PC Pinpad Smartcard reader PC/SC Application note.

Version : Version 3
Last update : 05/26/2007

Contents

Preface

This document describes the use of secure PIN commands compliant with the PC/SC V2 standard and of the messages displayed on the PC Pinpad screen. These commands are designed for use with the PC Pinpad smart card reader.

Audience

This document is intended for developers of PC/SC–based applications. It requires familiarity with the Microsoft PC/SC application programming interface (API)

For more information

For additional information about the PC/SC API, refer to:
Software Development Kit for Microsoft WIN32 [SDK].
Interoperability Specification for ICCs and Personal Computer Systems:
Part 10 IFDs with Secure Pin Entry Capabilities Rev 2.01.6. December 2005
[PC/SC] http://www.pcscworkgroup.com/specifications/specdownload.php .

Contact for comments

If you don't find the information you need in this manual, or if you find errors, contact the Gemalto hotline by phone, fax, or email. In your email, please include the document reference number, your job function, and the name of your company. (You will find the document reference number at the bottom of the legal notice on the inside front cover.)

Hotline: +33 (0)4 42 36 50 50

Hot Fax: +33 (0)4 42 36 64 00

Email: nis.support@gemalto.com

From Our Web Site

http://support.gemalto.com/

<u>**Secure Pin Information:**</u>

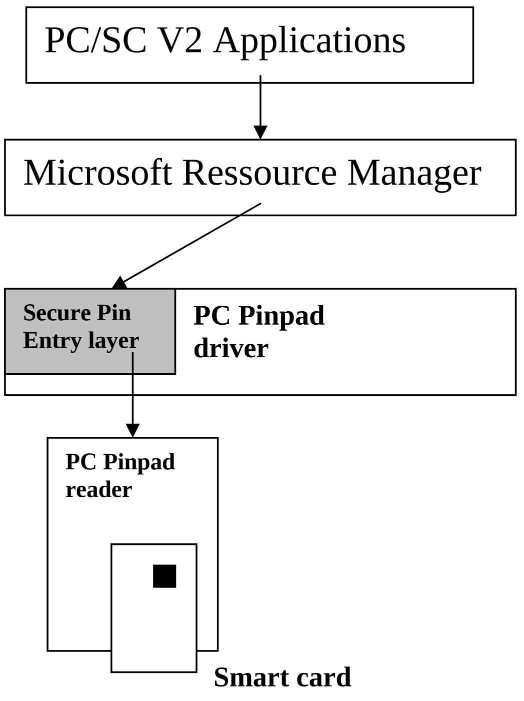
The PC Pinpad prevents Trojan horse-type attacks by taking advantage of the smart card’s built in security. The user enters the PIN code on the PC Pinpad keyboard and the value entered is then validated by the smart card, enabling direct communication between the keyboard and the smartcard. Because the PIN is not sent to the computer, breaches of security cannot occur.

When the entire code has been keyed in and validated by a carriage return, the code is presented to the card and the reader returns the card status to the computer.

The character displayed on the PC Pinpad screen does not at any point go beyond the reader. Consequently, the Personal Identification Number (PIN code is completely secure). It is impossible in any way to deduce it.

PC/SC V2 overview:

Part 10 of PC/SC V2 [PC/SC] defines standardized calling for supporting Secure PIN entry capability. The PC Pinpad reader, like any other class 2/3 readers must follow this specification.
Under Microsoft Windows, the PC/SC V2 layer is not implemented by the Operating system. This layer is implemented inside the PC Pinpad driver as shown in the figure here below:



To be compliant with the PC/SC V2 specification, a reader must support, at least, the GET_FEATURE_REQUEST command.

This GET_FEATURE_REQUEST command returns all the optional commands supported by the reader and how to call these commands. The command answer is a TLV structure which list all the supported command and the control code to call them (see Get Feature Request chapter for more details)

For the PC Pinpad reader, 2 optional commands are supported:

- FEATURE_VERIFY_PIN_DIRECT
- FEATURE_MODIFY_PIN_DIRECT

These commands are the only necessary commands for entering the PIN code in a secure way. For more details about these commands, refer to chapter Verify PIN and Modify PIN commands.

Management of secure PIN commands:

The way for calling this command is always the same:
ScardControl() function is called with the following main parameters (other parameters are detailed in Secure PIN commands chapter):

