1	SW2	Meaning
Error	According to table 7-3, common error code		

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

9.3.2 Select page command:

Select page command will select specific page in the iCLASS card

Table 9-9: Select page command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSData	Le
Select page	0x84	0xA6	xx	xx	xx	xxxx	xx

Data Out	
Card Response	SW1 SW2

LC_{INS}: According to INSData 0x00 to 0xFF

INSData: Absent or Page number (according to P1)

Le: If any data field is requested according to P2 and (Le = 0x00 or Le= 0x08) then 8-byte data is returned according to P2.

Table 9-10: P1 of Select page Command

b7	b6	b5	b4	b3	b2	b1	b0	Meaning	Command data field
0	0	0	0	0	0	0	0	Select the only page of iCLASS 2KS or single page of 16KS	Absent
0	0	0	0	0	0	0	1	Select page of multi-page iCLASS 16KS (8x2KS) or 32 KS	Page number **
Other values are RFU									

**** The data field (1 byte page number) is as follows:**

Table 9-11: Page number

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	Book number	0	Page number 0-7		

For iCLASS 16KS (8x2KS) or first book (8x2KS or 16KS) of iCLASS 32KS, the book number is always 0. For second book Book number is 1.

An example chart is as follows:

Table 9-12: Example of page number

Item	Page number
Select page 3 of an iCLASS 8x2KS card	0x03
Select page 3 of book 0 of an iCLASS 32KS (book 0: 8x2KS) card	0x03
Select page 3 of book 1 of an iCLASS 32KS (book 1: 8x2KS) card	0x13
Select book 1 (16KS) of an iCLASS 32KS	0x10
Select book 0 (16KS) of an iCLASS 32KS	0x00

Table 9-13: P2 of Select page Command

b7	b6	b5	b4	b3	b2	b1	b0	Meaning
0	0	0	0	0	0	0	0	Return nothing
0	0	0	0	0	1	0	0	Return 8-byte card serial number
0	0	0	0	1	0	0	0	Return 8-byte configuration block data
0	0	0	0	1	1	0	0	Return 8-byte application issuer data
Other values are RFU								

Table 9-14: SW1SW2 for Select page command

	SW1	SW2	Meaning
Error	62	83	Requested page number does not exist
	6C	XX	Wrong length Le. XX returns the number of data available

9.3.3 Load Key command:

Load Key command loads the Key in the reader memory. If 'Authenticate' command wants to use the card key from the reader key container, it must be in the reader key container or in the volatile memory. The volatile key may be used only for the succeeding authentication.

Table 9-15: Load Key command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSDat a	Le
Load Key	0x84	0x82	Key Structure	Key number	Key Length	Key	-

Data Out
SW1 SW2

Table 9-16: Definition of P1 of Load Key command APDU

b7	b6	b5	b4	b3	b2	b1	b0	Description
x								0 card key, 1 Reader key
	X							0:Plain transmission, 1:Secured transmission
		x						If 0, the keys are loaded in the IFDs volatile memory If 1, the keys are loaded in the IFDs nonvolatile memory.
			x					RFU (set 0, else return error)
				0000				Not Valid (set all 0, else return error)

Whole communication is secured, Not only the key, so b6 must be set to 0, no reader key is allowed to load so b7 is always set to 0.

P2 (Key Number): The key number is according to table 8-1.

The following table introduces some examples of SW1SW2 and their meaning.

Table 9-17: Load Keys command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
Error	'63'	'81'	Loading/Updating is not allowed
		'82'	Card key not supported
		'83'	Reader key not supported
		'84'	Plain transmission not supported
		'85'	Secured transmission not supported
		'86'	Volatile memory is not available
		'87'	Non volatile memory is not available
		'88'	Key number not valid
		'89'	Key length is not correct

9.3.4 Authenticate command:

Authenticate command authenticates the application of the selected page.

