



USER MANUAL

SecurePIN and SmartPIN

API Reference Guide PED DLL V3.09

80071505-001-F
Jan 21, 2010

User Manual, SecurePIN/ SmartPIN API Guide

Revision History

Revision	Date	Description of Changes	By
A	09/05/2007	Initial Release, Replaces 80071504-001 and used for all models of SecurePIN	GS
B	02/05/2009	Deleted SecurePIN model # 150, 180. Removed info on Smart Card related description and operation	JW
C	04/13/2009	Corrected the API version number, v3.07. Recovered Smart Card related operation	JW
D	08/28/2009	Added PIN_GetClearText and Text_AddKeyHandle for clear text operation	JW
E	09/29/2009	Changed the document name to SecurePIN and SmartPIN API Reference Guide	JW
F	01/21/2010	Added the PIN_GetClearText function for SecurePIN130	JW

This document contains proprietary information of ID TECH. Its receipt or possession does not convey any rights to reproduce or disclose its contents or to manufacture, use or sell anything it may describe. Reproduction, disclosure, or use without specific written authorization from ID TECH is strictly forbidden.

User Manual, SecurePIN/ SmartPIN API Guide

Software License Agreement

CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE USING OR INSTALLING THE SOFTWARE. YOUR USE OR INSTALLATION OF THE SOFTWARE PRESUMES YOUR AGREEMENT WITH AND ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE AND RELATED DOCUMENTATION TO – ID TECH Support, 10721 Walker Street, Cypress, CA 90630.

TERMS, CONDITIONS AND RESTRICTIONS

ID TECH, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software".

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with ID TECH products.

LICENSEE MAY NOT COPY, MODIFY OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except to archive the Software or to load the Software for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

User Manual, SecurePIN/ SmartPIN API Guide

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded to be free from defects in material or workmanship under normal use. THE SOFTWARE IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall insure to the benefit of International Technologies & Systems Corporation (d/b/a ID TECH), its successors, or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL, VERBAL AND WRITTEN, COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO ID TECH, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ABOVE ADDRESS OR E-MAILED TO: support@idtechproducts.com

Information Provided

The information contained herein is provided to the user as a convenience. While every effort has been made to ensure accuracy, ID TECH is not responsible for damages that might occur because of errors or omissions, including any loss of profit or other commercial damage, nor for any infringements or patents or other rights of third parties that may result from its use. The specifications described herein were current at the time of publication, but are subject to change at any time without prior notice.

User Manual, SecurePIN/ SmartPIN API Guide

CONTENTS

1.0	Introduction	7
1.1	Smart Card Operation.....	7
1.2	Usage	8
2.0	Command Summary	9
3.0	Function Descriptions.....	10
3.1	DLL Version Function	10
3.1.1	GetDllVersion	10
3.2	Communication functions	10
3.2.1	Com_OpenPort	10
3.2.2	Com_ClosePort	10
3.2.3	Com_SetBaud	11
3.2.4	Com_SetDataBit.....	11
3.2.5	Com_SetParity	11
3.2.6	Com_SetStopBit.....	11
3.3	System functions	12
3.3.1	Sys_GetVersion	12
3.3.2	Sys_GetID	12
3.3.3	Sys_GetModel.....	12
3.3.4	Sys_Reset.....	12
3.4	Timer Function.....	13
3.4.1	Time_SetIdleTime	13
3.5	Sound functions.....	13
3.5.1	Sound_Control	13
3.5.2	Sound_Tone.....	13
3.6	Display functions	14
3.6.1	Display_LoadPrompt	14
3.6.2	Display_Prompt	14
3.6.3	Display_Message	14
3.6.4	Display_Clear	14
3.6.5	Display_BacklightSet.....	15
3.6.6	Display_IdlePromptSelect	15
3.7	PIN functions	16
3.7.1	PIN_GetPINBlock.....	16
3.7.2	PIN_GetClearText	17
3.7.3	PIN_SetPINLength	18
3.7.4	PIN_SetLanguage	18
3.7.5	PIN_SetManualParams	18
3.7.6	PIN_CancelEnter.....	18
3.8	Keypad functions	19
3.8.1	Key_GetFunct	19
3.8.2	PIN_AddKeyHandle.....	19
3.8.3	Text_AddKeyHandle.....	19
3.9	MSR Functions	20

User Manual, SecurePIN/ SmartPIN API Guide

3.9.1	MSR_Restore_Default.....	20
3.9.2	MSR_Set_Reading.....	20
3.9.3	MSR_Data_Status.....	20
3.9.4	MSR_Set_SendOption	21
3.9.5	MSR_Get_Setting.....	21
3.9.6	MSR_Set_StartSentinel.....	22
3.9.7	MSR_Set_EndSentinel.....	22
3.9.8	MSR_Read_Track.....	23
3.9.9	MSR_Set_LRC.....	23
3.9.10	PIN_Verify_Offline.....	23
3.10	Smart Card Operation.....	24
3.10.1	MiniSmart_Operation.....	24
4.0	Example for DLL call:	25
4.1	Single-thread Method	25
4.2	Multi-threads Methods	29
5.0	Return Values	32
6.0	Introduction to MiniSmart Reader Commands	33
6.1	Conventions.....	33
6.2	Communication Protocols.....	34
6.3	The Command Layer	34
6.4	The Transport Layer	35
6.5	The Physical Layer	36
6.6	Microprocessor Card Operation.....	37
6.7	Communication Protocols.....	39
6.8	T=0 Protocol	39
6.9	T=1 Protocol	39
6.10	Command Set.....	39
6.11	Configuration Commands	39
6.12	Power Control Commands Set	43
6.13	Card Interface Command Set.....	45
6.14	Specific Commands for Asynchronous Cards	48
6.15	Appendix A – Status Codes.....	66

1.0 Introduction

SecurePIN are a Personal Identification Number (PIN) Entry Device, which encrypt and securely transmit a PIN to a POS terminal or similar equipment. Some models include MagStripe and Smart card capabilities. The lowest level SecurePIN communication protocol is proprietary, copyrighted, and not for distribution. The communication method to SecurePIN is through a DLL, which is an Application Program Interface (API). The API is used by Host applications. It allows access to SecurePIN functions needed by the application. See "Smart Card Operation" below. This document provides information for using the API, which Support SecurePIN100, SecurePIN130, SecurePIN150, SecurePIN180. SecurePIN100 is PIN entry only, SecurePIN 130 has a MagStripe reader, SecurePIN 150 has a Smart Card reader, & SecurePIN180 provides MagStripe & Smart Card readers.

Separate documents, The SecurePIN User Manual and SecurePIN Quick Start Manual, are available for help in SecurePIN operation and installation.

The API calls are listed in this document. Each function call example provides the operation and parameters as listed. The returned result indicates the function success. See the result table for the returned results.

1.1 Smart Card Operation

The smart card reader is the ID TECH MiniSmart product having its own ASIC. The use of the MiniSmart provides EMVCo approved and industry proven smart card reading capability. The commands, responses, & protocol for the MiniSmart ASIC operation are different from the PIN entry & MagStripe SecurePIN operations.

Communication to the MiniSmart and to the smart card is accomplished as a "passed through" operation provided by this API. The API provides the communication and protocol rapper for SecurePIN by providing the Transport Layer for communication with the MiniSmart reader. MiniSmart commands are sent and responses are received through data blocks.

The MiniSmart communication & command information is in section 6 in this manual.

User Manual, SecurePIN/ SmartPIN API Guide

Availability

The DLL zip folder is available free, on line at www.idtechproducts.com. Two demo projects are zipped together with DLL file. Sample dll calls are provided to show how to use the API commands in single thread method and in a multi-threaded method.

1.2 Usage

This API can be used with any Windows based development tool, such as Visual Studio, Delphi etc. ID TECH provides sample demo program and source code in Delphi, Visual C++ and C#.

When the API is used a Microsoft Visual C++ environment, first:

- Copy Pin.h, PinDll.dll and PinDll.lib to a relevant directory
- Add PinDll.lib to Project->Settings->Link->Object/library modules
- Include the head file Pin.h in the application source

After the above steps, calls can be made to the API. Generally speaking, Com_OpenPort is always the first function called, and Com_Close is the last function called. If the COM Port is not opened, there will be a communication error.

Com_Close and Com_OpenPort must be called again after changing any communication parameters (Baud rate, Stop bit, Parity, Data bit) and the updated parameters must be use to open the COMPort.

2.0 Command Summary

All supported commands are listed below.