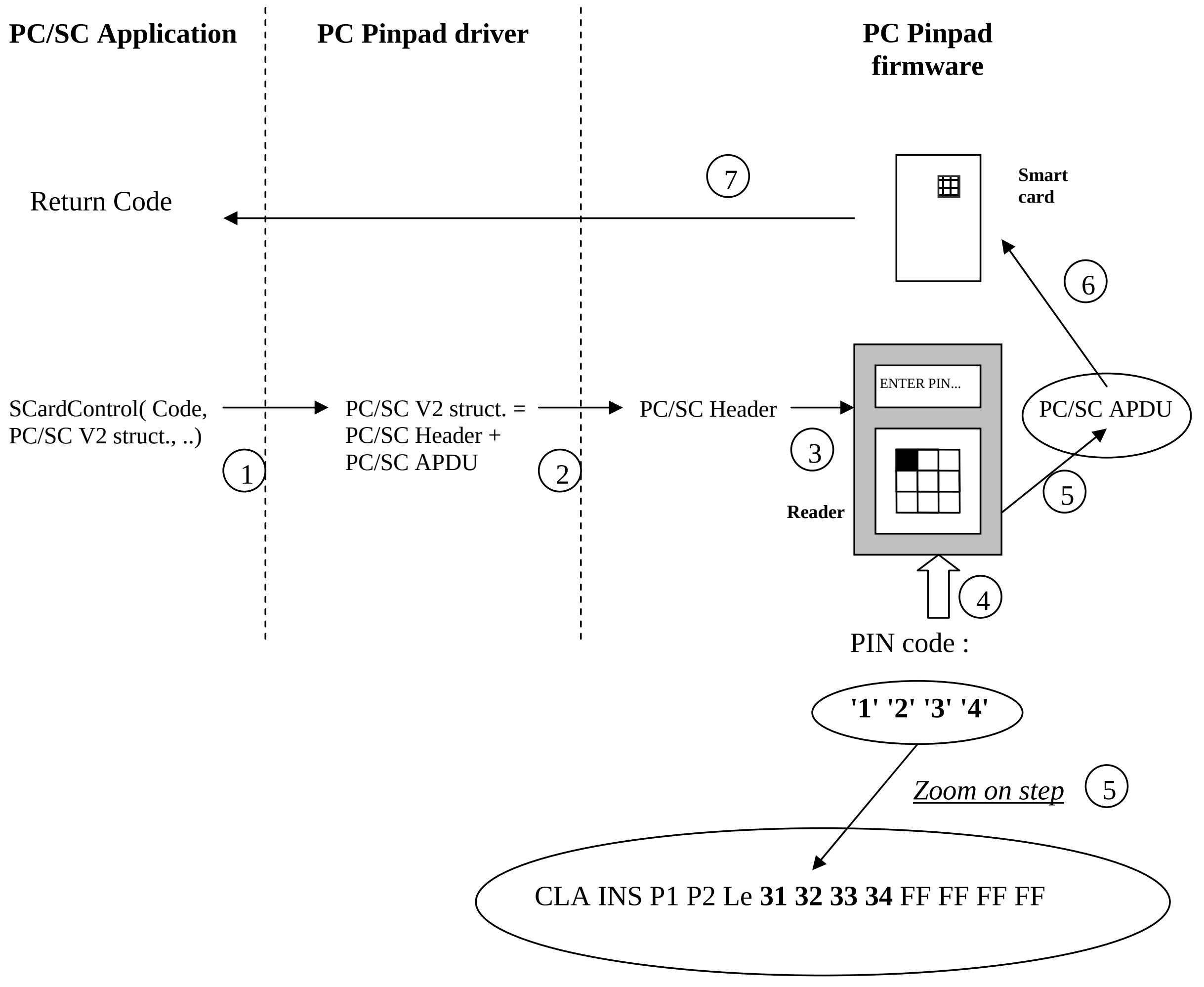
- dwControlCode: it is the command control code. This control code unique for each command has been retrieved in the return parameter of the GET_FEATURE_REQUEST() command.
- lpInBuffer: Input data depending the type of command sent. This data is a PC/SC V2 structure which can be divided in 2 parts:
    - PC/SC header part which contain information about time out, PIN formatting, number of message to be displayed, etc …
    - PC/SC APDU part which contain the APDU sent to the smart card after secure PIN entry. This part is totally dependant from the smart card. It contains CLA INS P1 P2 Le and enough padding bytes for PIN code insertion (0xFF 0xFF ….). This padding is smart card dependant.

Return code is sent by SCardControl( ) indicating if command has been correctly executed.

Each command processed by the GemPC Pinpad follows the same steps described here below:

1/ PC/SC application sends a ScardControl () command with the appropriate control code and PC/SC V2 structure.
2/ PC Pinpad driver retrieves the PC/SC V2 structure and send this structure to the PC Pinpad firmware through the USB port.
3/ PC Pinpad firmware displays the message sequence indicated in the PC/SC header part for PIN entry (for instance: ENTER PIN, CONFIRM PIN, …)
4/ End-user enter the PIN code.
5/ PC Pinpad firmware retrieves the PIN code and inserts it in the PC/SC APDU part. This PIN insertion is made by following instruction indicated in the PC/SC header part (offset, length, coding …)
6/ this APDU is sent to the smart card.
7/ the smart card return code is sent back to the PC/SC V2 application

We have to notice that the PIN code never leaves the PC Pinpad reader.



Secure PIN Commands

All the Secure PIN commands described in this chapter are compliant to [PC/SC].

All the Secure PIN commands are available to the user by ***SCardControl*** function (provided by PC/SC). SCardControl is only used when the smartcard resource manager is being used and after a successful call to ScardConnect has been made.

The ***SCardControl*** function must be called with the following parameters:

SCardControl*(hCard,*
dwControlCode,
lpInBuffer,
nInBufferSize,
lpOutBuffer,
nOutBufferSize,
lpBytesReturned) ;

Where:

Parameters	Remark
hCard	is the *SCARDHANDLE* value returned by SCardConnect
dwControlCode*	is the specific Gemalto control code defining the operation to perform.
lpInBuffer	is a pointer to the input buffer, specific to the dwControlCode
nInBufferSize	is the size, in bytes, of the input buffer
lpOutBuffer	is a pointer to the output buffer, specific to the dwControlCode. Must point to a valid buffer
nOutBufferSize	is the size, in bytes, of the output buffer
lpBytesReturned	is a pointer to a DWORD variable that receives the actual count of bytes returned by the function in the output buffer. Must be non-NULL pointer to a valid value

The specific parameters (typically lpInBuffer and lpOutBuffer) will be described for each control code value.

*For Windows implementation: each dwControlCode consists of a couple of 2 values used by the SCARD_CTL_CODE macro available in winsmcrd.h:

```
#define IOCTL_GET_FEATURE_REQUEST         SCARD_CTL_CODE(3400)
```

Get Feature Request

The driver to send TLV data to indicate which IOCTL is supported by the reader. In our implementation the IOCTL_SMARTCARD_PCSC_VERIFY_PIN and the IOCTL_SMARTCARD_MODIFY_PIN values are returned in the TLV data.

IOCTL_GET_FEATURE_REQUEST (3400 / 0xD48)		
Main Parameters	**in/out**	**Remark**
lpInBuffer	in	Pointer to an input buffer. This buffer is not used by the command.
lpOutBuffer	out	Pointer to the output buffer. Must be non-NULL and large enough to accommodate the TLV data sent by the driver.