Table 9-18: Authentication command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSData	Le
Authenticate	0x84	0x88	Key Type	Key Nr	0x00	--	-

Data Out
SW1 SW2

P1:

Table 9-19: Key Type

Value (b7 - b0)	Description
0x00	Inside Contactless Kd
0x01	Inside Contactless Kc
0x60	Mifare Key A
0x61	Mifare Key B
0xFF	Key Type unknown or not necessary
Other values	RFU

Key Nr.:

The card key number, which will be used for this authentication according to table 8-1

Table 9-20: Authentication command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
	'69'	'83'	Authentication cannot be done/Key is not correct
		'84'	Reference key not useable
		'85'	Key type not known
		'87'	HID application is not supported by this reader
		'88'	Key number not valid

Note: Right now only supported for iCLASS card. Valid P1 = 0x00 or 0x01.

9.3.5 Read command:

Read command reads the data from the given block address.

Table 9-21: Read command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSDATA	Le
Read	0x84	0xB0	Option+Block Nr. MSB	Block Nr. LSB	0x00	--	xx

Data Out	
Card response	SW1 SW2

Table 9-22: Option of P1

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

According to the option the data of the block is decrypted with the K_{ENC} and send to the host.

Le:

If Le = 0x08 or 0x00, 8 bytes are returned starting from block address offset.

If Le = 0x20, 32 bytes are returned starting from block address offset (if the card supports reading 32 bytes command).

Other values of Le return error '6C08'

Table 9-23: Read Binary error codes

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

Note: Reading the blocks without authenticating the corresponding application will return all "FF". Reading the blocks, which are not allowed to read, will return all "FF"

9.3.6 Update command:

Update command updates the given block number with the given data.

Table 9-24: Update command APDU

Command	Class	INS	P1	P2	LC _{INS}	INSDATA	Le
Update	0x84	0xD6	Option+Block Nr. MSB	Block Nr. LSB	xx	xxxx	--

Data Out	
Data	SW1 SW2

Table 9-25: P1 option

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

According to the option the data is encrypted with the K_{ENC} and then is written to the given block number.

LC_{INS}:

If $Lc \neq 8$, error code is returned, as only 8 bytes can be updated once.

Table 9-26: Update Binary error codes

	SW1	SW2	Meaning
Error	'65'	'81'	Memory failure (unsuccessful writing)
	'69'	'81'	Command incompatible
		'86'	Command not allowed
	'6A'	'81' '82'	Function not supported File not found / Addressed block or byte does not exist

Note: Trying to update without authenticating the corresponding application will return 'Card Execution Error'.

9.3.7 GetKeySlotInfo command:

OMNIKEY CardMan 5x21-CL reader has some predefined Key Slots in the key container. The user can only load key by using the key number. IFD decides where to store (in which slot) the key. But user can get the status of the key slots by using this GetKeySlotInfo command.

Table 9-27: GetKeySlotInfo Command

Command	Class	INS	P1	P2	LC _{INS}	INSDATA	Le
GetKeySlotInfo	0x84	0xC4	0x00	KeySlot number according to table 8-2	0x00	Absent	xx

Le: If Le = 0x00 or Le = 0x02, 2-byte information is returned else error ('6C02') is returned.

Data Out	
Reader Response	SW1 SW2

Reader Response will be 2 Bytes according to the following table:

Table 9-28: Key Information Byte

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
RFU						Access Type		Key Number according to table 8-1							

