



CardMan 5121 RFID Developers guide

Document Version: 1.01

Abstract: This document describes the easy usability of CardMan 5121 contactless interface for developers.

Last modified: 04.04.2005

Disclaimer: Copyright © 2004 by OMNIKEY AG

All Rights Reserved.

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

Table Of Contents

1 Document Information	4
1.1 Change History	4
1.2 Further Reading	4
1.3 Terms and Abbreviations	5
2 Scope	7
3 Installation	8
4 Diagnostic Tool	9
4.1 Usage	9
4.2 Change card detection order	10
5 ATR generation	11
5.1 Memory Cards (T≠CL)	11
5.1.1 Lower nibble of Card Type identifier	11
5.1.2 Higher nibble of Card Type identifier	12
5.1.3 Examples	13
5.2 Asynchronous Cards (T=CL)	13
5.2.1 Examples	14
6 Access asynchronous cards	15
7 Access synchronous cards (Memory cards)	16
7.1 Overview	17
7.2 Functions	17
7.2.1 SCardCLGetUID	17
7.2.2 SCardCLMifareLightWrite	18
7.2.3 SCardCLMifareStdAuthent	18
7.2.4 SCardCLMifareStdDecrementVal	19
7.2.5 SCardCLMifareStdIncrementVal	20
7.2.6 SCardCLMifareStdRead	21
7.2.7 SCardCLMifareStdRestoreVal	21
7.2.8 SCardCLMifareStdWrite	22
7.2.9 SCardCLWriteMifareKeyToReader	23
7.2.10 SCardCLWriteTransmissionKeyToReader	24
7.2.11 SCardCLICCTransmit	25
8 Standard Communication with iCLASS Cards	26
8.1 APDU structure for Standard communication	26
8.2 Supported INS in the standard communication	27
8.2.1 Select Page	27
8.2.2 Load Key	29

8.2.3	GetKeyInfo	31
8.2.4	Authenticate command:	33
8.2.5	Read command:	35
8.2.6	Update command:	36
8.3	Communication flow diagram for iCLASS card	37
8.3.1	Some Remarks on the communication structure:	37
9	Inter industry commands for synchronous cards	39
10	Mifare Security Support	40
10.1	Reader to Card secured transmission	40
10.2	Host to Reader secured transmission	40
11	Performance	42
11.1	Supported baud rates	42
11.2	DESFire Working speed from an example application	42
12	Application Programming	44
12.1	Sample project	44
12.1.1	Overview	44
12.1.2	Reader Related functions	45
12.1.3	Mifare Card Related functions	45
12.1.4	ISO 7816 - APDU	45
12.2	Code snippets	46
12.2.1	Connect card	46
12.2.2	Mifare 1K/4K Authenticate	47
12.2.3	Mifare 1K/4K Read/Write	48
12.2.4	Mifare 1K/4K Increment/Decrement	49
12.2.5	iCLASS Select Page	49
13	Tested cards	51

1 Document Information

1.1 Change History

<i>Version</i>	<i>Authors</i>	<i>Date</i>	<i>Description</i>
1.00	AI, CT	14-12-2004	Initial Version
1.01	AI	05-04-2005	iCLASS Standard support added

1.2 Further Reading

[MIFARE] MIFARE Datasheets

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

[PCSC] PC/SC Workgroup Specifications 2.0

<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

[ISO7816-4] Information technology Identification cards Integrated circuit(s) cards with contacts Part 4: Interindustry commands for interchange

1.3 Terms and Abbreviations

Unique Card Serial Number (CSNR):

Eight-byte unique serial number of each iCLASS card.

Customer Read Key (K_{CUR}):

Sixteen-byte Customer Read Key will give the application only read capability of the iCLASS card. This default key will be used only once for the low-level security command "Select Page" in every session.

Customer Write Key (K_{CUW}):

Sixteen-byte Customer Write Key will give the application read and write capability of the iCLASS card. This default key will be used only once for the low-level security command "Select Page" in every session.

Customer Transmission Key Current (K_{CUC}):

Sixteen-byte Customer Transmission key Current will be used for the encryption of current transmission from the host to the reader.

In every session $K_{CUC0} = K_{CUR}$, if this is read only session.

In every session $K_{CUC0} = K_{CUW}$, if this is read and write session.

Customer Transmission Key Next (K_{CUN}):

Sixteen-byte Customer Transmission key Next will be provided by the application in each data gram send to the reader, which will be used for the encryption of the reply of the data gram, then will be considered as the Customer Transmission Key Current

$$K_{CUC\ n+1} = K_{CUN\ n}; \text{ where } n \text{ is the number of transaction in a session.}$$

HID Master key Current (K_{MDC}):

Eight-byte key will be used for authentication of HID Application.

HID Master key Old (K_{MDO}):

If the HID Master key Current is changed then the previous K_{MDC} will be stored as K_{MDO} .

Inside Application Master key Current (K_{IAMC}):

Eight-byte key will be used for authentication of card application area other than HID Application. The reader supports two K_{IAMC} : one volatile and one non-volatile.

Card Data Encryption Key (K_{ENC}):

Sixteen-byte key will be used for 3-DES encryption of data to be stored in the card.

Data Header (DH):

4-byte Data header will be in the starting of each data gram.

CRC16:

In the end of each data gram there will be 2-byte CRC for integrity check.