Status	**Meaning**
STATUS_SUCCESS	The transmission was successful with the reader.
STATUS_IO_TIMEOUT	The operation has timed out.
STATUS_INVALID_PARAMETER	At least one parameter is not correct

Example: In our driver, we support the Verify Pin (IOCTL = 2060 decimal) and Modify Pin (IOCTL = 2061 decimal) functionalities. So, the IOCTL_GET_FEATURE_REQUEST answer is: 06 04 00 31 20 30 07 04 00 31 30 20 34.

Note: a code example can be found at the end of the document.

Verify Pin command

Using this IOCTL_SMARTCARD_PC_SC_VERIFY_PIN causes:
The reader to receive data (PIN) from keyboard in a secure way (without communicating with the host computer),
Integrate this data into a specific APDU and send this APDU to smartcard.
The card reply is returned.

Formatted: Bullets and Numbering

IOCTL_SMARTCARD_PC_SC_VERIFY_PIN (2060 / 0x80C)		
Main Parameters	**in/out**	**Remark**
lpInBuffer	in	Pointer to a buffer containing a PIN_VERIFY_STRUCTURE compliant to [PC/SC]
lpOutBuffer	out	Pointer to the output buffer. Must be non-NULL and large enough to accommodate the card answer to secure pin APDU.

Status	**Meaning**
STATUS_SUCCESS	The transmission was successful with the reader.
STATUS_IO_TIMEOUT	The operation has timed out.
STATUS_INVALID_PARAMETER	At least one parameter is not correct

Note: Detailed information about each structure element can be found in [PC/SC].
A code example can be found at the end of the document.

Example of the buffer (lpInBuffer) to send :

Offset	Field	Value	Remark
0	bTimeOut1	0x00	Number of seconds. Default timeout value (seconds)*
1	bTimeOut2	0x00	Number of seconds. Default timeout value (seconds)*
2	bmFormatString	0x82	PIN format options
3	bmPinBlockString	0X04	Length in bytes of PIN block to present the APDU command
4	bmPinLenghtFormat	0x00	Allows the insertion of the PIN length in the APDU command
5	wPinMaxExtraDigit	0x0404	Min value/Max value
7	EntryValidationCondition	0x02	Validation key pressed
8	bNumberMessage	0x01	(Always>0) number of message for PIN verification management
9	wLangID	0x04	English
10	"	0x09	
11	bMsgIndex	0x00	Message index in CCID reader; should be 0.
12	bTeoPrologue[3]	0x00…	T=1 I-block prologue field to use
15-18	ulDataLength	0x0D000000	Data to send to the ICC (APDU length)
Start of APDU for a GPK Gemsafe card			
19	CLA	0x00	APDU header
20	INS	0x20	Verify PIN instruction
21	P1	0x00	
22	P2	0x00	
23	Lc	0x08	Length of secure pin APDU
24	Data[0]	0xFF	Padding characters : these characters will be filled by the reader when the user enters the PIN code
25	Data[1]	0XFF	
26	Data[2]	0XFF	
27	Data[3]	0XFF	
28	Data[4]	0XFF	
29	Data[5]	0XFF	
30	Data[6]	0XFF	
31	Data[7]	0XFF	

*Note: The driver takes the max between bTimeOut1 and bTimeOut2 and the CCID read time out default value. This value is used by the driver before the first entered key and between each entered keys

Some parameters in the PC/SC V2 structure have the following limitations:

- The minimum PIN length (*wPINMaxExtraDigi*t) must be upper than 4.
- The maximum PIN length (*wPINMaxExtraDigi*t) must be lower than 8.
- The *bNumberMessage* parameter must be between 1 and 3 depending on the number of message displayed on the Pinpad. So, this parameter does not support the 0x00 value or the 0xFF value.
- The *bEntryValidationCondition* parameter only supports validation by key pressed (0x02). All other value, i.e. validation by "Max size reached" (0x01) and validation by "Time out" (0x04) are not allowed.

Modify Pin command

Using this IOCTL_SMARTCARD_PC_SC_MODIFY_PIN causes:

Deleted: WARNING –

Deleted: ¶

Formatted: Bullets and Numbering

The reader to receive data (PIN) from keyboard in a secure way (without communicating with the host computer),
Integrate this data into a specific APDU and send this APDU to smartcard.
The card reply is returned.

IOCTL_SMARTCARD_PC_SC_MODIFY_PIN (2061 / 0x80D)		
Main Parameters	**in/out**	**Remark**
lpInBuffer	in	Pointer to a buffer containing a PIN_MODIFY_STRUCTURE compliant to [PC/SC]
lpOutBuffer	out	Pointer to the output buffer. Must be non-NULL and large enough to accommodate the card answer to modify pin APDU.

Status	**Meaning**
STATUS_SUCCESS	The transmission was successful with the reader.
STATUS_IO_TIMEOUT	The operation has timed out.
STATUS_INVALID_PARAMETER	At least one parameter is not correct

Note: Detailed information about each structure element can be found in [PC/SC].
See code example at the end of the document.

Example of the buffer (lpInBuffer) to send :

Offset	Field	Value	Remark
0	bTimeOut1	0x00	Number of seconds. Default timeout value (seconds)*
1	bTimeOut2	0x00	Number of seconds. Default timeout value (seconds)*
2	bmFormatString	0x82	PIN format options
3	bmPinBlockString	0X04	Length in bytes of PIN block to present the APDU command
4	bmPinLenghtFormat	0x00	Allows the insertion of the PIN length in the APDU command
5	bInsertionOffsetOld	0x00	
6	bInsertionOffsetNew	0x08	
7	wPinMaxExtraDigit	0x0404	Min value/Max Value
9	bEntryValidationCondition	0x02	Validation key pressed
10	bConfirmPin	0x02	PIN confirmation required
11	bNumberMessage	0x02	(Always>0) message for PIN management
12	wLangID	0x0409	English
14	bMsgIndex1	0x00	Should be 0 (not used by reader).
15	bMsgIndex2	0x00	Should be 0 (not used by reader).
16	bMsgIndex3	0x00	Should be 0 (not used by reader).
17	bTeoPrologue[3]	0x00…	T=1 I-block prologue field to use
20-23	ulDataLength	0x15000000	Data to send to the ICC (APDU length)
	Start of APDU for a GPK Gemsafe card		
24	CLA	0x00	APDU header
25	INS	0x24	Modify PIN instruction
26	P1	0x00	
27	P2	0x00	
28	Lc	0x08	Length of secure pin APDU