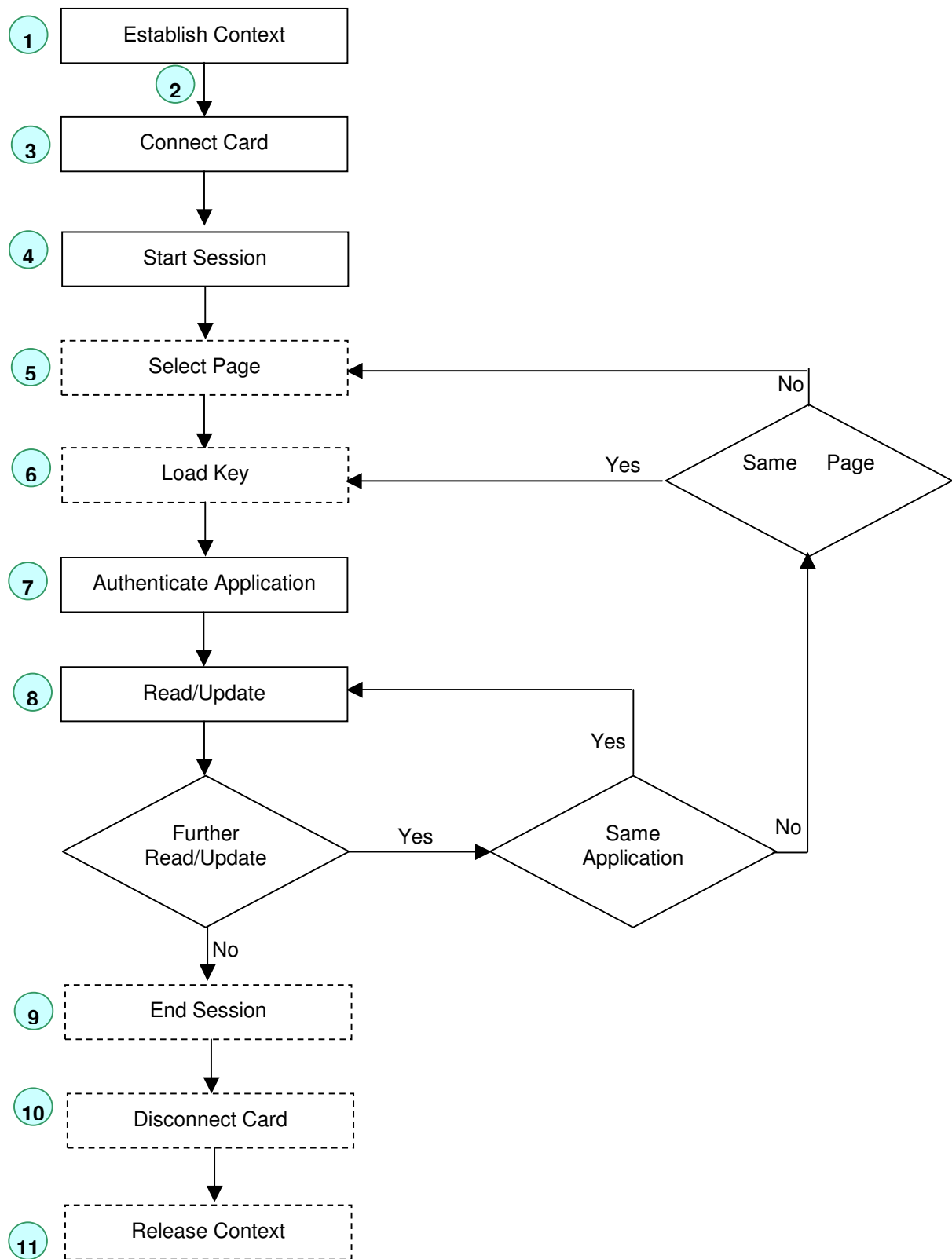
Table 9-29: Access option

b9	b8	Access Option
0	0	Can be loaded in any type of transmission e.g. plain or secured
0	1	Allowed to load only in Omnikey proprietary secured mode
1	0	Loading is never allowed
1	1	RFU

Table 9-30: GetKeySlotInfo error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is available
	'63'	'01'	Key slot does not contain valid key/empty
	62	83	Requested Key slot does not exist
Error	6C	XX	Wrong length Le < requested data, XX returns the number of data available

9.4 Communication at Secured Mode



9.5 Example APDUs for a Session at Secured Mode

$K_{CUR} = A0A1A2A3A4A5A6A7A8A9AAABACADAEAF$, Read only session

Host

Reader

1. Start Session

CLA	INS	P1	P2	Lc	Omnikey proprietary Send data gram					
84	72	00	00	18	1422	9D2B	08	4A895F20C2D30B5E	800000	9E5052819C5A8D3C
					HDH (Rnd)	RDH (Rnd)	LcINS	Rnd8 (INSDData)	Padding	Signature
					DH					MAC
					FD274CE840FA9AD139E4FC2923653A88743CB5986DB4F7A0					
					Proprietary Data					

