PMD15282-AA
DRAFT - PUBLIC

SPRINGCARD PC/SC COUPLERS

CCID over Serial and CCID over TCP implementations

Document Identification

Category	Specification		
Family/Customer	CCID PC/SC Couplers		
Reference	PMD15282	Version	AA
Status	Draft	Classification	Public
Keywords	Smart Card, Integrated Circuit(s) Cards, PC/SC, CCID, RS-232, RS-TTL, CCID		
Abstract			

File name	V:\Dossiers\SpringCard\A-Notices\PCSC\CCID over Serial and over TCP\[PMD15282-AA] CCID Couplers over Serial or TCP.odt		
Date saved	29/07/15	Date printed	16/04/15

Revision History

Ver.	Date	Author	Valid. by		Approv. by	Details
			Tech.	Qual.		
AA	29/06/15	JDA				Draft

Contents

1. Introduction

1.1. Abstract

SpringCard's product range could be divided into USB devices on the one hand, and non-USB devices (serial, TCP over Ethernet) on the other hand.

Since 2009, **SpringCard**'s USB couplers take benefit of a full PC/SC compliance. It is achieved by the mean of a PC/SC driver (available for Windows and for most UNIX systems, including Linux and MacOS X), and by compliance with the USB CCID (Chip Card Interface Device) specification.

Starting with firmware version 2.00, **SpringCard**'s non-USB couplers also provide a new communication scheme which is consistent with the CCID specification. This new communication scheme supersedes the legacy "SpringProx" protocol[1].

Designed with simplicity and interoperability standards in mind, this CCID implementation aims to shorten the development and validation times, and paves the way for a new generation of PC/SC driver using another medium and not USB.

This document is the specification of **SpringCard's CCID over Serial and CCID over TCP implementations**. It provides all the information a developer would need to communicate with a **SpringCard** coupler from his application.

Documents [PMD15305] and *[TBD]* show how to use these couplers on top of this implementation (within the frame offered by the PC/SC standard).

1.2. Supported product

At the date of writing, the products covered by this document are

- **SpringCard K663** version 2.02 or newer, and all products based on **K663** core (**K663-232**, **K663-TTL**, **TwistyWriter-232**, **TwistyWriter-TTL**, **CSB4.6S**, **CSB4.6U**) for the CCID over Serial implementation,
- **SpringCard E663** version 2.02 or newer, and all products based on **E663** core (**FunkyGate IP NFC**, **FunkyGate IP+POE NFC**, **TwistyWriter IP**) for the CCID over TCP implementation.

1.3. Audience

This manual is designed for use by application developers. It assumes that the reader has expert knowledge of computer development and a basic knowledge of PC/SC, of the ISO 7816-4 standard for smartcards, and of the NFC Forum's specifications.

[1] The SpringProx protocol remains fully available; upward compliance in therefore ensure with systems using the legacy SpringProx API to communicate with the couplers.

1.4. Support and updates

Useful related materials (product datasheets, application notes, sample software, HOWTOs and FAQs…) are available at SpringCard's web site:

www.springcard.com

Updated versions of this document and others are posted on this web site as soon as they are available.

For technical support enquiries, please refer to SpringCard support page, on the web at

www.springcard.com/support

1.5. Related documents

1.5.1. CCID standard

http://www.usb.org/developers/docs/devclass_docs/DWG_Smart-Card_CCID_Rev110.pdf

1.5.2. Developer's guides

Reference	Publisher	Title
PMD15305	SpringCard	K663 CCID Developer's Guide
[TBD]	SpringCard	E663 CCID Developer's Guide

1.5.3. Products' specifications

Reference	Publisher	Title
PFL15108	SpringCard	'K' Series OEM Serial Contactless Couplers leaflet
PFL8T6P	SpringCard	CSB4 product information sheet
[TBD]	SpringCard	E663 CCID Developer's Guide

2. CCID over Serial

This chapters concerns the K663 OEM Couplers, and all products based on the K663 core.