29	Data[0]	0xFF	Padding characters : these characters will be filled by the reader when the user enters the old PIN code
..	..	..	
36	Data[7]	0xFF	
37	Data[8]	0xFF	Padding characters : these characters will be filled by the reader when the user enters the new PIN code
..	..	..	
44	Data[15]	0xFF	

*Note: The driver takes the max between bTimeOut1 and bTimeOut2 and the CCID read time out default value. This value is used by the driver before the first entered key and between each entered keys.

Some parameters in the PC/SC V2 structure have the following limitations:

- The minimum PIN length (*wPINMaxExtraDigi*t) must be upper than 4.
- The maximum PIN length (*wPINMaxExtraDigi*t) must be lower than 8.
- The *bNumberMessage* parameter must be between 1 and 3 depending on the number of message displayed on the Pinpad. So, this parameter does not support the 0x00 value or the 0xFF value.
- The *bEntryValidationCondition* parameter only supports validation by key pressed (0x02). All other value, i.e. validation by “Max size reached” (0x01) and validation by “Time out” (0x04) are not allowed.

Example of message choice for Modify Pin Command

Parameters in the PC/SC V2 structure allow modifying the number and the type of the message displayed when executing the Modify Pin command.
All the possibilities are given in the table described below:

bConfirmPin	0x03	0x02	0x01	0x00
bNumberMessage	0x03	0x02	0x02	0x01
Messages seen on Pinpad display	Enter Pin New Pin Confirm Pin	Enter Pin New Pin	New Pin* Confirm Pin*	New Pin*

*In these two cases, old PIN is not asked by the Pinpad but do not forget to put the old PIN value in the APDU command.

PC Pinpad reader message management:

Loading messages by the multi-language installer:

The installation of the PC Pinpad driver is made by a multi-language installer. You can download this installer in connecting to the Gemalto Web site: http://support.gemalto.com/gemdownload/readers/index.aspx

Installer allows the customization of messages displayed on the PC Pinpad, it detects in the Operating System the local language and asks the user if he wants the same language for messages displayed on the PC Pinpad.

All the languages listed in the list below can be detected by the installer:

- **English**
- **French**
- **German**
- **Spanish**
- **Italian**
- **Dutch**
- **Turkish**

If the language is not known by the installer, default language chosen by the installer is English.

Installer loads in the "*KEY_LOCAL_MACHINE\SOFTWARE\Gemplus\GemPCPinPad*" windows key registry 10 strings with the language chosen by the user. By default, language is English and the following messages are present in the registry

PinPadString1	Enter PIN
PinPadString2	New PIN
PinPadString3	Confirm PIN
PinPadString4	PIN OK
PinPadString5	Incorrect PIN
PinPadString6	Time Out
PinPadString7	* retries left
PinPadString8	Insert Card
PinPadString9	Card Error
PinPadString10	Locked Card

When the driver starts, it loads the Pinpad strings found in this registry in the PC Pinpad reader.

Example of Pinpad strings for different languages:

String number	English	French	German	Spanish	Italian	Dutch	Turkish
1	Enter PIN	Entrer PIN	PIN eingeben	Introducir PIN	Inserire PIN	Inbrengen code	PIN Giriniz
2	New PIN	Nouveau PIN	Neue PIN	Nuevo PIN	Nuovo PIN	Nieuwe code	Yeni PIN
3	Confirm PIN	Confirmer PIN	PIN bestatigen	Confirmar PIN	Confermare PIN	Bevestig code	PIN Onayla
4	PIN OK	PIN OK	PIN korrekt	PIN OK	PIN Corretto	Code aanvaard	PIN OK
5	Incorrect PIN	PIN incorrect	Falsche PIN	PIN Incorrecto	PIN Errato	Foute code	Yanlis PIN
6	Time Out	Delai depasse	Zeit abgelaufen	Tiempo Finalizado	Tempo scaduto	Time Out	Zaman Asimi
7	* retries left	* essai restant	*. Versuche ubrig	quedan * ensayos	* prove rimaste	Nog * Pogingen	* deneme kaldi
8	Insert Card	Inserer carte	Karte einstecken	Introducir Tarjeta	Inserire Carta	Kaart inbrengen	Karti Takiniz
9	Card Error	Erreur carte	Fehler Karte	Error en Tarjeta	Errore Carta	Kaart fout	Kart Hatasi
10	Locked card	Carte bloquee	Karte gesperrt	Tarj. Bloqueada	Carta bloccata	Kaart blok	Kart Kilitli

Note: these strings can be overloaded in a PS/SC application if you use the command described in the chapter after. See code example written in C language.

Loading messages by a PC/SC application

PC Pinpad messages (Enter PIN, Confirm PIN, etc …) displayed on the screen are stored in a message table inside the reader memory. This message table can contain up to 10 messages of 16 characters at maximum.
In the initial PC Pinpad configuration, messages are in English in the following order:
- Enter PIN
- New PIN
- Confirm PIN
- PIN OK
- Incorrect PIN
- Time Out
- * retries left
- Insert Card
- Card Error
- Locked card

A user can customize these messages (language choice) in loading a new message table.

This customization is made by using the IOCTL_VENDOR_IFD_EXCHANGE (2058 decimal) control code with the SCardControl PC/SC function.
Parameter lpInBuffer must contain the message table encapsulated in a reader load command with the following format:
0xB2 0xA0 address (1 byte) 0x4D 0x4C (validation code memory) message table.