INSData:

Instruction specific data byte.

2 Scope

The CardMan 5121 is a combined contact and contactless reader. It can only read one card at a time, because it logically supports only one slot. So the access to contact and contactless cards can only be done sequentially at the moment.

This document is the developers guide for CardMan 5121. It describes the how to work with contactless cards. The functionality in conjunction with contact cards is the same as in CardMan 3121, so its developers guide can be used to develop applications that use contact cards.

The organization of this document is as follows:

Chapter 3 describes the installation of the device.

Chapter 4 explains the functionality and usage of the diagnostic tool, which is installed with the driver.

Chapter 5 explains, how ATRs are generated for contactless cards, that normally do not have an ATR.

Chapter 6 describes how asynchronous cards are used.

Chapter 7 describes how synchronous cards (memory cards) are used. Additionally the synchronous API is described.

Chapter 8 explains, iCLASS support through the sync API function.

Chapter 9 explains, what inter-industry commands for contactless memory cards are supported by this reader.

Chapter 10 explains the security concepts of the CardMan 5121.

Chapter 11 handles some performance issues.

Chapter 12 gives the application programmer some help how to write applications using contactless cards.

Chapter 13 lists tested cards, that are known to work with CardMan 5121.

3 Installation

For driver installation refer to the “**CardMan 5121 Installation manual**”.

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

4 Diagnostic Tool

The Diagnostic tool is a small control panel applet. It can be used to list all installed OMNIKEY readers, show file and driver version and change the settings for card detection order.

4.1 Usage

Diagnostic Tool is 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.

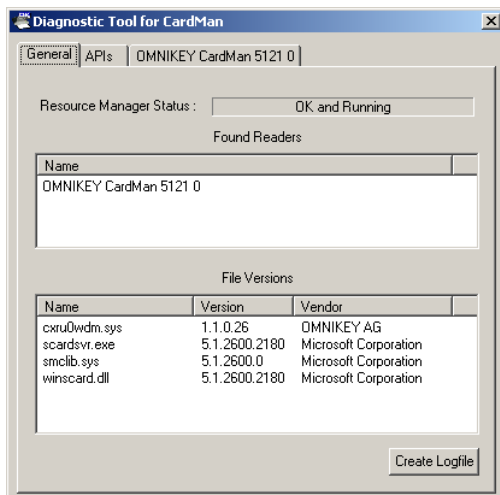


Figure 1: Diagnostic Tool, General Tab

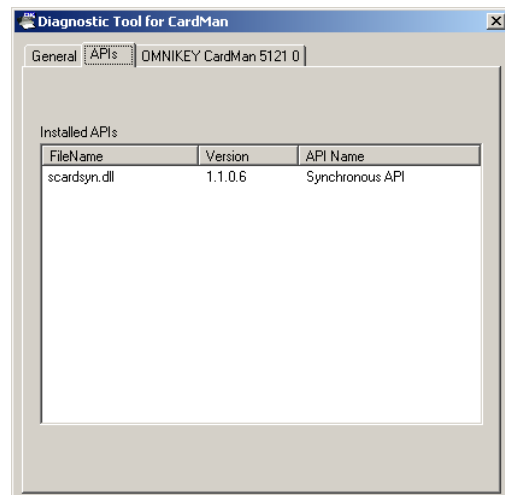


Figure 2: Diagnostic Tool, API Tab

Every installed and active reader has its own tab:

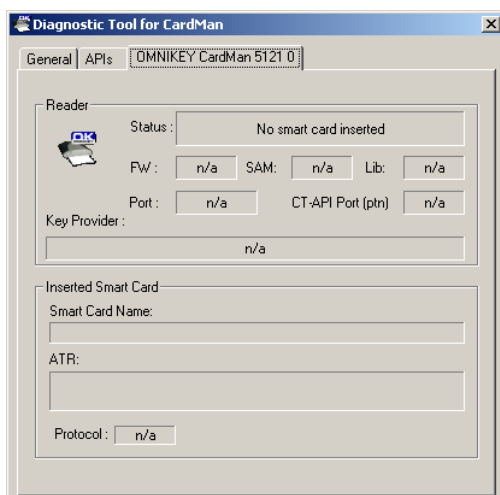


Figure 3: Diagnostic Tool, Reader Tab

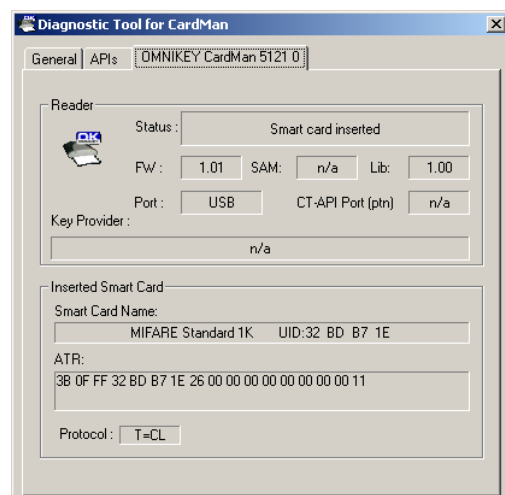


Figure 4: Diagnostic Tool, Reader Tab (Card inserted)

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 Change card detection order

For contactless cards there are many different standards, which require special activation routines (e.g. ISO14443A, ISO14443B, ISO15694, ...). As there is no physical card presence switch for the contactless cards, the reader has to check always if there is a card in the field. This is done by trying to do all activation sequences for different card standards. As there can be only on RF field, this activation sequences has to be done sequentially, which takes some time.