2.1. Physical layer

The physical layer uses a serial asynchronous protocol. The communication is done over a UART at a baudrate could be defined by configuration.

The default configuration is 38400bps, 8 data bits, 1 stop bit, no parity.

No flow control is involved, which means that the serial communication uses only 2 lines (PC_To_RDR and RDR_To_PC).

2.2. Transport layer

2.2.1. Block format

Every block transmitted in the channel is formatted as follow:

START BYTE	ENDPOINT	HEADER	DATA	CHECKSUM
1 byte	1 byte	10 byte	0 to 262 bytes	1 byte

2.2.2. Description of the fields

Field	Description
START BYTE	The START BYTE is the constant value $_{h}$CD
ENDPOINT	The ENDPOINT byte is used to convey the CCID HEADER and DATA to the appropriate target service (as the USB endpoint feature does).
HEADER	For Bulk-Out and Bulk-In messages, the 10-B HEADER field follows [CCID]. For all the other messages, a proprietary format is defined.
DATA	For Bulk-Out and Bulk-In messages, the DATA field follows [CCID]. For all the other messages, a proprietary format is defined.
CHECKSUM	The CHECKSUM field is a XOR computed over all the bytes in the ENDPOINT, HEADER and DATA fields.

2.2.3. Values for the ENDPOINT field

Value	Symbolic name	Understanding
h00	CCID_COMM_CONTROL_TO_RDR	Control Endpoint (orders and queries from PC to RDR)
h80	CCID_COMM_CONTROL_TO_PC	Control Endpoint (answers from RDR to PC)
h81	CCID_COMM_BULK_TO_PC	Bulk-In Endpoint (RDR_to_PC responses)
h02	CCID_COMM_BULK_TO_RDR	Bulk-Out Endpoint (PC_to_RDR commands)
h83	CCID_COMM_INTERRUPT_TO_PC	Interrupt Endpoint (notifications from RDR to PC)

2.2.4. Size of the blocks

The size of every block can't be less than 13 bytes.

The size of every block can't exceed 275 bytes.

2.2.5. Timeout

When the Start Byte is received, the Coupler opens a 500ms time window. The host must transmit an entire block within this time window, otherwise the block is discarded.

2.3. Command layer

Chapter 4, 5, 6 and 7 contain the documentation of the Command Layer.

Chapter 8 and document [PMD15305] explain how to use the Coupler once the protocol is correctly implemented.

2.4. General communication flow

2.4.1. Session establishment

The PC tries to connect to the Coupler over a serial link by sending GET DESCRIPTOR commands (§ 4.3).

Once a Coupler has been found, the PC queries all the Coupler's descriptors, and, when ready, starts the Coupler using the SET CONFIGURATION command (§ 4.4).

No communication could occur on the Bulk-In, Bulk-Out or Interrupt endpoints before the SET CONFIGURATION command has been issued by the PC and acknowledged by the Coupler.

2.4.2. Half-duplex or full-duplex?

Depending on the underlying hardware, the serial link could be either full-duplex (RS-232, RS-TTL) or half-duplex (RS-485).

To prevent any collision on a half-duplex medium, the Coupler must be configured accordingly. The Option byte in the SET CONFIGURATION command (§ 4.4) is used to do so. When the PC says that the medium is half-duplex only, the Coupler will not send notifications on the Interrupt Endpoint.

As a consequence, the PC must monitor the card insertion/removal by sending PC_To_RDR_GetSlotStatus commands on the Bulk-Out endpoint periodically.

2.5. Error handling and recovery

2.5.1. For the Coupler

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the Coupler discard the block and remain silent,
- **Communication timeout:** if the PC takes more than 1000ms to send a block, the Coupler discard the block and remain silent.