See code example in C Language at the end of the document for using this command.

Example 1: Filling the lpInBuffer parameter with the following values allows to load a French message table.

Data (in hexadecimal)	Interpretation
B2	Load command for the reader
A0	memory address
00	Offset address where to store the data
4D 4C	Mandatory memory validation code (offset 0)
45 6E 74 72 65 72 20 50 49 4E 20 20 20 20 20 20	message 1 “Entrer PIN”
45 6E 74 72 65 72 20 6E 6F 75 76 20 50 49 4E 20	message 2 “Entrer nouveau PIN”
43 6F 6E 66 69 72 6D 65 72 20 50 49 4E 20 20 20	message 3 “Confirmer PIN”
50 49 4E 20 4F 4B 20 20 20 20 20 20 20 20 20 20	message 4 “PIN OK”
50 49 4E 20 69 6E 63 6F 72 72 65 63 74 20 20 20	message 5 “PIN incorrect”
44 65 6C 61 69 20 64 65 70 61 73 73 65 20 20 20	message 6 “Delai depasse”
45 73 73 61 69 20 72 65 73 74 61 6E 74 20 20 20	message 7 ” Essai restant”
49 6E 73 65 72 65 72 20 63 61 72 74 65 20 20 20	message 8 “Inserer carte”
45 72 72 65 75 72 20 63 61 72 74 65 20 20 20 20	message 9 “Erreur carte”
43 61 72 74 65 20 64 LL 6F 71 75 65 65 20 20	Message 10 “Carte bloquee”

Example 2 : Filling the lpInBuffer parameter with the following values allows to replace message 1 “Enter PIN” by “Enter PUK”.

Data (in hexadecimal)	Interpretation
B2	Load command for the reader
A0	memory address
02	Offset address where to store the data (jump the memory validation code)
45 6E 74 65 72 20 50 55 4B 20 20 20 20 20 20 20	message 1 “Enter PUK”