Com_OpenPort
Com_ClosePort
Com_SetBaud
Com_SetParity
Com_SetDataBit
Com_SetStopBit
Sys_GetVersion
Sys_GetID
Sys_GetModel
Sys_Reset
Time_SetIdleTime
Sound_Control
Sound_Tone
Display_LoadPrompt
Display_Message
Display_Clear
Display_BacklightSet
Display_Prompt
Display_IdlePromptSelect
PIN_GetPINBlock
PIN_GetClearText
Key_GetFunct
PIN_AddKeyHandle
Text_AddKeyHandle
PIN_CancelEnter
PIN_SetPINLength
PIN_SetLanguage
PIN_SetManualParams
GetDIIVersion
MSR_Restore_Default
MSR_Set_Reading
MSR_Data_Status
MSR_Set_SendOption
MSR_Get_Setting
MSR_Set_StartSentinel
MSR_Set_EndSentinel
MSR_Read_Track
MSR_Set_LRC
PIN_Verify_Offline
MiniSmart_Operation

3.0 Function Descriptions

3.1 DLL Version Function

3.1.1 GetDllVersion

Function: GetDllVersion
Description: Get DLL version number.
Format: BYTE GetDllVersion(char *DllVersion, int *Length)
Parameter: DllVersion ASCII typically
Length The length of version
Return: Section 5
Example GetDllVersion(DllVersion, &length)

3.2 Communication functions

3.2.1 Com_OpenPort

Function: Com_OpenPort
Description: Open port for communication.
Format: BYTE Com_OpenPort(int Comport, long Baud, char Parity, int Stop, int Data);
Parameter: Comport Port for communication
Baud Baud rate
Parity Parity bit
Stop Stop bit
Data Data bit
Note: Baud, Parity, Stop and Data are not needed for USB interface. However, they must be valid parameters. For example, if Baud is set to 0 which is invalid, it returns an error.
Return: Section 5
Example Com_OpenPort(1, 9600, 'E', 1, 7);

3.2.2 Com_ClosePort

Function: Com_ClosePort
Description: Close port for communication.
Format: bool Com_ClosePort()
Parameter: None
Return: Section 5
Example Com_ClosePort()

3.2.3 Com_SetBaud

Function: Com_SetBaud
Description: Set baud rate for RS232 communication.
Format: BYTE Com_SetBaud(long Baud)
Parameter: Baud 1200,2400,4800,9600,14400,19200,28800,38400,
57600 or 115200
Return: Section 5
Example: Com_SetBaud(9600)

3.2.4 Com_SetDataBit

Function: Com_SetDataBit
Description: Set number of data bits for RS232 communication
Format: BYTE Com_SetDataBit(int Databit)
Parameter: Databit 7 or 8
Return: Section 5
Example: Com_SetDataBit(7)

3.2.5 Com_SetParity

Function: Com_SetParity
Description: Set parity for RS232 communication
Format: BYTE Com_SetParity(char Parity)
Parameter: Parity N (None), O (Odd) or E (Even)
Return: Section 5
Example: Com_SetParity('E')

3.2.6 Com_SetStopBit

Function: Com_SetStopBit
Description: Set number of stop bits for RS232 communication
Format: BYTE Com_SetStopBit(int Stopbit)
Parameter: Stopbit 1 or 2
Return: Section 5
Example: Com_SetStopBit(1)

3.3 System functions

3.3.1 Sys_GetVersion

Function: Sys_GetVersion
Description: Get firmware version
Format: Sys_GetVersion(char *Version, int *Length);
Parameter Version 4 bytes ASCII typically
Length The length of version
Return: Section 5
Example Sys_GetVersion(Version, &length)

3.3.2 Sys_GetID

Function: Sys_GetID
Description: Get device hardware ID
Format: BYTE Sys_GetID(char *HardwareID, int *Length);
Parameter: HardwareID 32 characters typically
Length The length of HardwareID
Return: Section 5
Example: Sys_GetID(HardwareID, &length)

3.3.3 Sys_GetModel

Function: Sys_GetModel
Description: Get device model number
Format: BYTE Sys_GetModel(char *ModelNumber, int *Length)
Parameter: ModelNumber 11 or 12 characters typically
Length The length of Model
Return: Section 5
Example: Sys_GetModel(ModelNumber, &length)

3.3.4 Sys_Reset

Function: Sys_Reset
Description: System reset (USB CDC interface doesn't support this function)
Format: BYTE Sys_Reset()
Parameter: None
Return: Section 5
Example: Sys_Reset()

3.4 Timer Function

3.4.1 Time_SetIdleTime

Function: Time_SetIdleTime
Description: Set idle time
Format: BYTE Time_SetIdleTime(int Time)
Parameter: Time 0 to 254 seconds where 0 = never
Return: Section 5
Example: Time_SetIdleTime(200)

3.5 Sound functions

3.5.1 Sound_Control

Function: Sound_Control
Description: Sound control enable/disable set
Format: BYTE Sound_Control(bool Flag);
Parameter: Flag 0 – disable
1 – enable
Return: Section 5
Example: Sound_Control(1)

3.5.2 Sound_Tone

Function: Sound_Tone
Description: Generate tone with specified frequency and duration
Format: BYTE Sound_Tone(long Frequency, long Duration)
Parameter: Frequency 5 < Frequency < 40,000 Hz
Duration 0 =< Duration < 65536 mS.
Return: Section 5
Example: Sound_Tone(2000,300)

3.6 Display functions

3.6.1 Display_LoadPrompt

Function: Display_LoadPrompt
Description: Load display prompts
Format: BYTE Display_LoadPrompt(int Num, char *Message);
Parameter: Num Prompt number 0 to 9
Message No more than 12 characters.
Return: Section 5
Example: char *Message = "ID-TECH-2006";
Display_LoadPrompt(0, Message);

3.6.2 Display_Prompt

Function: Display_Prompt
Description: Display loaded prompt
Format: BYTE Display_Prompt(int Line, int Num);
Parameter: Line 0 (top line) or 1 (bottom line)
Num Prompt number 0 to 9
Return: Section 5
Example: Display_Prompt(0, 1)

3.6.3 Display_Message

Function: Display_Message
Description: Display input message
Format: BYTE Display_Message(int Line, char *Message)
Parameter: Line 0 (top line) or 1 (bottom line)
Message No more than 12 characters.
Return: Section 5
Example: char *Message = "ID-TECH-2006";
Display_Message(1, message)

3.6.4 Display_Clear

Function: Display_Clear
Description: Clear LCD display line
Format: BYTE Display_Clear(int Line)
Parameter: Line 0 (top line) ,1 (bottom line) or 2 (entire LCD)
Return: Section 5
Example: Display_Clear(2);

3.6.5 Display_BacklightSet

Function: Display_BacklightSet
Description: Set backlight on or off
Format: BYTE Display_BacklightSet(bool Flag)
Parameter: Flag 0 – Off
 1 – On
Return: Section 5
Example: Display_BacklightSet(1)

3.6.6 Display_IdlePromptSelect

Function: Display_IdlePromptSelect
Description: Select idle screen prompt
Format: BYTE Display_IdlePromptSelect(int Num1, int Num2)
Parameter: Num1 Prompt number 0 to 9, "B" (Blank) or "D" (Default)
 for line 1
 Num2 Prompt number 0 to 9, "B" (Blank) or "D" (Default)
 for line 2
Return: Section 5
Example: Display_IdlePromptSelect(0, 0)

3.7 PIN functions

3.7.1 PIN_GetPINBlock

Function: PIN_GetPINBlock
Description: Get encrypted PIN Block
Format: BYTE PIN_GetPINBlock(char Key, char *AccountNumber, int Plength, char *EncryptedPIN, int *Length)
Parameter:

Key	D(DUKPT),M(Master/Session)
AccountNumber	Account number: 13 - 17 bytes ASCII number (0-9), the last number is LRC, the function will erase the last number automatically.
Plength	Account number length
EncryptedPIN	Encrypted PIN block. The below is Non-LCD device only: During user pressing keypad, the characters can be '*', 'B', 'C' and '='. After user pressing "Enter", the stored data is encrypted PIN. '*' for numeric key, 'B' for BACKSPACE key, 'C' for CANCEL key, "=" when maximum number of PIN number entered.
Length	Length of PIN block

Return: Section 5
Example: char *AccountNumber = "00004012345678909";
PIN_GetPINBlock('D', AccountNumber,17, EncryptedPIN, &length);

3.7.2 PIN_GetClearText

Function: PIN_GetClearText
Description: Get numeric entry in clear text form. (For clear text enabled units only)
Format: BYTE PIN_GetClearText(int Minlen, int Maxlen, char *ClearTextPIN, int *Length)
Parameter:

Minlen	0 < Minlen <= Maxlen <= 32(SmartPIN No_LCD) 0 < Minlen <= Maxlen <= 12(SecurePIN 100,SecurePIN130)
Maxlen	0 < Minlen <= Maxlen <= 32(SmartPIN No_LCD) 0 < Minlen <= Maxlen <= 12(SecurePIN 100,SecurePIN130)
ClearTextPIN	the Clear text(SecurePIN 100, SecurePIN130)
Length	the length of clear text(SecurePIN 100, SecurePIN130)

Return: Appendix A
Example: PIN_GetClearText(4, 12, ClearTextPIN, &length);

3.7.3 PIN_SetPINLength

Function: PIN_SetPINLength
Description: Set minimum and maximum number of digit for PIN number
Format: BYTE PIN_SetPINLength(int Minlen, int Maxlen)
Parameter: Minlen 4 to Maxlen
Maxlen Minlen to 12
Return: Section 5
Example: PIN_SetPINLength(4, 12)

3.7.4 PIN_SetLanguage

Function: PIN_SetLanguage
Description: Set language
Format: BYTE PIN_SetLanguage(char *Language)
Parameter: Language "English", "French", "German", "Italian", "Portuguese", "Spanish 1", "Spanish 2", "Others";
Return: Section 5
Example: PIN_SetLanguage("English");

3.7.5 PIN_SetManualParams

Function: PIN_SetManualParams
Description: Disable/enable manual parameter change
Format: BYTE PIN_SetManualParams(int Index, bool Flag)
Parameter: Index 0 – All parameter
1 – Audio
2 – Backlight
3 – Idle screen
Flag true – Enable
false – Disable
Return: Section 5
Example: PIN_SetManualParams(0, true);

3.7.6 PIN_CancelEnter

Function: PIN_CancelEnter
Description: Cancel get PIN block or key function
Format: BYTE PIN_CancelEnter ()
Parameter: None
Return: Section 5
Example: PIN_CancelEnter ()

3.8 Keypad functions

3.8.1 Key_GetFunct

Function: Key_GetFunct
Description: Get function key
Format: BYTE Key_GetFunct(char *Key, int *Length)
Parameter: Key F1 key – F1
 F2 Key – F2
 F3 Key – F3
 Cancel Key – CANCEL
 Enter Key – ENTER
 BackSpace key - BACKSPACE
 Length length of key
Return: Section 5
Example: Key_GetFunct(Key, &Length)