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.

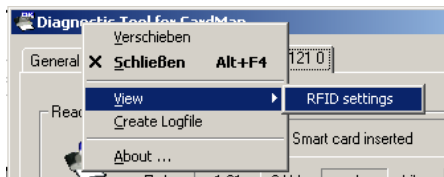


Figure 5: Diagnostic Tool, activate RFID settings

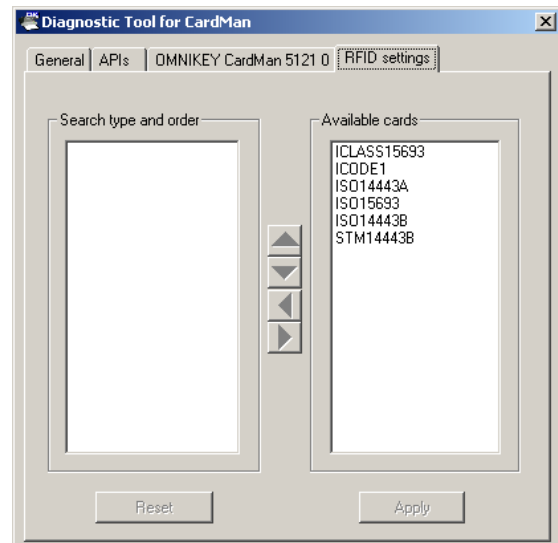






Figure 6: Diagnostic Tool, RFID settings Tab

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, chose 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 settings 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.

5 ATR generation

The ATR of a contact-less card is artificially mapped to support these cards through PC/SC. The synchronous cards produce an ATR of T = 0 type, on the other hand the asynchronous cards introduces both T = 0 and T = 1 type.

5.1 Memory Cards (T≠CL)

ATR is generated according to PC/SC Part 3 (Revision 2.0, February 2004) Section 3.1.3.2.3 ATR and last byte (Byte 16) is Omnikey proprietary card identity.

Initial character	Header	Content														
3B	0F	FF	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX
Byte 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Fixed		UID / PUPI / SNR														Card Type

The ATR length is fixed, always 17 bytes.

Byte 0; 0x3B Initial Character

Byte 1; 0x0F, (0 means no 7816 specific information is given, F (15) is the number of bytes in content, byte 2 – byte 16)

Byte 2; 0xFF, always same.

Byte 3 to Byte 15 (13 bytes) are filled with UID or PUPI or SNR, if it has not 13 bytes, the rests are filled with 0x00. In case of ISO 14443A -1,2,3(e.g. Mifare) UID is taken UID +1 byte BCC.

Byte 16; is the indication of the card type. The lower nibble tells to which specification the inserted card complies. The higher nibble tells which card is inserted (according to the names of the card manufacturer)

5.1.1 Lower nibble of Card Type identifier

Standard	Card Type	Comments
ISO14443A_123	X1	Part 1,2,3 of ISO 14443A
ISO15693_12	X3	Part 1,2 and InsideContactless Technology (iCLASS)
ISO15693_123	X4	Part 1,2,3 of ISO15693
ISO14443B_12	X5	Part 1,2 of ISO14443B and STmicroelectronics technology
ISO14443B_123	X6	Part 1,2,3, of ISO 14443B
ISO15693_12ICODE1	X8	Part 1,2 and Philips ICODE1

5.1.2 Higher nibble of Card Type identifier

ISO14443A_123:

Card	Card Type
UNKNOWN_CHIP	0X
MIFARE_ST1K	1X
MIFARE_ST4K	2X
MIFARE_ULIT	3X
SLE55R_XXXX	7X

ISO15693_12:

Card	Card Type
iCLASS Unknown	8X
iCLASS 2K	CX
iCLASS 2KS	9X
iCLASS 16K	DX
iCLASS 16KS	AX
iCLASS 8x2K	EX
iCLASS 8x2KS	BX

ISO15693_123:

Card	Card Type
SRF55V10P	1X
SRF55V02P	2X
SRF55V10S	3X
SRF55V02S	4X
TAG_IT	9X
LRI512	AX
ICODESLI	BX
SRF55XXX	CX
TEMPESENS	DX

ISO14443B_12:

Card	Card Type
SR176	EX
SRIX4K	FX

ISO14443B_123:

Card	Card Type
AT88RF020	1X

AT88SC6416CRF	2X
---------------	----

ISO15693_12ICODE1:

Card	Card Type
ICODE1	1X

5.1.3 Examples

ISO 14443 A, Mifare 1K # content: FF + UID + BCC + Rest 00 + CardType:

Initial character	Header	Content														
3B	0F	FF	32	4D	58	32	15	00	00	00	00	00	00	00	00	11

ISO 14443 B, AT88RF020 # content: FF + PUPI + Rest 00 + CardType:

Initial character	Header	Content														
3B	0F	FF	00	00	D6	B6	00	00	00	00	00	00	00	00	00	16

5.2 Asynchronous Cards (T=CL)

ATR is generated in such a way to support both T = 0 and T = 1 protocol according to ISO7816-3. The entry in the ATR is as follows:

Initial character	T0	TD1	TD2	Historical Bytes	LRC
3B	8n	80	01	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	XX
Byte 0	1	2	3	HistByte 0.....HistByte n-1	

T0:

Higher nibble 8 means no TA1, TB1, TC1 only TD1 following
 Lower nibble n is the number of historical bytes (HistByte 0 to HistByte n-1)

TD1:

Higher nibble 8 means no TA2, TB2, TC2 only TD2 following
 Lower nibble 0 means T = 0

TD2:

Higher nibble 0 means no TA3, TB3, TC3, TD3 following
 Lower nibble 1 means T = 1

Historical Bytes:

ISO14443A:
 The historical bytes from ATS response. ISO14443-4 page 7

ISO14443B:

The higher layer response from the ATTRIB response. ISO14443-3 page 36

LRC:

Exclusive-oring of all the bytes T0 to HisBytes n-1.

5.2.1 Examples

Mifare DESfire card:

Initial character	T0	TD1	TD2	Historical Bytes	LRC
3B	81	80	01	80	80

JCOP BIO31 card:

Initial character	T0	TD1	TD2	Historical Bytes	LRC
3B	86	80	01	4A 43 4F 50 33 31	13

6 Access asynchronous cards

Asynchronous cards can be accessed using PC/SC ISO 7816 command APDUs. No special API is needed.

7 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**
- **SCardCLWriteTransmissionKeyToReader**
- **SCardCLWriteMifareKeyToReader**
- **SCardCLICCTransmit**

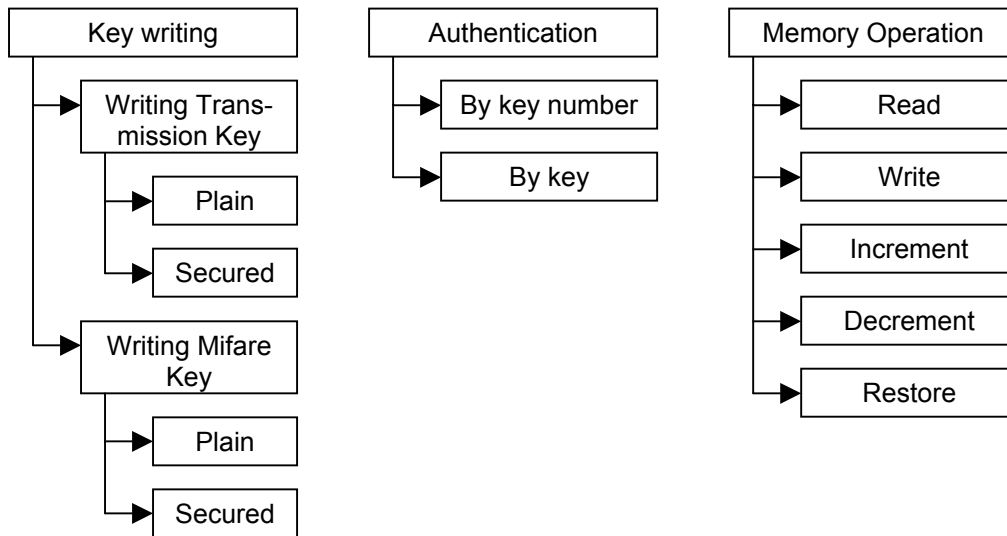
The functions **SCardCLWriteTransmissionKeyToReader** and **SCardCLWriteMifareKeyToReader** are for writing the keys to the reader. These functions do not care either the card is present, connected or not. The other functions are completely card related.

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

Currently it supports only the iCLASS card.

7.1 Overview

The functionality for Mifare cards, implemented in CM5121 can be classified as follows:



7.2 Functions

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

7.2.1 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
ulHandleCard		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.
ulUIDBufLen		The length of the ucUID buffer
pulnByteUID		Number of bytes in the UID

7.2.2 SCardCLMifareLightWrite

```
OKERR ENTRY SCardCLMifareLightWrite
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PCHAR 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

7.2.3 SCardCLMifareStdAuthent

```
OKERR ENTRY SCardCLMifareStdAuthent
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN UCHAR ucMifareAuthMode,
    IN UCHAR ucMifareAccessType,
    IN UCHAR ucMifareKeyNr,
    IN PCHAR 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.

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

7.2.4 SCardCLMifareStdDecrementVal

```
OKERR ENTRY SCardCLMifareStdDecrementVal
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PCHAR 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 Decrement. 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.

7.2.5 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

7.2.6 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

7.2.7 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

7.2.8 SCardCLMifareStdWrite

```
OKERR ENTRY SCardCLMifareStdWrite
(
    IN SCARDHANDLE ulHandleCard,
    IN ULONG ulMifareBlockNr,
    IN PCHAR 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

7.2.9 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 ulTransmissionKeyNr
);

```

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 connected considered.

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
ulTransmissionKeyNr		If the transmission is secured, then define the Transmission key Number which will be for encryption and decryption of MifareKey, this must be 0 or 1

7.2.10 SCardCLWriteTransmissionKeyToReader