2.5.2. For the PC

- **Malformed frame, Protocol violation:** in case it receives a block that doesn't obey to the block-formatting rules, the PC shall wait 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.4.1),
- **Communication timeout:** if the Coupler takes more than 1000ms to send a block (time between the Start Byte and the CRC), the PC shall wait at least 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.4.1),
- **Processing timeout:** the Coupler starts its answer (Start Byte) within 500ms for the commands sent to the Control Endpoint (CCID_COMM_CONTROL_TO_RDR), and within 1500ms for the commands sent to the Bulk-Out Endpoint (CCID_COMM_BULK_TO_RDR). If the Coupler doesn't answer within the specified time, the PC shall wait at least 2000ms, flush its input buffer, and run the Session establishment procedure again (§ 2.4.1).

2.5.3. Quick recovery

Instead of waiting 2000ms before running the Session establishment procedure again, the PC may

1. Reset the Coupler by setting the Coupler's /RESET pin to LOW level during at least 2ms,
2. Wait for the Coupler's startup string on the serial line (constant “K663”)[2],
3. Wait 50ms after the end of the startup string,
4. Run the Session establishment procedure.

[2] The time between the reset and the arrival of the “K663” startup string depends on how the bootloader is configured in the device. The host software shall be prepared to wait up to 1500ms to accommodate every configuration of the bootloader, but with the standard configuration the K663 starts in less than 75ms.

3. CCID over TCP

This chapters concerns the E663 OEM Couplers, and all products based on the E663 core.

3.1. TCP link

The **SpringCard** network-attached PC/SC Coupler behaves as TCP Server, and the Host is the Client.

Note that the Coupler is not able to accept more than one Client at the time. Trying to connect to the same Coupler from two different Host is not supported, and shall not be tried. An undefined behaviour may occur.

This chapter describes the **Plain Transport Layer**.

This Transport Layer is designed to support the transmission of variable-length blocks. The session-establishment makes it possible for both partners to check they are running the same protocol. The Host (Client) may also decide to switch to the optional Secure Transport Layer (chapter 9).

*The **Coupler** listens on TCP port 3999. This default value could be changed by writing into the IPP configuration register ($_h$81, § 10.2.2).*
To enforce security, the Plain Transport Layer could be disabled by setting bit 0 of byte 0 to 1 in the IPS configuration register ($_h$82, § 10.2.3). In this case, the Coupler will reject any Host that doesn't switch to the Secure Transport Layer.

3.2. Transport layer

3.2.1. Block format

Every block transmitted in the channel is formatted as follow:

ENDPOINT	HEADER	DATA
1 byte	10 byte	0 to 262 bytes

3.2.2. Description of the fields

Field	Description
ENDPOINT	The ENDPOINT byte is used to route the CCID HEADER and DATA to the appropriate target service (as the USB endpoint feature does).
HEADER	For Bulk-Out and Bulk-In messages, the 10-B HEADER field follows [CCID]. For all the other messages, a proprietary format is defined.
DATA	For Bulk-Out and Bulk-In messages, the DATA field follows [CCID]. For all the other messages, a proprietary format is defined.

3.2.3. Values for the ENDPOINT field

Value	Symbolic name	Understanding
h00	CCID_COMM_CONTROL_TO_RDR	Control Endpoint (orders and queries from PC to RDR)
h80	CCID_COMM_CONTROL_TO_PC	Control Endpoint (answers from RDR to PC)
h81	CCID_COMM_BULK_TO_PC	Bulk-In Endpoint (RDR_to_PC responses)
h02	CCID_COMM_BULK_TO_RDR	Bulk-Out Endpoint (PC_to_RDR commands)
h83	CCID_COMM_INTERRUPT_TO_PC	Interrupt Endpoint (notifications from RDR to PC)

3.2.4. Size of the blocks

The size of every block can't be less than 11 bytes.

The size of every block can't exceed 273 bytes in plain communication.

If the secure communication is used (see chapter 9), the size of every block is fixed to 289 bytes for Bulk-Out and Bulk-In, and to 33 bytes for Interrupt.