3.8.2 PIN_AddKeyHandle

Function: PIN_AddKeyHandle
Description: Register a call-back function for PIN entry
Format: BYTE PIN_AddKeyHandle(PKEY_FUNC func, LPVOID pParam)
Parameter: func The name of call-back function
 pParam The currently pointer
Return: Section 5
Example: PIN_AddKeyHandle(key_handle, this)

3.8.3 Text_AddKeyHandle

Function: Text_AddKeyHandle
Description: Register a call-back function for clear text entry. (For clear text enabled models only)
Format: BYTE Text_AddKeyHandle(PKEY_FUNC func, LPVOID pParam)
Parameter: func The name of call-back function
 clear text for numeric key;
 'B' for backspace key;
 "=" when maximum number of clear text number entered;
 'C' for Cancel key;
 if enter key is pressed: finish entering
 pParam The currently pointer
Return: Section 5
Example: Text_AddKeyHandle(text_handle, this)

3.9 MSR Functions

3.9.1 MSR_Restore_Default

Function: MSR_Restore_Default
Description: To return the reader to its default settings. Default settings as follow:
MSR reading enable,
Don't send Start/End sentinel, but send all data,
LRC character, without LRC in output,
'%' as Track 1 7 Bit Start Sentinel,
'%' as Track 1 6 Bit Start Sentinel,
';' as Track 1 5 Bit Start Sentinel,
'%' as Track 2 7 Bit Start Sentinel,
';' as Track 2 5 Bit Start Sentinel,
'%' as Track 3 7 Bit Start Sentinel,
'!' as Track 3 6 Bit Start Sentinel,
';' as Track 3 5 Bit Start Sentinel,
'?' as End Sentinel.
Format: BYTE MSR_Restore_Default()
Parameter: None
Return: Section 5
Example: MSR_Restore_Default()

3.9.2 MSR_Set_Reading

Function: MSR_Set_Reading
Description: turn the magnetic stripe reading function off or on
Format: BYTE MSR_Set_Reading(bool Flag)
Parameter: Flag 0 – MSR reading disable
 1 – MSR reading enable(default)
Return: Section 5
Example: MSR_Set_Reading(true)

3.9.3 MSR_Data_Status

Function: MSR_Data_Status
Description: Check the data status.
Format: BYTE MSR_Data_Status(int *Value)
Parameter: Value 0 – No data
 1 – New data since last read
 2 – Old data present
Return: Section 5
Example: MSR_Data_Status(Value)

3.9.4 MSR_Set_SendOption

Function: MSR_Set_SendOption
Description: Set up send mode for data.
Format: BYTE MSR_Set_SendOption(int Setting)
Parameter: Setting 0 – Do not send Start/End sentinel, but send all data.(default)
1 – Send Start/End sentinel and all data on all tracks
2 – Send raw data
Return: Section 5
Example: MSR_Set_SendOption(1)

3.9.5 MSR_Get_Setting

Function: MSR_Get_Setting
Description: Get the setting from PIN pad. It can read all setting for MSR, the setting sequence as follows:
MSR_read_flag,
send_option,
LRC_character;
track1_start5b_sentinel,
track1_start6b_sentinel,
track1_start7b_sentinel,
track2_start5b_sentinel,
track2_start7b_sentinel,
track3_start5b_sentinel,
track3_start6b_sentinel,
track3_start7b_sentinel,
track1_end_sentinel,
track2_end_sentinel,
track3_end_sentinel.
Format: BYTE MSR_Get_Setting(char *Setting, int *Length)
Parameter: Setting read all setting for MSR
Length The length of reading setting
Return: Section 5
Example: MSR_Get_Setting(Setting, &Length)

3.9.6 MSR_Set_StartSentinel

Function: MSR_Set_StartSentinel
Description: Set start sentinel for reading
Format: BYTE MSR_Set_StartSentinel(int Track, int Bit, char StartChar)
Parameter: Track 1 – track1
2 – track2
3 – track3
Bit 5, 6 or 7 (Track 2 can only be 5 or 7)
StartChar sentinel is a visual character
Return: Section 5
Example: MSR_Set_StartSentinel(1, 5, ';')

3.9.7 MSR_Set_EndSentinel

Function: MSR_Set_EndSentinel
Description: Set end sentinel for reading
Format: BYTE MSR_Set_EndSentinel (int Track, char EndChar)
Parameter: Track 1 – track1
2 – track2
3 – track3
EndChar sentinel is a visual character
Return: Section 5
Example: MSR_Set_EndSentinel(1, '?')

3.9.8 MSR_Read_Track

Function: MSR_Read_Track
Description: Read track data
Format: BYTE MSR_Read_Track(int Track, char *TrackData, int *Length)
Parameter: Track 0 – Any track(default)
 1 – track1 only
 2 – track2 only
 3 – track1 & track2
 4 – track3 only
 5 – track1 & track3
 6 – track2 & track3
 7 – all three tracks
 8 – track1 or track2
 9 – track1 or track3
 TrackData Read track data
 Length The length of read track data
Return: Section 5
Example: MSR_Read_Track(1, TrackData, &Length)

3.9.9 MSR_Set_LRC

Function: MSR_Set_LRC
Description: Enable/Disable LRC character in output format
Format: BYTE MSR_Set_LRC(bool Flag)
Parameter Flag 0 – without LRC in output (default)
 1 – with LRC in output
Return: Section 5
Example: MSR_Set_LRC(true)

3.9.10 PIN_Verify_Offline

Function: PIN_Verify_Offline
Description: Verify PIN Offline
Format: BYTE PIN_Verify_Offline(BYTE *Return_Data, int *Rlength)
Parameter Return_Data The return data of SecurePED.
 About the return data for further information,
 Reference the specific Smart Card information.
 Rlength The length of Return_Data
Return: Section 5
Example: PIN_Verify_Offline(Return_Data, &length)

3.10 Smart Card Operation

3.10.1 MiniSmart_Operation

Function: MiniSmart_Operation. This function supports communication with the MiniSmart reader. MiniSmart commands are sent in Block_Data; responses are received in Return_Data. See Section 6 for MiniSmart Reader command & related information.

Description: Communication support to the MiniSmart Reader.

Format: BYTE MiniSmart_Operation(BYTE *Block_Data, int Blength, BYTE *Return_Data, int *Rlength)

Parameter

Block_Data	The data block sent to MiniSmart, Includes "CommCode", "Parameters", & "Data"
Blength	The length of Block_Data
Return_Data	The return data of MiniSmart Reader
Rlength	The length of Return_Data

Return: Section 5

Example: MiniSmart_Operation(Block_Data, 1, Return_Data, &length)

4.0 Example for DLL call:

The following examples are based on Microsoft Visual C++
Pin.h, PinDll.lib, PinDll.dll must be put to relevant directory, and include head file #include "Pin.h"
add Lib(PinDll.lib): Add PinDll.lib to Project->Settings->Link->Object/library

Note: Some of the comments may differ from the comments found in the provided Files.

4.1 Single-thread Method

//Call DLL functions using single-thread method:

```
//Com_OpenPort
    BYTE res = Com_OpenPort(1,9600, 'E', 1, 7);
//Com_ClosePort
    bool Flag = Com_ClosePort();
//Com_SetBaud
    BYTE res = Com_SetBaud(9600);
//Com_SetParity
    BYTE res = Com_SetParity('E');
//Com_SetDataBit
    BYTE res = Com_SetDataBit(7);
//Com_SetStopBit
    BYTE res = Com_SetStopBit(1);
//Sys_GetVersion
    char Version[128];
    int length = 0;
    BYTE res = Sys_GetVersion(Version, &length);
//Sys_GetID
    char HardwareID[128];
    int length = 0;
    BYTE res = Sys_GetID(HardwareID, &length);
//Sys_GetModel
    char ModelNumber[128];
    int length = 0;
    BYTE res = Sys_GetModel(ModelNumber, &length);
//Sys_Reset
    BYTE res = Sys_Reset();
//Time_SetIdleTime
    BYTE res = Time_SetIdleTime(0);
//Sound_Control
    bool Flag = true;
    BYTE res = Sound_Control(Flag);
//Sound_Tone
```

User Manual, SecurePIN/ SmartPIN API Guide

```
        BYTE res = Sound_Tone(2000, 300);
//Display_LoadPrompt
        char *Message = "ID-TECH-2006";
        BYTE res = Display_LoadPrompt(0, Message);
//Display_Message
char *Message = "ID-TECH-2006";
        res = Display_Message(1, Message);
//Display_Clear
        BYTE res = Display_Clear(2);
//Display_BacklightSet
        bool Flag = true;
        BYTE res = Display_BacklightSet(Flag);
//Display_Prompt
        BYTE res = Display_Prompt(0, 0);
//Display_IdlePromptSelect
        BYTE res = Display_IdlePromptSelect(0, 0);
// PIN_CancelEnter
        BYTE res = PIN_CancelEnter ();
//PIN_SetPINLength
        BYTE res = PIN_SetPINLength(4, 12);
//PIN_SetLanguage
        BYTE res = PIN_SetLanguage("English");
//PIN_SetManualParams
        BYTE res = PIN_SetManualParams(0, true);
//GetDIIVersion
        char Version[128];
        int length = 0;
        BYTE res = GetDIIVersion(Version, &length);
// PIN_AddKeyHandle (Only when Non_LCD SmartPIN get PIN , it is necessary to use the
function)
        void __stdcall key_handle (char key, LPVOID pParam) //define a call-back function
                                                    which name would be registered when call
                                                    PIN_AddKeyHandle
        {
                CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
                pthis->m_EPINEntry += key;
                pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
        }
static UINT ThreadProc( LPVOID pParam )
{
        CString temp;
        CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
        char *AccountNumber = "00004012345678909";
        char EncryptedPIN[256];
        int length = 0;
```