Signature = $DESEn\{(A0A1A2A3A4A5A6A7), (14229D2B084A895F20C2D30B5E800000)\}$
 = 8A8D430D608714FE9E5052819C5A8D3C
 9E5052819C5A8D3C (last 8-byte block)

Proprietary Data = $3-DESEn\{(A0A1A2A3A4A5A6A7A8A9AAABACADAEAF), (14229D2B084A895F20C2D30B5E8000009E5052819C5A8D3C)\}$
 = FD274CE840FA9AD139E4FC2923653A88743CB5986DB4F7A0

SessionKey (K_S) = Rnd8+MAC
= 4A895F20C2D30B5E9E5052819C5A8D3C

Omnikey proprietary Response data gram					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: On request, Omnikey will be glad to provide an open source library to accomplish all security protocols introduced in the secured communication mode.

2. Authenticate HID Application

CLA	INS	P1	P2	Lc	Omnikey proprietary Send data gram				
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 data gram					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: On request, Omnikey will be glad to provide an open source library to accomplish all security protocols introduced in the secured communication mode.

3. Read Block 6

CLA	INS	P1	P2	Lc	Omnikey proprietary Send data gram						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 data gram						SW1S W2
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: On request, Omnikey will be glad to provide an open source library to accomplish all security protocols introduced in the secured communication mode.

10 Inter industry commands for synchronous cards

Global inter-industry commands for the synchronous cards as described by PC/SC part 3 v2.1 is already implemented in Linux and will be implemented in windows very soon. This will give the users possibility to work with only the PC/SC interface, not to have some proprietary API like the synchronous API currently we provide.

11 Performance

11.1 Supported baud rates

The current version of the reader is capable of supporting the following transmission speed:

ISO14443A:

- 106 kbits/sec
- 212 kbits/sec
- 424 kbits/sce
- 848 kbits/sec (limited)

ISO14443B:

- 106 kbits/sec
- 212 kbits/sec
- 424 kbits/sce

ISO15693:

- Low: 6.62 kbits/sec
- High:26. kbits/sec

11.2 DESFire Working speed from an example application

Test Procedure:

The test has been performed under the following entity:

- A DESFire v 0.5 sample card.
- ISO 7816 wrapped APDU
- A VC++ application run under Windows 2000 in a P III 500 MHz PC
- Omnikey CardMan 5121 Reader
- Driver Version 1.26
- Firmware Version 101 **
- Air interface at 424 kbits/sec

Test Result:

Reading:

1024 Bytes took 130 mSec, i.e. @61.54 kbps

2048 Bytes took 250 mSec, i.e. @ 65.54kbps

Writing:

1034 Bytes took 290 mSec, i.e. @ 27.85 Kbps

2048 Bytes took 531 mSec, i.e. @ 30.86 kbps

** Higher FW versions have better performance.

Note: Currently the readers allow maximum 47 bytes writing using the ISO7816 wrapped APDU for DESFire card at a time, on the other hand Reading is possible to maximum 255 bytes.

Example: ISO 7816 wrapped APDU for Write command:

Send:

CLA	INS	P1	P2	Lc	FileNo	Offset	Length	Data upto 47 bytes	Le
90	3D	00	00	xx	xx	xxxxxx	xxxxxx	xxxxxxxx...xxxx	00

Receive:

SW1	SW2	Meaning
91	00	Success
91	xx	Error: According to DESFire data sheet

Example: ISO 7816 wrapped APDU for Read command:

Send:

CLA	INS	P1	P2	Lc	FileNo	Offset	Length	Le
90	BD	00	00	07	XX	XXXXXX	LLLLLL (Maximum FF)	00

Receive:

LL-byte Data	SW1	SW2
--------------	-----	-----

SW1	SW2	Meaning
91	00	Success
91	xx	Error: According to DESFire data sheet

Note: Omnikey CardMan 5x21 supports DESFire native APDU as well as ISO 7816 wrapped APDU. The default configuration is supporting DESFire card through ISO 7816 wrapped APDU. To configure the reader for working with DESFire native APDU structure, please contact Omnikey.