Code example for PC/SC V2 part 10 implementation

```
/*
   scardcontrol.c: sample code to use/test SCardControl() API with PC/SC v2.0.2 Part 10
   Copyright (C) 2005              Ludovic Rousseau
   Modified by Didier Bonnardel for GemPC Pinpad loading
   And some minor modifications for Visual compatibility.

   WARNING: This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

           This program has been tested under Visual C++ 6.0 Enterprise Edition with XP SP2.
           It can be builded under a Win32 Console Application Microsoft project.
           Do not forget to include winscard.lib in the link option.


#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <winscard.h>


#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif

#define MAX_ATR_SIZE 256
#define MAX_READERNAME 256
#define MAX_BUFFER_SIZE 256
#define HOST_TO_CCID_16(x) (x)
#define HOST_TO_CCID_32(x) (x)

//Structure definition for PC/SC commands
/*
 * Class 2 reader tags
 */
#define CM_IOCTL_GET_FEATURE_REQUEST SCARD_CTL_CODE(3400)

#define FEATURE_VERIFY_PIN_START  0x01
#define FEATURE_VERIFY_PIN_FINISH 0x02
#define FEATURE_MODIFY_PIN_START  0x03
#define FEATURE_MODIFY_PIN_FINISH 0x04
#define FEATURE_GET_KEY_PRESSED   0x05
#define FEATURE_VERIFY_PIN_DIRECT 0x06
#define FEATURE_MODIFY_PIN_DIRECT 0x07
#define FEATURE_MCT_READERDIRECT  0x08
#define FEATURE_MCT_UNIVERSAL     0x09
#define FEATURE_IFD_PIN_PROP      0x0A
#define FEATURE_ABORT             0x0B

/*Definition for strings loading*/
#define CMD_LOAD_STRING_HEADER    5
#define CMD_LOAD_STRING_MAX_SIZE  16
#define     CMD_LOAD_NB_STRING            10

//Warning : need to BYTE aligned
#pragma pack(1)
/* the structure must be 6-bytes long */
typedef struct
```

```
{
        BYTE tag;
        BYTE length;
        DWORD value;
} PCSC_TLV_STRUCTURE;

typedef struct
{
        BYTE bTimerOut;    /* timeout is seconds (00 means use default timeout) */
        BYTE bTimerOut2; /* timeout in seconds after first key stroke */
        BYTE bmFormatString; /* formatting options */
        BYTE bmPINBlockString; /* bits 7-4 bit size of PIN length in APDU,
                          * bits 3-0 PIN block size in bytes after
                          * justification and formatting */
        BYTE bmPINLengthFormat; /* bits 7-5 RFU,
                           * bit 4 set if system units are bytes, clear if
                           * system units are bits,
                           * bits 3-0 PIN length position in system units */
        USHORT wPINMaxExtraDigit; /* 0xXXYY where XX is minimum PIN size in digits,
                             and YY is maximum PIN size in digits */
        BYTE bEntryValidationCondition; /* Conditions under which PIN entry should
                                * be considered complete */
        BYTE bNumberMessage; /* Number of messages to display for PIN verification */
        USHORT wLangId; /* Language for messages */
        BYTE bMsgIndex; /* Message index (should be 00) */
        BYTE bTeoPrologue[3]; /* T=1 block prologue field to use (fill with 00) */
        ULONG ulDataLength; /* length of Data to be sent to the ICC */
        BYTE abData[1]; /* Data to send to the ICC */
} PIN_VERIFY_STRUCTURE;

typedef struct
{
        BYTE bTimerOut;    /* timeout is seconds (00 means use default timeout) */
        BYTE bTimerOut2; /* timeout in seconds after first key stroke */
        BYTE bmFormatString; /* formatting options */
        BYTE bmPINBlockString; /* bits 7-4 bit size of PIN length in APDU,
                          * bits 3-0 PIN block size in bytes after
                          * justification and formatting */
        BYTE bmPINLengthFormat; /* bits 7-5 RFU,
                           * bit 4 set if system units are bytes, clear if
                           * system units are bits,
                           * bits 3-0 PIN length position in system units */
        BYTE bInsertionOffsetOld; /* Insertion position offset in bytes for
                              the current PIN */
        BYTE bInsertionOffsetNew; /* Insertion position offset in bytes for
                              the new PIN */
        USHORT wPINMaxExtraDigit;
                           /* 0xXXYY where XX is minimum PIN size in digits,
                             and YY is maximum PIN size in digits */
        BYTE bConfirmPIN; /* Flags governing need for confirmation of new PIN */
        BYTE bEntryValidationCondition; /* Conditions under which PIN entry should
                                * be considered complete */
        BYTE bNumberMessage; /* Number of messages to display for PIN verification*/
        USHORT wLangId; /* Language for messages */
        BYTE bMsgIndex1; /* index of 1st prompting message */
        BYTE bMsgIndex2; /* index of 2d prompting message */
        BYTE bMsgIndex3; /* index of 3d prompting message */
        BYTE bTeoPrologue[3]; /* T=1 block prologue field to use (fill with 00) */
        ULONG ulDataLength; /* length of Data to be sent to the ICC */
        BYTE abData[1]; /* Data to send to the ICC */
} PIN_MODIFY_STRUCTURE;


#define IOCTL_SMARTCARD_VENDOR_IFD_EXCHANGE     SCARD_CTL_CODE(2048)

/* PCSC error message pretty print */
#define PCSC_ERROR_EXIT(rv, text) \
if (rv != SCARD_S_SUCCESS) \
{ \
        printf(text ": Error status (0x%lX)\n", rv); \
        goto end; \
} \
else \
        printf(text ": OK\n\n");
```

```
#define PCSC_ERROR_CONT(rv, text) \
if (rv != SCARD_S_SUCCESS) \
	printf(text ": Error status (0x%lX)\n", rv); \
else \
	printf(text ": OK\n\n");

int main(int argc, char *argv[])
{
	ULONG rv;
	SCARDCONTEXT hContext;
	DWORD dwReaders;