User Manual, SecurePIN/ SmartPIN API Guide

```
BYTE res = 0;
pthis->m_EPINEntry = "";
res = PIN_GetPINBlock('D', AccountNumber, 17, EncryptedPIN, &length);
if(res == 1)
{
    for(int j = 0; j < length; j++)
    {
        pthis->m_EPINEntry += EncryptedPIN[j];
    }
    pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
    return 0;
}
if(res == 108)
    pthis->MessageBox("KEYS_NOT_LOADED");
return 0;
}
void CTestDIIDlg::OnPinentry()
{
    PIN_AddKeyHandle(key_handle, this); //register a call-back function which name is
                                        //key_handle
    AfxBeginThread(ThreadProc, this);
}
// Text_AddKeyHandle (Only when Non_LCD SmartPIN get clear text)
void __stdcall text_handle (char key, LPVOID pParam)
{
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    pthis->m_ClearText += key;
    TRACE("Get key:%c\n", key);
    pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
}
void CTestDIIDlg::OnGetcleartext()
{
    Text_AddKeyHandle(text_handle, this); //SmartPIN Non_LCD
    AfxBeginThread(ThreadProc_Text, this);
}
//MSR_Restore_Default
BYTE res = MSR_Restore_Default();
//MSR_Set_Reading
BYTE res = MSR_Set_Reading(true);
//MSR_Data_Status
int Value;
BYTE res = MSR_Data_Status(&Value);
//MSR_Set_SendOption
BYTE res = MSR_Set_SendOption(1);
//MSR_Get_Setting
```

User Manual, SecurePIN/ SmartPIN API Guide

```
int length = 0;
char Setting[256];
BYTE res = MSR_Get_Setting(Setting, &length);
//MSR_Set_StartSentinel
BYTE res = MSR_Set_StartSentinel(1, 5, '!');
//MSR_Set_EndSentinel
BYTE res = MSR_Set_EndSentinel(1, '?');
//MSR_Read_Track
char TrackData[512];
int length = 0;
BYTE res = MSR_Read_Track(0, TrackData, &length);
//MSR_Set_LRC
BYTE res = MSR_Set_LRC(true);
// MiniSmart_Operation
BYTE Block_Data[1] = {0x12};
BYTE Return_Data[512];
int length = 0;
BYTE res = MiniSmart_Operation (Block_Data, 1, Return_Data, &length);
```

4.2 Multi-threads Methods

// Call DLL functions using multi-threads methods:

```
//PIN_GetPINBlock (except Non_LCD SmartPIN)
static UINT ThreadProc( LPVOID pParam )
{
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    char *AccountNumber = "1234567812345678";
    char EncryptedPIN[256];
    int length = 0;
    BYTE res = PIN_GetPINBlock('D', AccountNumber,16, EncryptedPIN,
&length);
    return 0;
}
void CTestDIIDlg::OnPinentry()
{
    // TODO: Add your control notification handler code here
    AfxBeginThread(ThreadProc, this);
}
//PIN_GetPINBlock (Only Non_LCD SmartPIN)
void __stdcall key_handle (char key, LPVOID pParam)
{
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    pthis->m_EPINEntry += key;
    pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
}
static UINT ThreadProc( LPVOID pParam )
{
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    char *AccountNumber = "1234567812345678";
    char EncryptedPIN[256];
    int length = 0;
    BYTE res = PIN_GetPINBlock('D', AccountNumber,16, EncryptedPIN,
&length);
    return 0;
}
void CTestDIIDlg::OnPinentry()
{
    // TODO: Add your control notification handler code here
    PIN_AddKeyHandle(key_handle, this);
    AfxBeginThread(ThreadProc, this);
}
// PIN_GetClearText(Only Non_LCD SmartPIN and SecurePIN 100)
```

User Manual, SecurePIN/ SmartPIN API Guide

```
static UINT ThreadProc_Text( LPVOID pParam )
{
    CString temp;
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    BYTE res = 0;
    pthis->m_ClearText = "";
    char ClearTextPIN[256];
    int length = 0;
    res = PIN_GetClearText(4, 12, ClearTextPIN, &length);
    if(res == 1)
    {
        if(length > 0)
        {
            for(int j = 0; j < length; j++)
            {
                pthis->m_ClearText += ClearTextPIN[j];
            }
        }
        pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
        return 0;
    }
    return 0;
}

void CTestDIIDlg::OnGetcleartext()
{
    // TODO: Add your control notification handler code here
    Text_AddKeyHandle(text_handle,this); //SmartPIN Non_LCD
    AfxBeginThread(ThreadProc_Text, this);
}

//Key_GetFunct
static UINT ThreadProc_Key( LPVOID pParam )
{
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    char Key[256];
    int length = 0;
    BYTE res = Key_GetFunct(Key, &length);
    return 0;
}

void CTestDIIDlg::OnGetkey()
{
    // TODO: Add your control notification handler code here
    AfxBeginThread(ThreadProc_Key, this);
}

// PIN_Verify_Offline (only SecurePIN 150 and SecurePIN 180)
static UINT ThreadProc_Verify( LPVOID pParam )
```

User Manual, SecurePIN/ SmartPIN API Guide

```
{
    CString temp;
    CTestDIIDlg* pthis = (CTestDIIDlg*)pParam;
    BYTE Return_Data[512];
    int length = 0;
    BYTE res = PIN_Verify_Offline(Return_Data, &length);
    for(int i = 0; i < length; i++)
    {
        temp.Format("%02x",Return_Data[i]);
        pthis->m_Read += temp;
    }
    pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
    return 0;
}
void CTestDIIDlg::OnPinVerify()
{
    // TODO: Add your control notification handler code here
    AfxBeginThread(ThreadProc_Verify, this);
}
```

5.0 Return Values

Return Value	Description
0	FAIL
1	SUCCESS
50	NOT_SUPPORTED
99	PARAMETER_ERR
100	UNKNOWN_ERR
101	INVALID_COMMAND
102	COMMAND_ERR
103	TIME_OUT
104	UNIT_SUSPENDED
105	ACTION_CANCELED
106	ACTION_ABORTED
107	WRONG_KEY_TYPE
108	KEYS_NOT_LOADED
109	NO_PROMPT_LOADED
110	NO_HARDWARE_ID
111	NO_MSR_DATA
200	PORT_OPENED
201	PORT_CLOSED

6.0 Introduction to MiniSmart Reader Commands

This section provides the necessary information about the MiniSmart command set. The command set is driven by the Host system, passed through the SecurePIN communication channel, and is interpreted and acted upon by the MiniSmart reader as follows:

6.1 Conventions

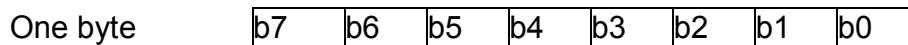
The following conventions are used in this document:

Numeric values:

- Numeric values are expressed in decimal notation unless otherwise noted.
- Binary numbers are followed by the 'b' character. For example, the decimal value 13 is expressed in binary as 1101b.
- Hexadecimal numbers are followed by the 'h' character. For example, the decimal value 13 is expressed in hexadecimal as 0Dh.
- The value 00h is assigned to any byte that is reserved for future use.

Bit Numbering:

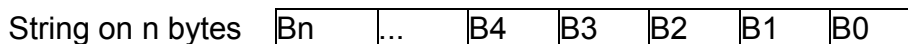
A byte consists of 8 bits, b7 to b0, where b7 is the most significant bit and b0 the least significant bit, as shown below:



Byte numbering:

A string of n bytes consists of n number of concatenated bytes:

Bn-1 ...B2, B1, where Bn is the most significant byte and B0 is the least significant byte:



6.2 Communication Protocols

All exchanges with the MiniSmart are handled by three communication layers:

- **The Command Layer**
- **The Transport Layer**
- **The Physical Layer**

The Command Layer handles and interprets the MiniSmart commands. The Transport Layer handles the message addressing, specifies the transmission type, and validates each transmission. The transport layer uses a Block Protocol, which is described later in this document. The Physical Layer handles data transmission.

6.3 The Command Layer

The command layer handles and interprets the MiniSmart commands. These commands are made up of the command code, data, and parameters.

Commands are sent in the following format:

| CommCode | Parameters | Data |

Where: **CommCode** is the command code.
 Parameters are the parameters sent with the command.
 Data is the data accompanying the command, where appropriate.

A Status Code is returned for every command received; sent in the following format:

| S | Data |

Where: **S** is the status code identifier.
 Data is the data returned with the status code, where appropriate.

Refer to Appendix A for a complete list of status codes meanings.

6.4 The Transport Layer

The transport layer handles message addressing, specifies the transmission type, and validates each transmission. It uses a Block Protocol.

The Block Protocol is a simplified version of the T=1 card protocol. Under this protocol, data is transmitted in blocks between the source and the destination.

There are three types of blocks:

I-Blocks (Information Blocks).	I-Blocks hold the data to be exchanged between the source and the destination.
R-Blocks (Receive Ready Blocks).	R-Blocks hold the positive or negative acknowledgements to transmissions.
S-Blocks (Send Completed Blocks).	S-Blocks indicate when the transmission is complete.

The data is exchanged in the following format:

NAD	PCB	LEN	DAT	EDC
------------	------------	------------	------------	------------

Where:

NAD is the source and destination identifier with the first nibble as the destination and the second nibble as the source.

PCB indicates the block type as described below:

I-Block PCB takes the following format:

0 S 0 X X X X X

The sequence bit **S** is set to 0 at power-up. The source sends the first I-Block transmitted with the sequence bit set to 0. It toggles the sequence bit every time it sends an information block. The MiniSmart and the Host system generate sequence bit values independently.