Appendix A Application Programming

A1 Sample project

A complete sample in C++ can be found in Samples\contactlessdemovc of the Synchronous API installation. The same sample in Visual Basic can be found in Samples\contactlessdemo vb.

A1.1 Overview

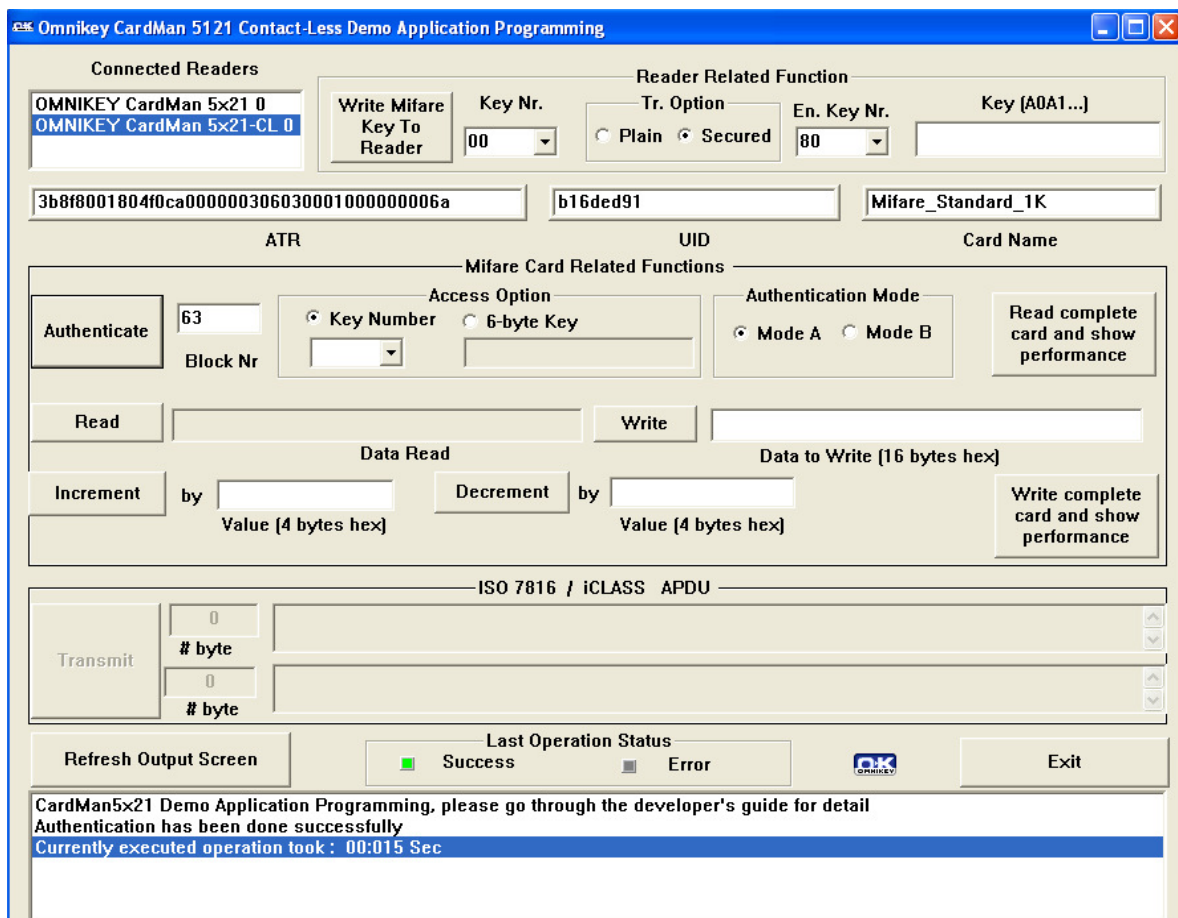


Figure 8: Screenshot of contactlessdemovc

In the list box in the top-left corner of the window you can select the reader to use. When a card is inserted the **ATR**, **UID** and **Card Name** will be shown in the text fields below. The functions in the group box **Reader Related functions** can be used with or without card in the field. In the center of the window are some **Mifare Card Related functions**, that can only be used when a card is in the field. Below you find the group box **ISO 7816 – APDU**, there you can send and receive APDUs for asynchronous cards.