3.3. Command layer

Chapter 4, 5, 6 and 7 contain the documentation of the Command Layer.

Chapter 8 and document *[TBD]* explains how to use the Coupler once the protocol is correctly implemented.

3.4. General communication flow

3.4.1. Session establishment

The PC tries to connect to one (or many) Couplers.

When a connection is established on the Coupler, the PC queries the Coupler's descriptors, and, when ready, starts the Coupler using the SET CONFIGURATION command (§ 4.4).

No communication could occur on the Bulk-In, Bulk-Out or Interrupt endpoints before the SET CONFIGURATION command has been issued by the PC and acknowledged by the Coupler.

3.4.2. Nominal dialogue

The TCP channel is full-duplex; both the Coupler and the PC may send at any time, and therefore must be ready to receive at any time.

The PC may send GET STATUS commands to monitor the link (§ 4.2). The Coupler then sends a GET STATUS response within 500ms max.

3.5. Error handling and recovery

3.5.1. For the Coupler

- **Too many hosts:** when receiving a valid SET CONFIGURATION command, the Coupler drops any previous connection,
- **Malformed frame, Protocol violation:** in case it receives a frame that doesn't obey to the block-formatting rules, the Coupler drops the connection,
- **No activity error:** if the PC remains silent for 120s, the Coupler drops the connection.

3.5.2. For the PC

- **Malformed frame, Protocol violation:** in case it receives a frame that doesn't obey to the block-formatting rules, the PC shall drop the connection,
- **Timeout error:** if the PC doesn't receive an answer to a GET STATUS command within 1s + (estimated network time), the PC shall drop the connection.

3.5.3. Recovery

If the connection is dropped for any reason, the PC shall wait at least 5s before trying to connect again to the same Coupler.

4. Command Layer – Control Endpoint

The commands/responses described in this chapter are proprietary, yet inspired for the largest part by the USB Specification [USB].

4.1. List of Control message pairs

ID	Control request	See
h00	GET STATUS	4.2
h06	GET DESCRIPTOR	4.3
h09	SET CONFIGURATION	4.4

4.2. Get Status command/response

The GET STATUS command/response pair is used to monitor the link.

a. GET STATUS command format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h00	CCID_COMM_CONTROL_TO_RDR
1	Request number	1	h00	GET_STATUS
2	Data Length	4	h00000000	*Data field is empty*
6	Value	2	h0000	*Not used*
8	Index	2	h0000	*Not used*
10	Option	1	h00	*Not used*

b. GET STATUS response format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	h00	GET_STATUS
2	Data Length	4	h00000000	*Data field is empty*
6	Value	2	h0000	*Not used*
8	Index	2	h0000	*Not used*
10	Status	1	h00	*Not used*

4.3. Get Descriptor command/response

4.3.1. Command/response format

a. GET DESCRIPTOR command format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h00	CCID_COMM_CONTROL_TO_RDR
1	Request number	1	h06	GET_DESCRIPTOR
2	Data Length	4	h00000000	*Data field is empty*
6	Value_L	1		Descriptor Type
7	Value_H	1		Descriptor Index
8	Index	2		For String descriptor, the Language ID; otherwise zero. *Not used by the Coupler.*
10	Option	1	h00	*Not used*

b. GET DESCRIPTOR response format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	h06	GET_DESCRIPTOR
2	Data Length	4		The length of the Descriptor (L)
6	Value_L	1		Same as Descriptor Type specified by the PC
7	Value_H	1		Same as Descriptor Index specified by the PC
8	Index	2		Same as the Index field specified by the PC
10	Status	1	h00	*Not used*
11	Data	L		The content of the Descriptor

4.3.2. List of available descriptors

Value_L	Value_H	Retrieves	See
Descr. Type	Descr. Index		
h01	h00	The Device descriptor	4.3.2.a
h02	h00	The Configuration descriptor	4.3.2.b
h03	h01	The Vendor Name (“SpringCard”)	
h03	h02	The Product Name	
h03	h03	The Product Serial Number	

a. The Device descriptor

The Device descriptor is defined as follow:

Offset	Field	Size	Value	Description / remark
0	Size	1	h12	
1	Type	1	h01	This is a Device descriptor
2	USB version	2	h0200	*Don't care*
4	Class	1	h00	
5	SubClass	1	h00	
6	Protocol	1	h00	
7	MaxPacketSize0	1	h00	*Don't care*
8	Vendor ID	2	h1C34	Pro Active / SpringCard
10	Product ID	2		The Product ID
12	Version	2		The Firmware Version
14	iManufacturer	1	h01	Index of the Vendor Name string
15	iProduct	1	h02	Index of the Product Name string
16	iSerialNumber	1	h03	Index of the Product Serial Number string
17	Configurations	1	h01	Only one configuration supported

b. The Configuration descriptor

The Configuration descriptor is defined as follow:

Offset	Field	Size	Value	Description / remark
Configuration part				
0	Size	1	h09	
1	Type	1	h02	This is a Configuration descriptor
2	Total size	2		
4	Interfaces	1	h01	Only one interface
5	Configuration	1	h01	This is the first (and single) configuration
6	iConfiguration	1	h00	No string to describe the Configuration
7	Attributes	1	h00	*Don't care*
8	MaxPower	1	h00	*Don't care*
Interface part				
9	Size	1	h09	
10	Type	1	h04	This is an Interface descriptor
11	Interface	1	h00	Interface number = 0
12	AlternateSettings	1	h00	*Don't care*
13	Endpoints	1	h03	3 endpoints
14	Class	1	h0B	Class is CCID
15	SubClass	1	h00	
16	Protocol	1	h00	
17	iInterface	1	h00	No string to describe the Interface
CCID-specific part				
18	Size	1	h36	
19	Type	1	h21	Specific to CCID
20	Version	2	h0110	Version of CCID implementation (1.10)
22	MaxSlotIndex	1		The number of CCID slots, minus 1
23	VoltageSupport	1		
24	Protocols	4	h00000003	Supports T=0 and T=1
28	DefaultClock	4	h0000A00F	Default clock is 4MHz (dummy value)

32	MaximumClock	4	h0000A00F	Manimum clock is 4MHz (dummy value)
36	NumClockSupported	1	h00	
37	DataRate	4		
41	MaxDataRate	4		
45	NumDataRateSupported	1		
46	MaxIFSD	4		
50	SynchProtocols	4		
54	Mechanical	4		
58	Features	4		
62	MaxCCIDMessageLength	4		
66	ClassGetResponse	1	hFF	
67	ClassEnvelope	1	hFF	
68	LcdLayout	2	h0000	
70	PinSupport	1	h00	
71	MaxCCIDBusySlot	1	h01	
1st endpoint (Bulk In)				
72	Size	1	h07	
73	Type	1	h05	This is an Endpoint descriptor
74	Address	1	h81	EP1, RDR to PC
75	Attributes	1	h02	Bulk
76	MaxPacketSize	2	h0118	Up to 280 bytes
78	Interval	1	h00	*Don't care*
2nd endpoint (Bulk Out)				
72	Size	1	h07	
73	Type	1	h05	This is an Endpoint descriptor
74	Address	1	h02	EP2, PC to RDR
75	Attributes	1	h02	Bulk
76	MaxPacketSize	2	h0118	Up to 280 bytes
78	Interval	1	h00	*Don't care*
3rd endpoint (Interrupt In)				
72	Size	1	h07	

73	Type	1	h05	This is an Endpoint descriptor
74	Address	1	h83	EP3, RDR to PC
75	Attributes	1	h03	Interrupt
76	MaxPacketSize	2	h0118	Up to 280 bytes
78	Interval	1	h01	*Don't care*