R-Block PCBs take the following format:

1 0 0 S 0 0 E V

Where: **V** is for "Error Being Verified by EDC"

E for "Another Error is Detected"

S for Sequence Number where the error is detected

User Manual, SecurePIN/ SmartPIN API Guide

S-Blocks request the destination to set the sequence bit to 0 and return a response to the source indicating that the transmission is complete.

There are two types S-Block PCBs:

Re-synch request: 1 1 0 0 0 0 0 0

Re-synch response: 1 1 1 0 0 0 0 0

LEN specifies on one byte the number of bytes in the DAT field.

DAT holds the data being transmitted.

EDC is the result of an exclusive OR performed on NAD, PCB, LEN, and DAT bytes.

6.5 The Physical Layer

The physical layer handles the data transmission itself. The Physical Layer uses the serial asynchronous protocol with which the data can be sent directly over the serial line. The bytes are sent over the line by an UART. The transmission speed is programmable. The default configuration is 9,600 bytes, eight bits, no parity, and one stop bit. The physical interface can be either at RS232 levels or at CMOS levels. The CMOS interface is a direct output from the controller IC's UART. The RS232 interface is obtained level shifting the CMOS signals using an RS232 converter IC.

Only the communication signals can be configured for RS232. See Flex Cable Pin-Out Assignments for information on specific signals that can be configured to RS232 levels. The configuration for either CMOS or RS232 is produced during the manufacturing process and cannot be change.

6.6 Microprocessor Card Operation

The MiniSmart handles ISO 7816 T=0 and T=1 protocol microprocessor cards. The following section describes the implementation of these standards.

Clock Signal

The MiniSmart chip transmits a clock signal to the card. The clock frequency is 3.6864 MHz. The operating mode is specified while selecting the card type and the Define Card Type command. Card type 02h should be selected for the standard mode.

Global Interface Parameters

These parameters are returned by the microprocessor card during the ATR. For more information on these parameters, refer to the ISO 7816-3 standard document.

TA1

The MiniSmart interprets this parameter to match its communication rate with that of the card, according to the clock rate conversion factor F. F is coded on the most significant nibble and the bit rate adjustment factor D is coded on the least significant nibble. If needed, the coding of F and D can be found in part 3 of the ISO 7816 standard.

The initial communication rate used during the ATR is 9 909.68 bps in standard mode.

After receiving the ATR, the MiniSmart selects the communication rate according to TA1.

Table 1 shows the clock rate conversion factors, the bit rate conversion factors, and the selected baud according to TA1 values.

TB1 and TB2

The Vpp option is not available. TB1 and TB2 parameters are ignored, Vpp default value is set to 5 V.

TC1

This parameter defines the extra guard time N, required by the card. This parameter is processed when sending characters to the card, to ensure a delay of at least (12+N) etu between two characters.

User Manual, SecurePIN/ SmartPIN API Guide

D	F	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048
1	TA1	01	11	21	31	41	51	61	91	A1	B1	C1	D1
	Bps	9909.68	9909.68	6606.45	4954.84	3303.23	2477.42	1981.94	7200.00	4800.00	3600.00	2400.00	1800.00
2	TA1	02	12	22	32	42	52	62	92	A2	B2	C2	D2
	Bps	19819.35	19819.35	13212.90	9909.68	6606.45	4954.84	3963.87	14400.00	9600.00	7200.00	4800.00	3600.00
4	TA1	03	13	23	33	43	53	63	93	A3	B3	C3	D3
	Bps	39638.71	39638.71	26425.81	19819.35	13212.90	9909.68	7927.74	28800.00	19200.00	14400.00	9600.00	7200.00
8	TA1	04	14	24	34	44	54	64	94	A4	B4	C4	D4
	Bps	79277.42	79277.42	52662.86	39638.71	26425.81	19819.35	15855.48	57600.00	38400.00	28800.00	19200.00	14400.00
12	TA1	08	18	28	38	48	58	68	98	A8	B8	C8	D8
	Bps	-	-	79277.42	59458.06	39638.71	29729.03	23783.23	86738.82	57600.00	43369.41	28800.00	21621.11
16	TA1	05	15	25	35	45	55	65	95	A5	B5	C5	D5
	Bps	-	-	105325.71	79277.42	52662.86	39638.71	31779.31	115200.00	76800.00	57600.00	38400.00	28800.00
20	TA1	09	19	29	39	49	59	69	99	A9	B9	C9	D9
	Bps	-	-	-	99632.43	65828.57	49481.88	39638.71	-	95750.65	72282.35	47875.32	35964.88
32	TA1	06	16	26	36	46	56	66	96	A6	B6	C6	D6
	Bps	-	-	-	-	105325.71	79277.42	63558.62	-	-	115200.00	76800.00	57600.00

Table 2 - TA1 Values (Frequency: 3.6864 MHz)

Note: For a communication rate higher than 115200bps, extra card guard time is required.

6.7 Communication Protocols

The least significant nibble of the TDI parameter in the ATR defines the protocol to be used by the reader (T=0 or T=1), according to the following table:

Value	Protocol
0	T=0
1	T=1

If the reader does not receive a TDI value, it defaults to the T=0 protocol.

6.8 T=0 Protocol

The specific TC2 interface parameter is interpreted to set the value of the work waiting time, W. If this parameter is absent, a maximum of 960xD etu elapses before timing-out on a character sent by the card. Otherwise a maximum of 960xDxW etu elapses before timing-out.

To send instructions to a T=0 microprocessor card, the ISO Input and ISO Output or the exchange APDU commands are used.

6.9 T=1 Protocol

To send instructions to a T=1 microprocessor card, the Exchange APDU command is used. The T=1 specific interface bytes are interpreted as per clause 9 of the ISO 7816-3 standard. These bytes are TA3, TB3, TC3.

TA3 codes the Information Field Size of the Card (IFSC). The default value is 32 bytes.

TB3 codes the BWI (Block Writing Time Integer) and the CWI (Character Waiting Time Integer).

TC3 defines the Error Detection Code (EDC) type.

6.10 Command Set

This section describes the MiniSmart command set. For each command, it indicates the following:

- **The function performed**
- **The syntax**
- **The data returned**

6.11 Configuration Commands

Configuration commands are used to define reader settings. They are:

- **Configure SIO Line**
- **Set Delay**
- **Read Firmware Version**
- **Restart**
- **Set Reader In Halt Mode**
- **Toggle Plug And Play capability**

User Manual, SecurePIN/ SmartPIN API Guide

Each command is described in the following pages.
See Appendix A for a description of status codes.

Configure SIO Line

This command sets the Serial Input Output (SIO) line baud rate. After a power up, the line defaults to no parity, eight bits per character, 9,600 bps and one stop bit.

Note: The line is reconfigured as soon as this command is executed. The response is returned with the new parameters.

Format:
0Ah CB

Where: CB is the configuration byte. The configuration flag settings are defined in the following table:

Bit	Value	Description
7 to 3		Not used
2 to 0	XXX	000: 115,200 bps 001: 76,800 bps 010: 38,400 bps 011: 19,200 bps 100: 9,600 bps 101: 4,800 bps 110: 2,400 bps 111: 1,200 bps

Response:
S = 00h

Set Delay

When a slow Host computer is used, this command delays the MiniSmart's responses.

Format:
23h 01h 00h 4Ch 01h Delay

Where: Delay is the response delay in milliseconds. Enter a value between 0 & 255.
Upon power up, the delay time defaults to 0.

Response:
S

User Manual, SecurePIN/ SmartPIN API Guide

Read Firmware Version

This command returns the version of the MiniSmart firmware.

Format:

22h 05h 3Fh E0h 10h

Response:

S Version

Where: Version is the installed software version in ASCII.

Example:

"GemCore-V1 .4x-Gy"

Where:

x = minor release number

y = either...

H for a ROM masked chip or...

I for a RAM/EEPROM chip

OROS-Compatible Command:

Format:

22h 05h 3Fh F0h 10h

Response:

S <OROS-R2.99-R1.4x>

Restart

This command is used to reset the MiniSmart firmware. All the parameters are initialized with the default values.

Format:

0Ch 00h 00h 00h

Response:

None.

Note: The reader is ready to receive the next command after 260 milliseconds.

User Manual, SecurePIN/ SmartPIN API Guide

Set Reader in Halt Mode

This command switches the reader to halt mode. When switching to this mode, the card is powered down and the reader goes into low power consumption mode.

Format:
0Eh 00h 00h 00h

Response:
S = 00h

Note: The reader goes into halt mode approximately 2 milliseconds after sending the response.

Note: Also see the Power Control command: Reader Power Down

Note: The reader awakens upon receipt of any command from the Host. This command is ignored by the reader. The reader also awakens after a smart card insertion or withdrawal. When the reader awakens, it performs a reset sequence and stays in the default state. The reader is ready to receive the next command after 260 milliseconds.

Toggle PNP Capability

This command disables the Plug And Play enumeration detection if it was enabled. It enables this capability if it was disabled

Format:
5Ah

Response:
SPNP

Where: PNP is the current stat of Plug And Play enumeration detection capability:
00h: Detection is disabled
01h: Detection is enabled

6.12 Power Control Commands Set

Power Control Commands are:

- **Set Timeout**
- **Refresh**
- **Reader Power Down**
- **Status**

See Appendix A for a description of the status codes.

Set Timeout

This command modifies the silent timeout value. This timeout is reset upon receipt of a command. If this timeout ends, the reader is set to Halt mode, as with the Set Reader In Halt Mode command or the Reader Power Down command. The default value for timeout is infinite.

Format:
52hT

Where: T is the silent timeout value in seconds. Value 00h sets an infinite timeout (The timeout never ends).