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

The button **Exit** closes the application

A1.2 Reader Related functions

The functions in this group box can be used with or without a card in the field.

To store a mifare key do the following:

- Chose a key number, to which location the key will be stored
- Chose if the key will be transferred plain or secured and if secured place Transmission key nr. 0x80 or 0x81
- Write the key in hex string format in the text field **Mifare Key**. If plain 6 bytes, if secured it will be 8 bytes.
- Press the Button **Write Mifare Key to Reader**.

A1.3 Mifare Card Related functions

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

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

- Put the block number to authenticate to in the field **Block Nr**.
- In the box **Access Option** chose whether a key number or a plain key will be supplied.
- In the box **Authentication Mode** chose **Mode A** oder **Mode B**.
- Press the button **Authenticate**.

After authentication it is possible to read and write blocks of the card and use the increment and decrement functions.

A1.4 ISO 7816 - APDU

If an asynchronous card is presented to the reader, it is possible to send APDUs directly to the card and receive the answer with these functions.

A1.5 iCLASS Standard Mode

If an iCLASS card is present to the reader, it is possible to send APDUs directly to the card and receive the answer with these functions.

A2 Code snippets

Here are some short code-snippets that explains how to use some of the functions.

A2.1 Connect card

The following sample code will establish the context to resource manager, select a reader and connect to a card. The commands to access the card can be added after the comment "Work with the card".

```
#include <stdio.h>

#include <winscard.h>

//defines and includes for Sync API
#define S_WNT
#include <okos.h>
#include <ok.h>
#include <scardcl.h>

int main(void) {
    SCARDCONTEXT hContext;
    DWORD dwErrorFlags;

    LPTSTR pmszReaders = NULL;
    LPTSTR pchCardReaderName = NULL;
    CHAR szReaderName[128];
    DWORD cch = SCARD_AUTOALLOCATE;

    DWORD dwActiveProtocol;
    SCARDHANDLE hSCARDHandle;

    //
    // Establish Context to resource manager
    //
    dwErrorFlags = SCardEstablishContext( SCARD_SCOPE_USER,
                                         NULL,
                                         NULL,
                                         &hContext);

    if (dwErrorFlags != SCARD_S_SUCCESS) {
        fprintf(stderr, "ERROR: SCardEstablishContext failed\n");
        exit(-1);
    }

    //
    // List Readers in the system
    //
    dwErrorFlags = SCardListReaders( hContext,
                                     NULL, //list all readers in the system
                                     (LPTSTR)&pmszReaders,
                                     &cch); //auto allocate

    if (dwErrorFlags != SCARD_S_SUCCESS) {
        fprintf(stderr, "ERROR: SCardListReaders failed\n");
        exit(-1);
    }

    //
    // Print the Reader List and select the first CardMan 5121
    //
    pchCardReaderName=pmszReaders;
    while (*pchCardReaderName!=0) {
        printf("<%s>\n", pchCardReaderName);
        if (!strncmp("OMNIKEY CardMan 5121", pchCardReaderName, 20)) {
            strcpy(szReaderName, pchCardReaderName);
            break;
        }
        pchCardReaderName += strlen(pchCardReaderName)+1;
    }
    if (*pchCardReaderName == 0) {
        fprintf(stderr, "ERROR: No CardMan 5121 found\n");
        exit(-1);
    }
    else {
        printf("Selected: %s\n", pchCardReaderName);
    }
}
```

```

}

//
// Free the memory allocated by SCardListReaders
//
dwErrorFlags = SCardFreeMemory( hContext,
                                pmszReaders );
if (dwErrorFlags != SCARD_S_SUCCESS) {
    fprintf(stderr, "ERROR: SCardFreeMemory failed\n");
    exit(-1);
}

//
// Connect to the card
//
// wait for card
printf("Waiting for card\n");
do {
    dwErrorFlags = SCardConnect( hContext,
                                szReaderName,
                                SCARD_SHARE_SHARED,
                                SCARD_PROTOCOL_T0,
                                &hSCARDHandle,
                                &dwActiveProtocol);
} while (dwErrorFlags == SCARD_E_NO_SMARTCARD ||
        dwErrorFlags == SCARD_W_REMOVED_CARD ||
        dwErrorFlags == SCARD_W_UNPOWERED_CARD);

//
// Work with the card
//

//
// Disconnect from the card
//
dwErrorFlags = SCardDisconnect( hSCARDHandle,
                                SCARD_RESET_CARD );

//
// Release Context
//
dwErrorFlags = SCardReleaseContext(hContext);

return 0;
}