```
OKERR ENTRY SCardCLWriteTransmissionKeyToReader
(
    IN SCARDHANDLE ulHandleCard,
    IN SCARDCONTEXT hContext,
    IN PCHAR pcCardReader,
    IN ULONG ulTransmissionKeyNr,
    IN ULONG ulTransmissionKeyLen,
    IN PUCCHAR pucTransmissionKey,
    IN BOOLEAN fSecuredTransmission,
    IN ULONG ulTransTransmissionKeyNr
);
```

The function SCardCLWriteTransmissionKeyToReader writes the Transmission keys in the reader. These keys are rewriteable. These keys will be used for secured key transmission from reader to host, if you set fSecuredTransmission of SCardCLWriteMifareKeyToReader TRUE. One can also make secured these TransmissionKey transmission by setting fSecuredTransmission to TRUE here. In this case One has to supply the encrypted TransmissionKey and the number of the TransmissionKey, which has been used for this encryption. 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 connected considered.

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
ulTransmissionKeyNr		The key number which will be written, the number must be from 0 or 1
ulTransmissionKeyLen		The length of the Transmission key, must be 16
pucTransmissionKey		16-byte key, if secured, it must be encrypted with other key
fSecuredTransmission		If the TransmissionKey transmission is intended to be secured, place TRUE, for plain place FALSE
ulTransTransmissionKeyNr		The TransmissionKeyNr, which has been used to encrypt the TransmissionKey going to write. The valid value is 0 or 1.

7.2.11 SCardCLICCTransmit

There is only one function responsible for communication between the storage card and application in both modes standard and secured.

OKERR ENTRY SCardCLICCTransmit (