Response:
S

Refresh

This command is not an operational command. Its sole purpose consists of resetting the silent timeout.

Format:
53h

Response:
S

Reader Power Down

This command sets the reader to Halt mode. It has the same effect as the Set Reader In Halt Mode command.

Format:
54h

Response:
S=00h

Note: The reader goes into Halt mode in less than 2 milliseconds after sending the command.

User Manual, SecurePIN/ SmartPIN API Guide

Note: Also see the Set Reader in Halt Mode command.

Note: The reader wakes up upon receipt of any command from the Host. This command is ignored by the reader. The reader also wakes up after a smart card insertion or withdrawal. When the reader wakes up, it performs a reset sequence and stays in the default state. The reader is ready to receive the next command after around 260 milliseconds.

Status

This command returns the current silent timeout value.

Format:

56h

Response:

S 0h T

Where: T is the current silent timeout value.

6.13 Card Interface Command Set

The card interface commands manage the communication with smart cards.

The behavior of certain commands changes depending on the selected card type. Therefore, some commands are common to all types while others are redefined or disabled according to the card type.

Four groups of commands are defined:

- **Common card interface commands**
- **Specific commands for asynchronous cards, generic ISO operating mode**
- **Specific commands for asynchronous cards, EMV-compliant operating mode**
- **Specific command for synchronous cards**

Common Card Interface Commands

These commands are valid regardless of the selected type. Common card interface commands are:

- **Power Down**
- **Define Card Type and Card Presence Detection**
- **Directory**
- **Set Operating Mode**

Each command is described in the following pages.
See Appendix A for a description of status codes.

Power Down

This command powers down the card. The MiniSmart also powers down the card as soon as it detects a withdrawal.

Format:
11h

Response:
S

The Power Down command always ends normally if a card is present in the reader (S=00h). If no card is inserted, the command returns the FBh "card missing" error.

User Manual, SecurePIN/ SmartPIN API Guide

Define Card Type and Card Presence Detection

The MiniSmart does not have a smart card recognition algorithm. It therefore is necessary to define the type of card used. This command sets the card type.

Note: When the MiniSmart is reset or powered up, the card type defaults to standard microprocessor card mode (type 2).

Format:
17h T [00h [P]]

Where: T is the card type selection byte. The card type codes are as follows:

Card Type	Description
01h	Synchronous smart card, contact ID TECH for more information on memory cards if the application is required.
02h	Standard speed mode (clock frequency = 3.6864 MHz) ISO 7816-3 T=0 and T=1 microprocessor cards.
0EFh	Standard speed mode microprocessor card. Transparent protocol, the protocol management is not handled by the reader.

If the command is sent with a family number which does not match one of the current card types, the current card is powered down:

P is the card presence byte. This optional parameter is used to modify the card presence indication options. When this parameter is not specified, the card presence is not indicated. See the following table for further information.

After the power up, the card presence is not indicated CSEAT (7) signal on the interface flex-cable is set to its default state (1).

Card Presence Byte	Description
00000XX0	CSEAT = 1 means a card is present. If the CSEAT = 0, no card is present.
00000XX1	CSEAT = 0 means a card is present. If the CSEAT = 1, no card is present.
00001XX0	On CSEAT, a (5 msec) positive pulse occurs upon card insertion/withdrawal
00001XX1	On CSEAT, a (5 msec) negative pulse occurs upon card insertion/withdrawal

Response:
S

User Manual, SecurePIN/ SmartPIN API Guide

Note: When it switches from microprocessor to synchronous smart card and vice versa, the MiniSmart automatically powers down the card. When EMV operating mode is selected, it is not possible to select the transparent protocol type. The MiniSmart rejects such a command.

Directory

This command is used to obtain the types of cards handled, the release number and the characteristics for each card driver.

Format:
17h 00h

Response:
S <TYPE, CMD, REV> ... <TYPE, CMD, REV>

Where:

Type	Card Type (for example: 02h = Asynchronous Card)
CMD	00h: ISO IN/OUT 01h: APDU 02h: ISO IN/OUT and APDU
REV	Card driver release (2 bytes)

Set Operating Mode

This command selects the operating mode of treatment of an asynchronous card. Two modes exist:

- **Generic ISO mode**
- **EMV-compliant mode**

Some commands are not allowed in EMV mode, while others undergo changes in their behavior.

Format:
17h 00h Mode

Where: Mode is the operating mode to be selected
47h selects the generic ISO mode
45h selects the EMV-compliant mode
00h requests the mode currently selected

Response:
SMode

Where: Mode is the mode currently selected
47h = generic ISO mode
45h = EMV-compliant mode

Note: The default operating mode is ISO.

6.14 Specific Commands for Asynchronous Cards

These commands, which are used with a card selected as asynchronous (type = 02h), have a specific behavior.

Commands that are valid with these types are:

- Power Up - Asynchronous Cards
- Change Card Communication Parameters - Asynchronous Cards
- ISO Output - Asynchronous Cards
- ISO Input - Asynchronous Cards
- Exchange APDU - Asynchronous Cards
- Card Status - Asynchronous Cards

See Appendix A for a description of status codes.

Power Up – Asynchronous Cards

This command powers up and resets a card.

Format:

12h [CFG][PPS0,PPS1,PPS2,PPS3][PCK]

If the CFG parameter is not specified, the card is powered with 5 V, there is no PTS management and the operating mode is compatible with OROS2.2X.

User Manual, SecurePIN/ SmartPIN API Guide

If the CFG parameter is specified:

CFG	Description
X0XXX001	Class A: Vcc for Card is 5V
X0XXX010	Class B: Vcc for Card is 3V
X0XXX100	Class C: Vcc for Card is 1.8V
X0XXX011	Class AB: Vcc for Card is 5V or 3V
X0XXX110	Class BC: Vcc for Card is 3V or 1.8V
X0XXX111	Class ABC: Vcc for Card is 5V, 3V or 1.8V
0000XXXX	Operation is compatible with OROS2.2X
0001XXXX	Reset and no PPS management. The reader stays at 9,600 bps if the card is in negotiable mode.
0010XXXX	Reset and automatic PPS management. The reader uses the highest speed proposed by the card. Change to T=1 protocol if there is a choice between T=0 and T=1.
1111XXXX	Manual PPS management. This command does not reset the card. It must be preceded by a Power Up command with the CFG parameter set to 0001XXXX. The parameters from PPS0 to PCK are sent to the card at 9,600 bps. If PCK is omitted, it is computed and added by the MiniSmart. If the card responds with PPS RESPONSE, the reader is configured using the parameters returned.
00001000	Valid only if T=1 is the current protocol, otherwise no action occurs. An S-IFS block exchange is initiated by the MiniSmart. The IFSD (maximum length of INF field accepted by the MiniSmart) sent to the card is the value of parameter PPS0. No other parameters are allowed.
X0XX1XXX or 11111XXX	If the selected protocol after the ATR or the PPS exchange is T=1, the MiniSmart initiates an S-IFS block exchange. The IFSD value indicated to the card is FEh. After a command reset with no PPS and with IFSD exchange, a command of manual PPS management is invalid.

Response:

S <card response>

Where: <card response> is the card Answer To Reset (ATR).

User Manual, SecurePIN/ SmartPIN API Guide

Change Card Communication Parameters – Asynchronous Cards

This command dynamically changes the parameters of the communication with the card. This command is mainly used to switch the speed or the protocol when the card uses a proprietary mode to switch these parameters.

Format:

12h PRT CNF1 CNF2 CNF3 CNF4

Where: PRT Selects the protocol. The format of this byte is:

0	1	0	C	P	E	S	R
---	---	---	---	---	---	---	---

C = 0; bits P E S R are significant.
1; bits P E S R are not taken into account.

P selects the protocol to be used.
When P = 0, the protocol is T=0.
When P = 1, the protocol is T=1.

E selects the computing mode for EDC. It is significant only if T=1 is selected.
When E = 0, EDC is LRC.
When E=1, EDC is CRC.

S initializes the sequence number of the last I-block sent.
When S = 0, the next I-block will be sent with sequence number 1.
When S = 1, the next I-block will be sent with sequence number 0.

R is the sequence number for the next I-block to be received. When R = 0, the next I block is expected to have sequence number 0. When R = 1, the next I block is expected to have sequence number 1.

CNF1 Selects the new TA1 (FI/DI=speed) to be used.
CNF2 Selects the new TCI (N=extra guard time) to be used.
CNF3 If protocol T=0 is selected, this indicates the new TC2 (WI= waiting time) to be used. If protocol T=1 is selected, this indicates the new TA3 (TFSC= maximum length of information field of blocks which can be received by the card) to be used.
CNF4 If protocol T=0 is selected, reserved for future use. If protocol T=1 is selected, indicates the new TB3 (BWI/CWI= block and character waiting time) to be used.

Response:

S

Note: No check is performed on parameters PRT, CNF1, CNF2, CNF3, and CNF4.

User Manual, SecurePIN/ SmartPIN API Guide

ISO Output – Asynchronous Card

This command sends ISO Out commands, that is, commands which retrieve data from the asynchronous card.

This command can return up to 252 data bytes in one operation. Consequently, two operations are required to obtain the minimum of 256 data bytes.

Format:

13h CLA INS AI A2 LN

Where: CLA, INS, AI, A2 and LN are the five ISO header bytes. For more details about ISO header contents, refer to the documentation concerning the card being used. The ISO header is transmitted directly to asynchronous cards.

Response:

S <data> SW1 SW2

Where: <data> Is the data returned by the card.

If a smart card error or a MiniSmart error is detected (S<>00h and S<>E7h), the MiniSmart does not return any valid data. The card may return any number of bytes up to LN.