```

A2.2 Mifare 1K/4K Authenticate

The following code-snippet will authenticate to the card

```

...
//MIFARE 1K/4K authenticate
OKERR okErr;
BYTE keya[6] = {0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5};

//Authenticate to card using keya and direct key input
okErr = SCardCLMifareStdAuthent( hSCARDHandle,
                                1, //block
                                MIFARE_AUTHENT1A,
                                MIFARE_KEY_INPUT,
                                0, //keyNr
                                keya,
                                6 );

if (okErr != NO_ERROR) {
    fprintf(stderr, "ERROR: SCardCLMifareStdAuthent failed\n");
    exit(-1);
}
...

```

A2.3 Mifare 1K/4K Read/Write

The following code-snippet will write and read one sector of the card.

```
...
//MIFARE 1K/4K read and write
OKERR okErr;
BYTE patternMF1KSect[16] = {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,
                           0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA};

BYTE buffer[16];
ULONG ulRead;

//ScardCLMifareStdWrite
okErr = SCardCLMifareStdWrite( hSCARDHandle,
                               1, //block
                               patternMF1KSect, //data buffer
                               16); //no of bytes to write

if (okErr != NO_ERROR) {
    fprintf(stderr, "ERROR: SCardCLMifareStdWrite failed\n");
    exit(-1);
}

//SCardCLMifareStdRead
memset(buffer, 0x00, sizeof(buffer));
okErr = SCardCLMifareStdRead( hSCARDHandle,
                              1, //block
                              buffer, //buffer to read in
                              16, //no of bytes to read
                              &ulRead); //no of read bytes

if (okErr != NO_ERROR) {
    fprintf(stderr, "ERROR: SCardCLMifareStdRead failed\n");
    exit(-1);
}

if (ulRead != 16) {
    fprintf(stderr, "ERROR: SCardCLMifareStdRead failed (not enough bytes)\n");
    exit(-1);
}

if (memcmp(buffer, patternMF1KSect, 16) != 0) {
    fprintf(stderr, "ERROR: SCardCLMifareStdRead failed (compare failed)\n");
    exit(-1);
}
...

```

A2.4 Mifare 1K/4K Increment/Decrement

The following code-snippet initializes one sector with a pattern which can be used for increment and decrement operations and increments this sector by one. For a reference of increment and decrement operations and sectors refer to the Mifare Datasheet.

```
...
//Mifare 1K/4K Increment and Decrement
//Mifare block with increment/decrement pattern. value = 0
BYTE patternIncBlock0[16] = {0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
                             0x00, 0x00, 0x00, 0x00, 0x01,
0xFE, 0x01, 0xFE};
//Mifare block with increment/decrement pattern. value = 1
BYTE patternIncBlock1[16] = {0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0xFE,
                             0x00, 0x00, 0x00, 0x01, 0x01, 0xFE, 0x01, 0xFE};
BYTE incl[4] = {0x00, 0x00, 0x00, 0x01};

BYTE buffer[16];
ULONG ulRead;
//Write Increment/Decrement pattern
SCardCLMifareStdWrite( hSCARDHandle,
                      1,
                      patternIncBlock0,
                      16);

okErr = SCardCLMifareStdIncrementVal( hSCARDHandle,
                                     1, //block
                                     incl,
                                     4 );

//compare
SCardCLMifareStdRead( hSCARDHandle,
                    1, //block
                    buffer, //buffer to read data in
                    16,
                    &ulRead);
if (okErr != NO_ERROR) {
    fprintf(stderr, "ERROR: SCardCLMifareIncrementVal failed\n");
    exit(-1);
}
if (memcmp(patternIncBlock1, buffer, 16) != 0) {
    fprintf(stderr, "ERROR: SCardCLMifareIncrementVal failed (compare)\n");
    exit(-1);
}
...