```

    IN SCARDHANDLE    ulHandleCard,
    IN PCHAR          pucSendData,
    IN ULONG          ulSendDataBufLen,
    IN OUT PCHAR      pucReceivedData,
    IN OUT PULONG     pulReceivedDataBufLen);

```

pucSendData: Pointer to the buffer of data send to reader, this data buffer will according to table 7-1.

pucReceivedData: Pointer to the buffer of response data according to table 7-2.

Table 7-1: Data gram, application to reader

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

Table 7-2: Data gram, reader to application

Response data gram ^{***}	SW2	SW1
xxxxxxx	xx	xx

***** In the secured communication mode, this Send data gram and Response data gram will be in Omnikey propriety format. For Standard mode it is explained in the following chapter.**

Table 7-3: Common status codes

	SW1	SW2	Meaning
No Error	'90'	'00'	Success
Error	'67'	'00'	Wrong length
	'68'	'00'	Class byte is not correct
	'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 has been introduced in every command sections.

8 Standard Communication with iCLASS Cards

The Synchronous API function SCardCLICCTransmit is called to communicate with the iCLASS cards. 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
XXXXXX	xx	xx

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

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

The following instruction (INS) commands are supported by CardMan 5121 in the standard communication mode.

Table 8-3: Instructions

Instruction	Description
0xA6	Select Page
0x82	Load Key
0xC4	GetKeyInfo
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 $N_c = 0$, present for encoding $N_c > 0$

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

Le: If requested data according to P2 and $Le = 0$ then Data Out will be the requested data field, if $Le > 0$, N_e data bytes will be returned. If Le field is absent no data byte will be returned.

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)	Page number 0 to 7
Other values are RFU									

Table 8-7: 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 Card serial number
0	0	0	0	1	0	0	0	Return configuration block data
0	0	0	0	1	1	0	0	Return Application issuer data
Other values are RFU								

Table 8-8: SW1SW2 for Select page command

	SW1	SW2	Meaning
Warning	62	82	Le > requested data, available data is returned
Error	62	83	Requested page number does not exist
	6C	XX	Wrong length Le < requested data, 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 8-9: 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 8-10: Data Out

Data Out	
-	SW1 SW2

Table 8-11: 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: As right now, the intension is to allow loading of K_{IAMC} only, so b7 must be set to 0 and b6 must be set to 0 as no encryption of key only

P2 (Key Number):

Please use the key Number according to the following table.

Table 8-12: Definition of P2 of Load Key command APDU

Key Number.	Comments	Key Length	Key Type	Memory location
0x00 to 0x1F are for all 6-byte Mifare key				
0x00 to 0x1F	Mifare Key 0 to Mifare Key 31	6 bytes	Card Key	Non volatile memory
0x20 to 0x7F are for all 8-byte key				
0x20	K_{IAMC}	8 bytes	Card Key	Non volatile memory
0x21	K_{MDC}	8 bytes	Card Key	Non volatile memory
0x22	K_{MDO} (This key is internally used)	8 bytes	Card Key	Non volatile memory
0x80 to 0xAF are for all 16-byte key.				
0x80	Host Key 0 (K_{CUR})	16 bytes	Reader Key	Non volatile memory
0x81	Host Key 1 (K_{CUW})	16 bytes	Reader Key	Non volatile memory
0x82	K_{ENC}	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	Any application key	8 bytes	Card Key	Volatile memory

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.

Please note for iCLASS card other than HID Application only one non volatile key K_{IAMC} (0x20) and one volatile key (0xF0) are supported.

Right now only valid combinations are

P1 0x20, P2 0x20 or P1 0x00, P2 0xF0 and always Lc = 0x08 and 8 byte data

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

Table 8-13: Load Keys command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
Error	'63'	'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 GetKeyInfo

Presents the information of the requested key place of the reader key container

Table 8-14: GetKeyInfo Command

Command	Class	INS	P1	P2	Lc	Data in	Le
GetKeyInfo	0x80	0xC4	xx	0x00	0x01	xx	xx

P1:

0x00: The information of the key resides in non-volatile memory

0x01: The information of the key resides in the volatile memory

Data in:

The key place holder (The IFD may provide 0 to 255 Key place holder) according to the following table.

Table 8-15: iCLASS Key container

Key Name	Key number according to table 7-12	Key place
Mifare Key (non-volatile)	Mifare Key 0 –31, there will be no key information available for Key place 0. This is reserved for 6 byte all 32 Mifare keys	0x00 - 0x1F

Key Name	These keys are non volatile	Key place (KP)
K _{CUR}	Key number 0x80	0x20
K _{CUW}	Key number 0x81	0x21
K _{ENC}	Key number 0x82	0x22
K _{IAMC}	Key number 0x20	0x23
K _{MDO}	Key number 0x22	0x24
K _{MDC}	Key number 0x21	0x25

Key		Key place (KP)
Volatile Application key (KIAMC)	Key number 0xF0	0x26

Le:

If Le = 0x00 then also 2 information bytes are returned if Le =0x01 error is returned if Le > 0x02 only two bytes are returned. If Le absent error is returned.

Table 8-16: GetKeyInfo Command Response

Data Out	
xxxx	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					Key status	Access Type	Key Number according to table 8-12								

b10 : If it is 0, the location is occupied and the following bits are valid. If it is set to 1 key place is empty b0 to b7 bits are not valid.

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 Omnikey proprietary secured mode
1	0	Loading is never allowed
1	1	RFU

Table 8-19: GetKeyInfo command error codes

	SW1	SW2	Meaning
Warning	'62'	'82'	Le > requested data, available data is returned
	'63'	'00'	No information is available
Error	62	83	Requested Key place does not exist
	6C	XX	Wrong length Le < requested data, XX returns the number of data available
	'69'	'82'	Security status not satisfied

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

Key Nr.:

The card key number, which will be used for this authentication according to table 8-12

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.

Right now only valid combinations are

P1 0x00, or P1 0x01, P2 0x20 or P2 0xF0, Lc , data in, Le must be empty.

Table 8-23: Authentication command error codes

	SW1	SW2	Meaning
Warning	'63'	'00'	No information is given
	'69'	'82'	Security status not satisfied
		'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:

For iCLASS card the Le has the following meaning

If Le is 0, 8 bytes are returned starting from block address offset.

If $0 < Le \leq 32$, Le number of bytes are returned starting from block address offset.

If $Le > 32$, 32 bytes are returned starting from block address offset.

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' '82' '86'	Command incompatible Security status not satisfied Command not allowed
		'81' '82'	Function not supported File not found / Addressed block or byte does not exist
		'XX'	Wrong length (wrong number Ne; 'XX' encodes the exact number)

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

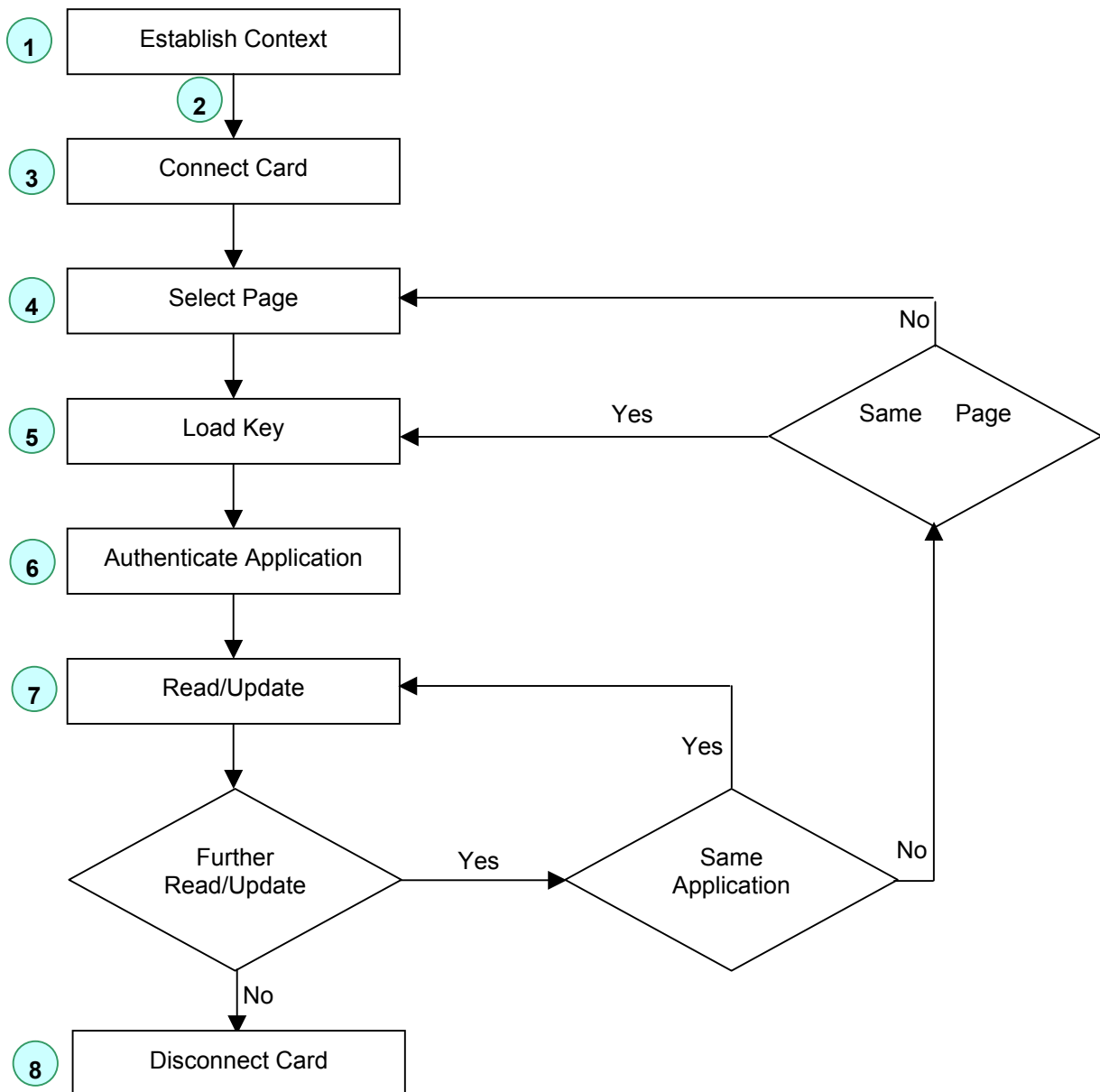
Lc:

For iCLASS, if Lc ≠ 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.
		'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.	

8.3 Communication flow diagram for iCLASS card



8.3.1 Some Remarks on the communication structure:

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 CardMan 5121 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);
```