If the number of data bytes to be returned is greater than 252, the first 253 bytes are contained in the <data> field. The SW1 and SW2 bytes follow the second response, not the first one. In order to obtain the rest of the response, the following command must be sent:

SW1 SW2Status Word 1, Status Word 2

Format:

13h FFh FFh FFh FFh FFh

Note: The description of SW1 and SW2 is given in the ISO 7816 standard.

Response:

S <data> SW1 SW2

Where: <data> Is the remainder of the response (LN-253 bytes).
SW1 SW2 Status Word 1, Status Word 2

Note: The MiniSmart returns error code 1Bh if a card interface command other than the above is sent.

User Manual, SecurePIN/ SmartPIN API Guide

ISO Input – Asynchronous Card

This command sends ISO In commands, that is, commands which send data to an asynchronous card. This command can send up to 248 data bytes in one operation. Consequently, two operations are required to send 255 data bytes.

Format:

14h CLA INS AI A2 LN <data>

Where:

CLA, INS, AI, A2 and LN

Are the five ISO header bytes. For more details about the ISO header contents, refer to the documentation concerning the card being used. The ISO header is transmitted directly to microprocessor cards (asynchronous cards).

<data>

Represents the LN data bytes transmitted to the card after the ISO header. The maximum length of the data is 248 bytes.

Response:

S SW1 SW2

The SW1 and SW2 bytes hold the standard status codes returned by the card. Their respective values are 90h and 00h if the operation is successful.

If the number of data bytes to be transmitted is greater than 248, the command below containing the last data bytes must be sent before the 'normal' ISO Input command containing the first 248 data bytes.

Format:

14h FFh FFh FFh FFh (LN-248) <data249.dataLN>

Where:

<data 249.dataLN> Are bytes number 249 through LN.

Response:

S SW1 SW2

The SW1 and SW2 bytes hold the standard status codes returned by the card. Their respective values are 90h and 00h if the operation is successful.

User Manual, SecurePIN/ SmartPIN API Guide

Exchange APDU – Asynchronous Cards

This command sends a command Application Data Protocol Unit (APDU) to the card, and retrieves the response APDU. Only the short APDU format is supported.

Format:

15h APDU

Where: APDU is the command APDU.

If the T=1 protocol is selected and the APDU command length is greater than the card information field size, it is truncated and sent to the card in several chained blocks. If the T=0 protocol is selected, the APDU transportation in T=0 TPDU (Transport Protocol Data Unit) is handled by the MiniSmart. Please refer to the documentation concerning the card currently used for APDU command details. Up to three operations are required to perform a maximum length ISO short APDU exchange (261 bytes for APDU and 258 bytes for APDU responses).

Response:

S Response APDU

Where: Response APDU is the response APDU to the command.

If the T=1 protocol is selected and the card replies in chained blocks, they are concatenated. If the T=0 protocol is selected, the T=0 TPDU of the response is mapped in the APDU response format by the MiniSmart. Refer to the documentation concerning the card currently used for APDU response details. If the command APDU length (LA) exceeds 254 bytes, the command below containing the last part of the command APDU must be sent before the "normal" APDU exchange command containing the first 254 bytes.

Format:

15h FFh FFh FFh FFh (LA-254) <apdu255.apduLA>

If the response APDU length (Lr) exceeds 254 bytes, the first 254 bytes of the response are returned with the status code 1Bh indicating that the command below must be sent to retrieve the last bytes of the response.

Where: <apdu255.apduLA> Are bytes number 255 through LA.

Format:

15h FFh FFh FFh FFh XX

Where: XX Can be any dummy byte value

User Manual, SecurePIN/ SmartPIN API Guide

APDU Format:

The APDU format is defined by the ISO 7816-4 standard.

APDUs can belong to one of several cases, depending on the length and contents of the APDU. MiniSmart handles the following cases:

Case 1 No command or response data.

Case 2 Short format: no command data, response data between 1 and 256 bytes.

Case 3 Short format: command data between 1 and 255 bytes and no response data.

Case 4 Short format: command data between 1 and 255 bytes, response data between 1 and 256 bytes. These cases are referred to as 1, 2S, 3S and 4S respectively.

Command Format:

Commands are sent in the following format:

Header	Body		
CLA INS P1 P2	Lc	Parameters/data	Le

The fields are described below.

Header Fields:

Header fields are mandatory. They are as follows:

Field Name	Length	Description
CLA	1	Instruction class
INS	1	Instruction code. This is given in the command descriptions.
P1	1	Parameter 1
P2	1	Parameter 2

Body Fields:

The command body is optional. It includes the following fields:

Field Name	Length	Description
Lc	1	Length of the data field
Data	Lc	Command parameters or data
Le	1	Expected length of data to be returned

For full details about the header and body field contents, refer to the documentation concerning the card currently used.

User Manual, SecurePIN/ SmartPIN API Guide

Response Format:

Responses to commands are received in the following format.

Body	Trailer
Data	SW1, SW2

The body is optional and holds any data returned by the card.

The trailer includes the following two mandatory bytes:

SW1: Status byte 1 which returns the command processing status

SW2: Status byte 2 which returns the command processing qualification

For full details about the response field contents refer to the documentation concerning the card currently used.

T=1 IFSC/IFSD.

If the T=1 protocol is used, when block chaining occurs, the buffers' length is determined by IFSC and IFSD parameters.

The default values for the MiniSmart buffer (IFSD) and the card buffer (IFSC) are 32 bytes.

The smart card can indicate a specific value of IFSC during the ATR. MiniSmart takes into account this new value instead of the default one.

To specify an IFSD value other than the default one to the card, see the Power Up -Asynchronous Card command.

User Manual, SecurePIN/ SmartPIN API Guide

Card Status – Asynchronous Card

This command is used to obtain the status of the card interface. It returns information indicating:

- The type of card currently used
- Card presence
- The power supply value
- The card power status
- The communication protocol (T=0 or T=1)
- The speed parameters between the card and the reader

Format:

17h

Response:

S STAT TYPE CNF1 CNF2 CNF3 CNF4

Where:

STAT	0000X000	Card not inserted
	0000X100	Card inserted but not powered
	0000X101	Card inserted, power supply = 1.8V
	0000X110	Card inserted, power supply = 5V
	0000X111	Card inserted, power supply = 3V
	00000XXX	T=0 protocol
	00001XXX	T=1 protocol
TYPE	Activated Card type	
CNF1 CNF2 CNF3 CNF4	CNF1=TA1 (FI/DI) CNF2=TC1 (EGT) CNF3=W1 CNF4=00h	T=0 Card as per ISO 7816-3
	CNF1=TA1 (FI/DI) CNF2=TC1 (EGT) CNF3=IFSC CNF4=TB3 (BWI/CWI)	T=1 Card as per ISO 7816/3

User Manual, SecurePIN/ SmartPIN API Guide

Asynchronous Cards - Generic Operating Mode - Transparent Protocol

These commands are designed for use with an asynchronous card in transparent protocol (type=EFh). They have a specific behavior.

Note: It is not possible to select the transparent protocol in EMV mode.

Valid commands for the transparent protocol are:

- Change Transparent Protocol Parameters
- Power Up - Transparent Protocol
- Exchange Block - Transparent Protocol
- Card Status - Transparent Protocol

Change Transparent Protocol Parameters

This command is used to set the working parameters of the transparent protocol.

Format:

12h CFG ETU EGT CWT BWT

Where:

CFG Specifies the card characteristics and selects the operating mode.

CFG	Description
XXXXXX00 or XXXXXX10	Vcc for the card is 5V.
XXXXXX01	Vcc for the card is 3V.
XXXXXX11	Vcc for the card is 1.8V.
XXXX0XXX	The format of the blocks received is not defined: the end of a received block is determined by the CWT timeout.
XXXX1XXX	The format of the blocks received is comparable to that of the T=1 protocol. The third byte of the block indicates the length of the data to be received before the EDC field.
XX0XXXXX	The direct convention is used to transfer byte.
XX1XXXXX	The inverse convention is used to transfer byte.
X0XXXXXX	During the ATR, a check is performed for the T0 and TDi characters to compute the number of characters to be received.
X1XXXXXX	No check of computation is performed during the ATR. The ATR is complete upon CWT timeout.
0XXXXXXX	Significant only if bit 3 of the CFG is set. EDC is one byte long.
1XXXXXXX	Significant only if bit 3 of the CFG is set. EDC is two bytes long.

ETU is the etu duration, coded in clock period number minus one and divided by three: $[(FI/DI)-1]/3$.

User Manual, SecurePIN/ SmartPIN API Guide

Samples for Common TA1:

TA1	FI/DI	ETU Parameter
0x11	372	123 = 7Bh
0x12	186	61 = 3Dh
0x13	96	30 = 1 Eh
0x14	41.5	15 = 0Fh
0x18	31	10 = 0Ah
0x58	124	41 = 29h
0x95	32	10 = 0Ah

EGT Defines the extra guard time etus between characters sent by the MiniSmart. The total duration of this character is $(12+EGT) \times (\text{etu duration})$. The value 255 for EGT has a special meaning; the character duration is 11 etus.

CWT Defines the maximum waiting time between the leading edges of two consecutive characters in the same direction.

Timeout duration is $(11+2^{\wedge}CWT) \times (\text{etu duration})$.

The maximum value for CWT is 15.

BWT Defines the maximum waiting time between the leading edges of two consecutive characters sent in opposite directions.

Timeout duration is $[(11+960 \times 2^{\wedge}BWT)] \times [\text{etu duration}]$.

Response:

S

Default parameter values:

CFG	=	04h	
ETU	=	7Bh	
EGT	=	02h	
CWT	=	0Dh	
BWT	=	04h	Power Up - Transparent Protocol

This command powers up and resets an asynchronous card in transparent mode.