```

A2.5 iCLASS Select Page

The following code-snippet selects page 01 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
ucDataSend [1] = 0xA6 //INS
ucDataSend [2] = 0x01 //P1
ucDataSend [3] = 0x04 //P2, 0x04 returns card serial number
ucDataSend [4] = 0x01 //Lc
ucDataSend [5] = 0x02 //Page number
ucDataSend [6] = 0x08 //Le

```

```
OKErr =
SCardCLICCTransmit (hCard,ucDataSend,ulNoOfDataSend,ucReceivedData,&ulNoOfDataReceived);

if(OKErr != NO_ERROR)
{
printf("Error in SCardCLICCTransmit, with error code %8X", OKErr);
exit(-1);
}
```

Appendix B Accessing of iCLASS free zones

In the following diagrams the free zones in iCLASS memory organization is shown:

Block	Size: 8 bytes
0x00	Card serial number
0x01	Configuration block
0x02	e-Purse
0x03	Kd (Key for Application 1)
0x04	Kc (Key for Application 2)
0x05	Application issuer area
0x06	HID Application
....	
0x12	
0x13	Free zones in iCLASS 2KS, iCLASS 16KS or page 0 of iCLASS 8x2KS
....	
0x1F(2KS)	
0xFF(16KS)	

(a) Free zone of iCLASS 2KS, iCLASS 16KS or page 0 of iCLASS 8x2KS card

Block	Size: 8 bytes
0x00	Card serial number
0x01	Configuration block
0x02	e-Purse
0x03	Kd (Key for Application 1)
0x04	Kc (Key for Application 2)
0x05	Application issuer area
0x06	Application 1 (Free zones in iCLASS 8x2KS other than page 0)
....	
0xXX	
0xXX+1	Application 2 (Free zones in iCLASS 8x2KS other than page 0)
....	
0x1F	

(b) Free zones of iCLASS 8x2KS other than page 0

In the iCLASS default card 'XX' is not set, complete area is allotted to application 1. If it is necessary to have two separate applications then the configuration block must be rewritten. For this purpose the following steps can be followed:

1. Select the desired page
2. Authenticate using the Kd of that page
3. Read 8 bytes from block 0x01
4. Replace first byte with 'XX' (If other configurations stays as it is)
5. Write these 8 bytes to block 0x01
6. Remove the card.

There are some steps as follows to access the free zone of iCLASS card:

(a) Free zone of iCLASS 2KS, iCLASS 16KS or page 0 of iCLASS 8x2KS card:

1. Insert card
2. Connect card*
3. Authenticate with K_{MC0} , (P1 = 0x01, P2 = 0x23)**.
4. Read/write any block (block number 0x13 to 0x1F for 2KS and 0xFF for 16KS).***
5. Disconnect card
6. Remove card

(b) Free zone of iCLASS 8x2KS card other than page 0:

1. Insert card
2. Connect card*
3. Select page N (N = 1 to 7)
4. Authenticate with K_{MDN} / K_{MCN} (P1 = 0x00 for K_{MDN} , or 0x01 for K_{MCN} , P2 = K_{MDN} / K_{MCN} number from the table 8.1) depends on the application number **.
5. Read/write any block (block number 0x13 to 0x1F for 2KS and 0xFF for 16KS).***
6. Disconnect card
7. Remove card

* For secured mode add here Start Session command

** 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

*** For secured mode add here End Session command