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 Inter industry commands for synchronous cards

Omnikey prefers some global inter-industry commands for the synchronous cards, which will give the user possibility to work with only the PC/SC interface not to have some proprietary API like the synchronous API currently we provide. In this issue, we have submitted a proposal on unique Inter-industry commands to the PC/SC workgroup members. After the official acceptance and publication, these commands will be explained here.

10 Mifare Security Support

This security is only valid for communication with the Mifare standard card.

There are two episodes in the security feature.

Reader and Card: The security in the air interface between the card and the reader.

Reader and Host: The security in the transmission cable between the reader and the host.

10.1 Reader to Card secured transmission

The different cards are using different authentication schemes for secured transmission of data between the reader and the card. Most of the cases the mutual authentication is based on a key known by the card and the reader. Some times the data is transmitted between the chips and the reader is encrypted with the card manufacturer's defined technology. This security completely chips dependent. As example DESFire can use DES Encryption.

10.2 Host to Reader secured transmission

For the best use of the security feature provided by the card, the application must write the key in the reader before using it. Now while the key is written to the reader, could be sniffed from the USB transmission line and can produce a security hole and results the card manufacturer authentication scheme useless. Therefore CardMan 5121 is providing a cryptographic technology for transmission of the secured data from the host to the reader.

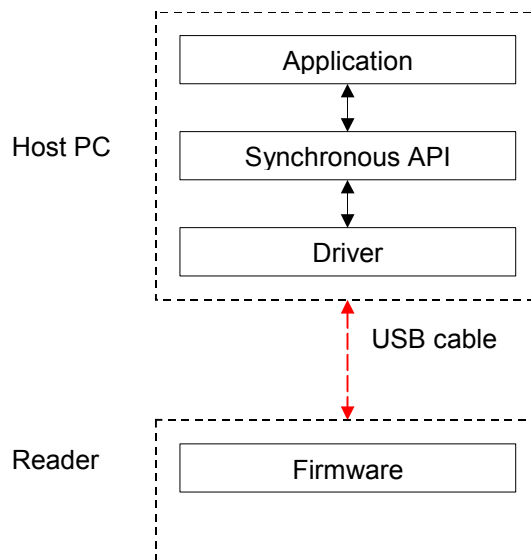


Figure 7: Scope of Security feature - Omnikey Cryptography is proposed for red dashed path

This approach introduces two types of keys:

Card Key: These are the card specific key (e.g. Mifare key, iClass Debit or credit key) used to authenticate the cards. The current version supports only the Mifare Key. If not mentioned Mifare Key term used any where will be treated as one Card Key. There are 6-byte long 32 number of non-volatile Mifare Keys can be written to the reader. For the authentication of standard Mifare block, one can use the specific key number, Please see the SCardCLMifareStdAuthent functions

Transmission Key: This key may be used to secure the transmission of **Card Key** . There are two 16 bytes long non volatile Transmission keys.

Now the functionality and writing of the keys are as follows:

There could be two types of CardMan 5121: one with a default Transmission key and other without any default Transmission key. For the reader without any default key please write one of the two transmission keys (Transmission key 0 or 1) in plain mode using

SCardCLWriteTransmissionKeyToReader. Now one can encrypt a 16-byte long key with the previously written transmission key using standard 3-DES and can write in secured mode. If there is a default key in the reader it will be always as Transmission Key 0, one can use this key as a previously written key. In that case Omnikey will provide the default key in a secured way on request. Writing of the transmission key is always possible. Now there are Transmission keys in the reader, The reader is ready for the secured transmission of Card Key.

As mentioned before current versions support only writing Mifare key. To write Mifare Key in a secured way encrypt the Mifare Key with one of the Transmission Key using standard 3-DES. As Mifare Key is 6 byte long, add two bytes padding of zero. Now take the secured option in

SCardCLWriteMifareKeyToReader TRUE and provide the Transmission Key number used for the encryption.