Format:

12h

Response:

S <card response> <card response> is the card Answer To Reset (ATR).

Note: No verification is performed on characters returned by the card, in particular with respect to TS and TCK.

User Manual, SecurePIN/ SmartPIN API Guide

Exchange Block - Transparent Protocol

This command sends a block to a card and receives a block back in response.

Format:
15h BLOCK

Where: BLOCK Is the block to be sent.

Response:
S Response BLOCK

Where: Response BLOCK Is the block received in response.
Up to three operations are required in order to perform an exchange of blocks of maximum length (259 bytes). If the length (LB) of the block to be sent to the card exceeds 254 bytes, the command below containing the last part of the block must be sent before the "normal" exchange block command containing the first 254 bytes.

14h<block255.blockLB>

Where: <block255.blockLB> Are bytes 255 through LB

Note: If the length of the block received in response (LR) exceeds 254 bytes, the first 254 bytes are returned with the status code 1Bh, indicating that the command below must be sent to retrieve the last bytes of the response.

Format:
13h.

Note: If no block is given in the command, the MiniSmart waits for the response block.

User Manual, SecurePIN/ SmartPIN API Guide

Card Status - Transparent Protocol

This command is used to obtain the current transparent protocol parameters. It returns information regarding:

- The transparent protocol selected
- The card presence
- The power supply value
- The card power status
- The speed and timeout parameters

Format:
17h

Response:

S STAT TYPE ETU EGT CWT BWT Where:

STAT	0000X000	Card not inserted
	0000X100	Card inserted but not powered
	0000X101	Card inserted, power supply = 1.8V
	0000X110	Card inserted, power supply = 5V
	0000X111	Card inserted, power supply = 3V
	00000XXX	T=0 protocol
	00001XXX	T=1 protocol
TYPE		Activated card type EFh = Transparent protocol
ETU		etu duration
EGT		Extra guard time requested
CWT		Character waiting time
BWT		Response block waiting time

User Manual, SecurePIN/ SmartPIN API Guide

Synchronous Cards – EMV Compliant Operating Mode

These commands behave specifically when the EMV-compliant mode is selected.

- Power Up - EMV-compliant
- Exchange APDU - EMV-compliant
- Card Status - EMV-compliant

When in EMV-compliant operating mode, MiniSmart rejects the following commands:

- Change Card Communication Parameters - Asynchronous cards
- ISO Output - Asynchronous cards

ISO Input - Asynchronous cards

See "Appendix A – Status Codes" for a description of status codes.

Power Up – EMV Compliant

This command powers up and resets the card.

The card response is transmitted using the EMV criteria, and the card behavior is EMV-compliant.

The card can be:

Accepted

- Accepted after a warm reset

Rejected

If protocol T=1 is selected, an automatic IFSD exchange is performed.

Format:

12h

Response:

S <card response>

Where: <card response> is the card Answer To Reset (ATR)

User Manual, SecurePIN/ SmartPIN API Guide

Exchange APDU – EMV Compliant

Sends a command Application Data Protocol Unit (APDU) to a card, and retrieves the response APDU. This command's behavior obeys EMV requirements. For example, deactivation on timeout.

Format:
15h APDU

Where: APDU is the command APDU.

If the T=1 protocol is selected and the APDU command length is greater than the card information field size, it is truncated and sent to the card in several chained blocks. If the T=0 protocol is selected, the APDU transportation in T=0 TPDU (Transport Protocol Data Unit) is handled by the MiniSmart. Please refer to the documentation concerning the card currently used for APDU command details.

Up to two commands are required to perform a maximum length ISO short APDU exchange (261 bytes for APDU and 258 bytes for APDU responses).

Response:
S Response APDU

Where: Response APDU is the response APDU to the command.

If the T=1 protocol is selected and the card replies in chained blocks, they are concatenated. If the T=0 protocol is selected, the T=0 TPDU of the response is mapped in the APDU response format by the MiniSmart. Refer to the documentation concerning the card currently used for APDU response details. If the command APDU length (LA) exceeds 254 bytes, the command below containing the last part of the command APDU must be sent before the "normal" APDU exchange command containing the first 254 bytes.

Format:
15h FFh FFh FFh FFh (LA-254) <apdu255.apduLA>

Where: <apdu255.apduLA> are bytes 255 through LA.

If the response APDU length (Lr) exceeds 254 bytes, the first 254 bytes of the response are returned with the status code 1Bh indicating that the command below must be sent to retrieve the last bytes of the response.

Format:
15h FFh FFh FFh FFh XX

Where: XX Can be any dummy byte value.

User Manual, SecurePIN/ SmartPIN API Guide

APDU Format:

The APDU format is defined by the ISO 7816-4 standard.

APDUs can belong to one of several cases, depending on the length and contents of the APDU.

MiniSmart handles the following cases:

Case 1 No command or response data.

Case 2 Short format: no command data, response data between 1 and 256 bytes.

Case 3 Short format: command data between 1 and 255 bytes and no response data.

Case 4 Short format: command data between 1 and 255 bytes, response data between 1 & 256 bytes.

These cases are referred to as 1, 2S, 3S and 4S respectively.

Command Format:

Commands are sent in the following format:

Header	Body		
CLA INS P1 P2	Lc	Parameters/data	Le

The fields are described below.

Header Fields:

Header fields are mandatory. They are as follows:

Field Name	Length	Description
CLA	1	Instruction class
INS	1	Instruction code. This is given in the command descriptions.
P1	1	Parameter 1
P2	1	Parameter 2

Body Fields:

The command body is optional. It includes the following fields:

Field Name	Length	Description
Lc	1	Length of the data field
Data	Lc	Command parameters or data
Le	1	Expected length of data to be returned

For full details about the header and body field contents, refer to the documentation concerning the card currently used.

User Manual, SecurePIN/ SmartPIN API Guide

Response Format:

Responses to commands are received in the following format.

Body	Trailer
Data	SW1, SW2

The body is optional and holds any data returned by the card.

The trailer includes the following two mandatory bytes:

SW1: Status byte 1 which returns the command processing status

SW2: Status byte 2 which returns the command processing qualification

For full details about the response field contents refer to the documentation concerning the card currently used.

T=1 IFSC/IFSD.

If the T=1 protocol is used, when block chaining occurs, the buffers' length is determined by IFSC and IFSD parameters.

The default values for the MiniSmart buffer (IFSD) and the card buffer (IFSC) are 32 bytes.

The smart card can indicate a specific value of IFSC during the ATR. MiniSmart takes into account this new value instead of the default one.

User Manual, SecurePIN/ SmartPIN API Guide

Card Status – EMV Compliant

This command is used to obtain the status of the main card interface or of the auxiliary card. It returns information indicating:

- The type of card currently used
- Card presence
- The power supply value
- The card power status
- The communication protocol (T=0 or T= 1)
- The speed parameters between the card and the reader

Format:
17h

Response:
S STAT TYPE CNF1 CNF2 CNF3 CNF4

Where:

STAT	0000X000	Card not inserted
	0000X100	Card inserted, but not powered
	0000X101	Card inserted, power supply = 1.8V
	0000X110	Card inserted, power supply = 5V
	0000X111	Card inserted, power supply = 3V
	00000XXX	T=0 protocol
	00001XXX	T=1 protocol
TYPE	Activated Card type	
CNF1 CNF2 CNF3 CNF4	CNF1=TA1 (FI/DI) CNF2=TC1 (EGT) CNF3=WI CNF4=00	T=0 Card as per ISO 7816/3
	CNF1=TA1 (FI/DI) CNF2=TC1 (EGT) CNF3=IFSC CNF4=TB3 (BWI/CWI)	T=1 Card as per ISO 7816/3

User Manual, SecurePIN/ SmartPIN API Guide

6.15 Appendix A – Status Codes

The status codes returned by the cards are listed in the following table:

Code	Description
------	-------------

00h	Successful command.
01h	Unknown driver or command.
02h	Operation impossible with this driver.
03h	Incorrect number of arguments.
04h	Reader command unknown. The first byte of the command is an invalid command code.
05h	Response exceeds buffer capacity.
10h	Wrong response upon card reset. The first byte of the response (TS) is invalid.
12h	Message is too long. The buffer is limited to 254 bytes, 248 of which are for data exchanged with the card.
13h	Byte reading error returned by an asynchronous card.
15h	Card powered down. A Power Up command must be sent to the card before any other operation.
1 Bh	A command with an incorrect number of parameters has been sent.
1 Dh	The TCK check byte is incorrect in a microprocessor card ATR.
A0h	Error in the card reset response, such as unknown exchange protocol, or TA1 byte not recognized. The card is not supported. The card ATR is returned nonetheless.
A1h	Card protocol error (T=0/T=1).
A2h	Card malfunction. The card does not respond to the reset or has interrupted an exchange by timing out.
A3h	Parity error during a microprocessor exchange. This error only occurs after several unsuccessful attempts to resend.
A4h	Card has aborted chaining (T=1).
A5h	Reader has aborted chaining (T=1).
A6h	RESYNCH successfully performed by MiniSmart.
A7h	Protocol Parameter Selection error.
A8h	Card already powered on.
B0h	PC-Link command not supported.
E4h	The card has just sent an invalid "Procedure Byte" (see ISO 7816-3).
E5h	The card has interrupted an exchange (the card sends an SW1byte but more data remains to be sent or received).
E7h	Error returned by the card. The SW1 and SW2 bytes returned by the card are other than 90h 00h.
F7h	Card removed. The card has been withdrawn during the execution of a command. Check that the card instruction is not partially completed.
F8h	The card is consuming too much electricity or is short-circuiting.
FBh	Card missing. There is no card in the smart card interface. The card may have been removed when it was powered up, but no command has been interrupted.