	LPTSTR mszReaders;
	char *ptr, **readers = NULL;
	int nbReaders;
	SCARDHANDLE hCard;
	DWORD dwActiveProtocol, dwReaderLen, dwState, dwProt, dwAtrLen;
	BYTE pbAtr[MAX_ATR_SIZE] = "";
	char pbReader[MAX_READERNAME] = "";
	int reader_nb;
	unsigned int i, j, k;
	unsigned char bSendBuffer[MAX_BUFFER_SIZE];
	unsigned char bRecvBuffer[MAX_BUFFER_SIZE];
	DWORD length, lengthRes;
	DWORD verify_ioctl = 0;
	DWORD modify_ioctl = 0;
	PCSC_TLV_STRUCTURE *pcsc_tlv;

	char *ppaPinpadString[] = {"Hello Enter PIN", "New PIN", "Confirm PIN", "PIN OK", "Incorrect PIN",
		"Time Out", "* retries left", "Insert Card", "Card Error", "Locked card"};

	char acCommandLoad[CMD_LOAD_STRING_HEADER +
	CMD_LOAD_STRING_MAX_SIZE*CMD_LOAD_NB_STRING];

	int offset;
	PIN_VERIFY_STRUCTURE *pin_verify;
	PIN_MODIFY_STRUCTURE *pin_modify;

	printf("SCardControl sample code with examples of PCSC V2 part 10 commands\n");

	rv = SCardEstablishContext(SCARD_SCOPE_SYSTEM, NULL, NULL, &hContext);
	if (rv != SCARD_S_SUCCESS)
	{
		printf("SCardEstablishContext: Cannot Connect to Resource Manager %lX\n", rv);
		return 1;
	}

	/* Retrieve the available readers list */
	rv = SCardListReaders(hContext, NULL, NULL, &dwReaders);
	if (rv != SCARD_S_SUCCESS)
	{
		printf("SCardListReader: %lX\n", rv);
	}

	mszReaders = malloc(sizeof(char)*dwReaders);
	if (mszReaders == NULL)
	{
		printf("malloc: not enough memory\n");
		goto end;
	}

	rv = SCardListReaders(hContext, NULL, mszReaders, &dwReaders);
	if (rv != SCARD_S_SUCCESS)
		printf("SCardListReader: %lX\n", rv);

	/* Extract readers from the null separated string and get the total
	 * number of readers */
	nbReaders = 0;
	ptr = mszReaders;
	while (*ptr != '\0')
	{
		ptr += strlen(ptr)+1;
		nbReaders++;
	}

	if (nbReaders == 0)
```

```
{
	printf("No reader found\n");
	goto end;
}

/* allocate the readers table */
readers = calloc(nbReaders, sizeof(char *));
if (NULL == readers)
{
	printf("Not enough memory for readers[]\n");
	goto end;
}

/* fill the readers table */
nbReaders = 0;
ptr = mszReaders;
while (*ptr != '\0')
{
	printf("%d: %s\n", nbReaders, ptr);
	readers[nbReaders] = ptr;
	ptr += strlen(ptr)+1;
	nbReaders++;
}

if (argc > 1)
{
	reader_nb = atoi(argv[1]);
	if (reader_nb < 0 || reader_nb >= nbReaders)
	{
		printf("Wrong reader index: %d\n", reader_nb);
		goto end;
	}
}
else
	reader_nb = 0;

/* connect to a reader (even without a card) */
dwActiveProtocol = -1;
rv = SCardConnect(hContext, readers[reader_nb], SCARD_SHARE_DIRECT, 0, &hCard, &dwActiveProtocol);
printf(" Protocol: %ld on reader: %s\n", dwActiveProtocol, readers[reader_nb]);
PCSC_ERROR_EXIT(rv, "SCardConnect")

/* get GemPC firmware */
printf("==> Get GemPC Firmware\n");

/* This is specific to Gemalto readers */
bSendBuffer[0] = 0x02;
rv = SCardControl(hCard, IOCTL_SMARTCARD_VENDOR_IFD_EXCHANGE, bSendBuffer,
	1, bRecvBuffer, sizeof(bRecvBuffer), &length);

printf(" Firmware: ");
for (i=0; i<length; i++)
	printf("%02X ", bRecvBuffer[i]);
printf("\n");

bRecvBuffer[length] = '\0';
printf(" Firmware: %s (length %ld bytes)\n", bRecvBuffer, length);

PCSC_ERROR_CONT(rv, "SCardControl")


/* does the reader support PIN verification? */
printf("==> Get Feature Request\n");

rv = SCardControl(hCard, CM_IOCTL_GET_FEATURE_REQUEST, NULL, 0,
	bRecvBuffer, sizeof(bRecvBuffer), &length);

printf(" TLV length = (%ld) sizeof(PCSC_TLV_STRUCTURE) = %ld: ", length,
sizeof(PCSC_TLV_STRUCTURE));
for (i=0; i<length; i++)
	printf("%02X ", bRecvBuffer[i]);
printf("\n");

if (length % sizeof(PCSC_TLV_STRUCTURE))
{
		printf("Inconsistent result! Bad TLV values!\n");
		goto end;
}

/* get the number of elements instead of the complete size */
length /= sizeof(PCSC_TLV_STRUCTURE);

pcsc_tlv = (PCSC_TLV_STRUCTURE *)bRecvBuffer;
for (i = 0; i < length; i++)
{
		if (pcsc_tlv[i].tag == FEATURE_VERIFY_PIN_DIRECT)
		{
				/*Recover the control code*/
				/*and do the change big endian -> little endian*/
				for ( j = 0; j < 4; j ++)
						((BYTE *)&verify_ioctl)[j] = (BYTE)((pcsc_tlv[i].value >> (24 - 8*j) ) & 0xFF);
				printf("FEATURE_VERIFY_PIN_DIRECT implemented in the reader \n");
		}
		if (pcsc_tlv[i].tag == FEATURE_MODIFY_PIN_DIRECT)
		{
				/*Recover the control code*/
				/*and do the change big endian -> little endian*/
				for ( j = 0; j < 4; j ++)
						((BYTE *)&modify_ioctl)[j] = (BYTE)((pcsc_tlv[i].value >> (24 - 8*j) ) & 0xFF);
				printf("FEATURE_MODIFY_PIN_DIRECT implemented in the reader \n");
		}
}

PCSC_ERROR_CONT(rv, "SCardControl")



/*Load strings in the pinpad memory*/
k = 0;
printf("==> Load strings in the Pinpad memory\n");
acCommandLoad[k++] = (char)0xB2;
acCommandLoad[k++] = (char)0xA0;
acCommandLoad[k++] = 0x00;
acCommandLoad[k++] = 0x4D;//Show that data is valid
acCommandLoad[k++] = 0x4C;

for ( i = 0; i < CMD_LOAD_NB_STRING; i ++)
{
		/*fill the command array with string*/
		for ( j = 0; j < min( CMD_LOAD_STRING_MAX_SIZE, strlen(ppaPinpadString[i]) ); j++ )
				acCommandLoad[k++] = ppaPinpadString[i][j];
		/*pad with 0x20*/
		while ( j++ < CMD_LOAD_STRING_MAX_SIZE )
				acCommandLoad[k++] = 0x20;
}

rv = SCardControl(hCard, IOCTL_SMARTCARD_VENDOR_IFD_EXCHANGE, acCommandLoad,
		sizeof( acCommandLoad ), bRecvBuffer, sizeof(bRecvBuffer), &length);

PCSC_ERROR_CONT(rv, "*** SCardControl : Load GemPC Pinpad strings ***")


/* Get card status */
printf("==> Get card status\n");
dwAtrLen = sizeof(pbAtr);
dwReaderLen = sizeof(pbReader);
rv = SCardStatus(hCard, pbReader, &dwReaderLen, &dwState, &dwProt,
		pbAtr, &dwAtrLen);
printf(" Reader: %s (length %ld bytes)\n", pbReader, dwReaderLen);
printf(" State: 0x%lX\n", dwState);
printf(" Prot: %ld\n", dwProt);