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

ISO14443B:

- 106 kbits/sec

ISO15693:

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

Note: The higher baud rates e.g. 847 kbits/sec for ISO14443A, and higher baud rate ISO14443B type chips will be supported very soon.

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

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 APDU for Write command:

Send:

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

Receive:

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

Example 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

12 Application Programming

12.1 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\contactlessdemovb.

12.1.1 Overview

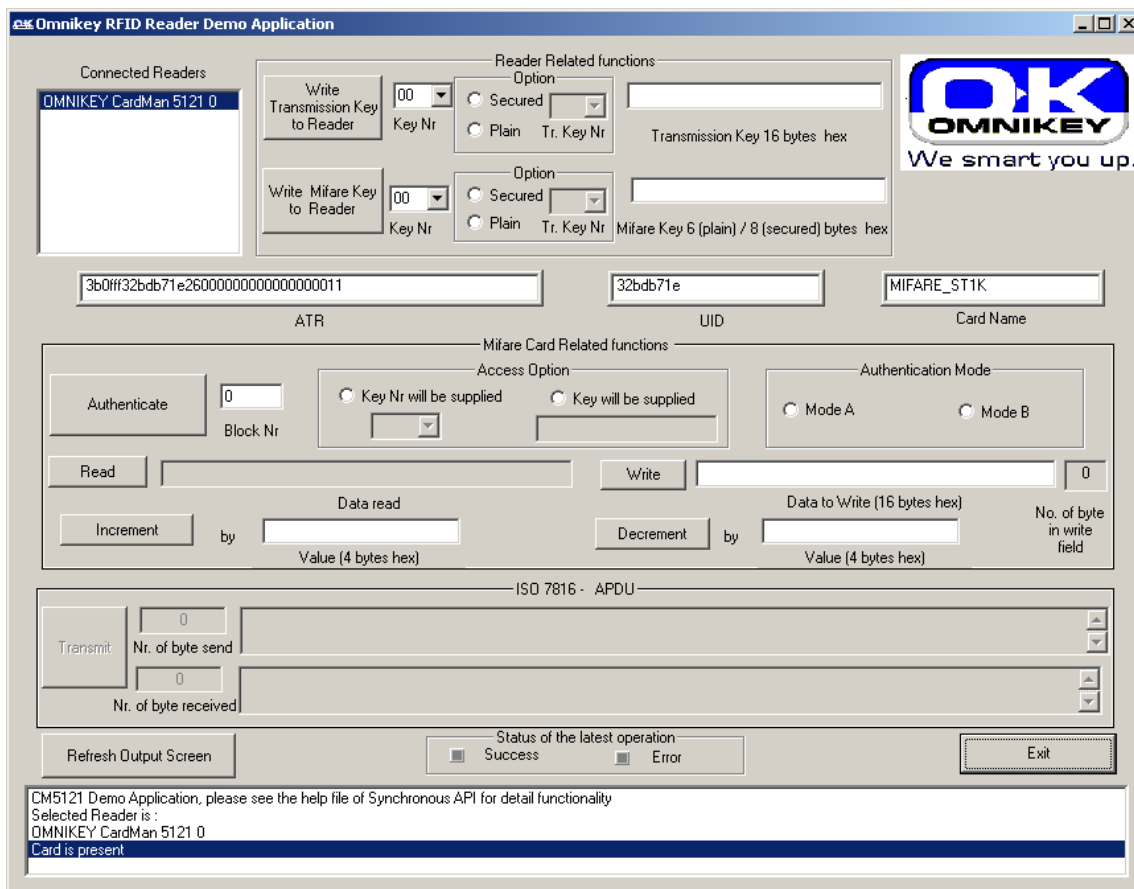


Figure 8: Screenshot of contactlessdemovc

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

12.1.2 Reader Related functions

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

To store a transmission 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 with which Transmission key it will be encrypted.
- Write the key in hex string format in the text field **Transmission Key**.
- Press the Button **Write Transmission Key to Reader**.

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 with which Transmission key it will be encrypted.
- 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**.

12.1.3 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.

12.1.4 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.

12.2 Code snippets

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

12.2.1 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;
}

```

12.2.2 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);
}
...

```

12.2.3 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);
}
...

```


12.2.4 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);
}
...
```

12.2.5 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);
}
```

13 Tested cards

CardMan 5121 is capable to support every card which follows any of the standards ISO14443A, ISO14443B or ISO15693.

The following cards have been tested to work with CardMan 5121:

ISO14443A:

- Mifare 1K
- Mifare 4K
- Mifare Ultra Light
- Desfire v0.4
- Desfire v0.5
- Mifare Prox SPK D1
- Mifare Prox
- SLE 44R35S
- SLE 66R35
- SLE 55R16
- TCOS CLX
- TCOS NET
- IC-ONE

ISO14443B:

- AT88RF020
- AT88SC0204CRF
- AT88SC0808CRF
- AT88SC1616CRF
- AT88SC3216CRF
- AT88SC6416CRF
- SR176
- SRIX4K
- SLE66CL
- Setec

ISO15693:

- SRF 55V10P
- SRF 55V10S
- Tag-IT
- LRI 512
- KSW TempSens
- ICODE SLI
- iCLASS 2K
- iCLASS 2K AG
- iCLASS 16K
- iCLASS 16K CE
- PicoPass 2KS
- PicoTag 2KS
- ICODE-1