printf(" ATR (length %ld bytes):", dwAtrLen);
for (i=0; i<dwAtrLen; i++)
		printf(" %02X", pbAtr[i]);
printf("\n");
PCSC_ERROR_CONT(rv, "SCardStatus")


/* connect to a card */
dwActiveProtocol = -1;
```

```
rv = SCardReconnect(hCard, SCARD_SHARE_SHARED,
		SCARD_PROTOCOL_T0|SCARD_PROTOCOL_T1|SCARD_PROTOCOL_DEFAULT,
SCARD_UNPOWER_CARD,
		&dwActiveProtocol);
printf(" Protocol: %ld\n", dwActiveProtocol);
PCSC_ERROR_EXIT(rv, "SCardReconnect")


if (0 == verify_ioctl)
{
		printf("Reader %s does not support PIN verification\n",
				readers[reader_nb]);
		goto end;
}

/* verify PIN */
printf("==> Secure verify PIN\n");
pin_verify = (PIN_VERIFY_STRUCTURE *)bSendBuffer;

/* PC/SC v2.0.2 Part 10 PIN verification data structure */
		pin_verify -> bTimerOut = 30; /*Time out between key stroke = max(bTimerOut, bTimerOut2). Must be
between 15 and 40 sec.*/
pin_verify -> bTimerOut2 = 00;
pin_verify -> bmFormatString = 0x82;
pin_verify -> bmPINBlockString = 0x04;
pin_verify -> bmPINLengthFormat = 0x00;
pin_verify -> wPINMaxExtraDigit = HOST_TO_CCID_16(0x0408); /* Min Max */
pin_verify -> bEntryValidationCondition = 0x02;		/* validation key pressed */
pin_verify -> bNumberMessage = 0x01;
pin_verify -> wLangId = HOST_TO_CCID_16(0x0904);
pin_verify -> bMsgIndex = 0x00;
pin_verify -> bTeoPrologue[0] = 0x00;
pin_verify -> bTeoPrologue[1] = 0x00;
pin_verify -> bTeoPrologue[2] = 0x00;
/* pin_verify -> ulDataLength = 0x00; we don't know the size yet */

/* WARNING : this APDU is for a GemXplore 'Xpresso smartcard => APDU: A0 20 00 01 08 FF FF FF FF FF FF
FF FF */
offset = 0;
pin_verify -> abData[offset++] = 0xA0;	/* CLA */ /**********************************/
pin_verify -> abData[offset++] = 0x20;	/* INS: VERIFY */
pin_verify -> abData[offset++] = 0x00;	/* P1 */
pin_verify -> abData[offset++] = 0x01;	/* P2 */
pin_verify -> abData[offset++] = 0x08;	/* Lc: 8 data bytes */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> abData[offset++] = 0xFF;	/* 'FF' */
pin_verify -> ulDataLength = HOST_TO_CCID_32(offset);		/* APDU size */

length = sizeof(PIN_VERIFY_STRUCTURE) + offset -1;		/* -1 because PIN_VERIFY_STRUCTURE
contains the first byte of abData[] */

printf(" command:");
for (i=0; i<length; i++)
		printf(" %02X", bSendBuffer[i]);
printf("\n");
printf("Enter your PIN in the Pinpad reader keypad...\n");
fflush(stdout);
rv = SCardControl(hCard, verify_ioctl, bSendBuffer,
		length, bRecvBuffer, sizeof(bRecvBuffer), &lengthRes);

if ( rv == SCARD_S_SUCCESS )
{
		printf(" card response:");
		for (i=0; i<lengthRes; i++)
				printf(" %02X", bRecvBuffer[i]);
		printf("\n");
}
PCSC_ERROR_CONT(rv, "SCardControl")
printf("\n");
```

```
/* Modify PIN */
if (0 == modify_ioctl)
{
		printf("Reader %s does not support PIN modification\n",
				readers[reader_nb]);
		goto end;
}

/* check if the reader supports Modify PIN */
printf("==> Secure modify PIN\n");
pin_modify = (PIN_MODIFY_STRUCTURE *)bSendBuffer;

/* PC/SC v2.0.2 Part 10 PIN verification data structure */
pin_modify -> bTimerOut = 0x00;
pin_modify -> bTimerOut2 = 0x00;
pin_modify -> bmFormatString = 0x82;
pin_modify -> bmPINBlockString = 0x04;
pin_modify -> bmPINLengthFormat = 0x00;
pin_modify -> bInsertionOffsetOld = 0x00;  /* offset from APDU start */
pin_modify -> bInsertionOffsetNew = 0x08; /* offset from APDU start */
pin_modify -> wPINMaxExtraDigit = HOST_TO_CCID_16(0x0408);		/* Min Max */
pin_modify -> bConfirmPIN = 0x02;/* b1 set = current PIN entry requested */
pin_modify -> bEntryValidationCondition = 0x02;		/* validation key pressed */
pin_modify -> bNumberMessage = 0x02;
pin_modify -> wLangId = HOST_TO_CCID_16(0x0904);
pin_modify -> bMsgIndex1 = 0x00;
pin_modify -> bMsgIndex2 = 0x00;
pin_modify -> bMsgIndex3 = 0x00;
pin_modify -> bTeoPrologue[0] = 0x00;
pin_modify -> bTeoPrologue[1] = 0x00;
pin_modify -> bTeoPrologue[2] = 0x00;
/* pin_modify -> ulDataLength = 0x00; we don't know the size yet */

/*WARNING : this APDU is for a GemXplore 'Xpresso smartcard => APDU: A0 24 00 01 10 FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF */
offset = 0;
pin_modify -> abData[offset++] = 0xA0;	/* CLA *//*********************************************/
pin_modify -> abData[offset++] = 0x24;	/* INS: CHANGE/UNBLOCK */
pin_modify -> abData[offset++] = 0x00;	/* P1 */
pin_modify -> abData[offset++] = 0x01;	/* P2 */
pin_modify -> abData[offset++] = 0x10;	/* Lc: 2x8 data bytes */
pin_modify -> abData[offset++] = 0xFF;	/* FF old PIN */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF new PIN */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> abData[offset++] = 0xFF;	/* FF */
pin_modify -> ulDataLength = HOST_TO_CCID_32(offset);		/* APDU size */

length = sizeof(PIN_MODIFY_STRUCTURE) + offset -1;		/* -1 because PIN_MODIFY_STRUCTURE
contains the first byte of abData[] */

printf(" command:");
for (i=0; i<length; i++)
		printf(" %02X", bSendBuffer[i]);
printf("\n");
printf("Enter your PIN in the Pinpad reader keypad...\n");
fflush(stdout);
rv = SCardControl(hCard, modify_ioctl, bSendBuffer,
		length, bRecvBuffer, sizeof(bRecvBuffer), &lengthRes);

if ( rv == SCARD_S_SUCCESS )
{
		printf(" card response:");
```

```
			for (i=0; i<lengthRes; i++)
				printf(" %02X", bRecvBuffer[i]);
			printf("\n");
		}
		PCSC_ERROR_CONT(rv, "SCardControl")

		/* card disconnect */
		rv = SCardDisconnect(hCard, SCARD_UNPOWER_CARD);
		PCSC_ERROR_CONT(rv, "SCardDisconnect")

		end:
		/* We try to leave things as clean as possible */
		rv = SCardReleaseContext(hContext);
		if (rv != SCARD_S_SUCCESS)
			printf("SCardReleaseContext: Error status (0x%lX)\n", rv);

		/* free allocated memory */
		free(mszReaders);
		free(readers);

		return 0;
} /* main *
```