4.3.3. Response to a query for an unknown descriptor

If the Coupler receives an unsupported message on the Control Endpoint, it sends an empty GET DESCRIPTOR response, as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	h06	GET_DESCRIPTOR
2	Data Length	4	h00000000	Empty descriptor
6	Value_L	1		Same as Descriptor Type specified by the PC
7	Value_H	1		Same as Descriptor Index specified by the PC
8	Index	2		Same as the Index field specified by the PC
10	Status	1	h00	*Not used*

4.4. Set Configuration command/response

a. SET CONFIGURATION command format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$00	CCID_COMM_CONTROL_TO_RDR
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value	2		$_h$0001 to start the Coupler $_h$0000 to stop the Coupler
8	Index	2	$_h$0000	*Not used*
10	Option	1		*Serial RDR:* $_h$00 half-duplex link, do not use Interrupt $_h$01 full-duplex link, use Interrupt $_h$03 full-duplex link, use Interrupt and LPCD *TCP RDR:* $_h$00 no authentication, no security $_h$10 authenticated mode, plain communication $_h$30 authenticated mode, secure comm.

The authentication and the secure communication of the TCP RDR are described in chapter 9.

b. SET CONFIGURATION response format

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value	2		Same as the Value specified by the PC
8	Index	2	$_h$0000	Same as the Index specified by the PC
10	Status	1		$_h$00 Coupler is stopped $_h$01 Coupler is running $_h$FF An error has occurred

4.5. Answers to unsupported messages

If the Coupler receives an unsupported command, or a command with invalid parameters, it sends a GET STATUS response claiming an error, as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	$_h$00	GET_STATUS
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value	2	$_h$0000	*Not used*
8	Index	2	$_h$0000	*Not used*
10	Status	1	$_h$FF	Unsupported command

5. Command Layer – Bulk-Out Endpoint (PC to RDR messages)

All Bulk-Out commands described in this chapter are documented with more details in the CCID Specifications [CCID].

5.1. List of supported/unsupported Bulk-Out messages

ID	Command message	See	Response message	Supported
h61	PC_To_RDR_SetParameters		RDR_To_PC_Parameters	✗
h62	PC_To_RDR_IccPowerOn	5.3.1	RDR_To_PC_DataBlock	✓
h63	PC_To_RDR_IccPowerOff	5.3.2	RDR_To_PC_SlotStatus	✓
h65	PC_To_RDR_GetSlotStatus	5.3.3	RDR_To_PC_SlotStatus	✓
h69	PC_To_RDR_Secure			✗
h6A	PC_To_RDR_T0APDU			✗
h6B	PC_To_RDR_Escape	5.3.5	RDR_To_PC_Escape	✓
h6C	PC_To_RDR_GetParameters		RDR_To_PC_Parameters	✗
h6D	PC_To_RDR_ResetParameters		RDR_To_PC_Parameters	✗
h6E	PC_To_RDR_IccClock			✗
h6F	PC_To_RDR_XfrBlock	5.3.4	RDR_To_PC_DataBlock	✓
h71	PC_To_RDR_Mechanical			✗
h72	PC_To_RDR_Abort		RDR_To_PC_SlotStatus	✓
h73	PC_To_RDR_SetDataRateAnd ClockFrequency			✗

5.2. Binding to the Transport Layers

a. Binding to a Serial Transport

For a Serial communication, the Bulk-Out message is sent by the PC in the following format:

START BYTE	ENDPOINT	CCID command	CHECKSUM
$_h$CD	$_h$02	10 + (0 to 262) bytes	1 byte

b. Binding to a TCP Transport

For a TCP communication, the Bulk-Out message is sent by the PC in the following format:

ENDPOINT	CCID command
$_h$02	10 + (0 to 262) bytes

5.3. Details

5.3.1. PC_To_RDR_IccPowerOn

The PC_to_RDR_IccPowerOn command allows to get the ATR of the card inserted in a slot.

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h02	CCID_COMM_BULK_PC_TO_RDR
1	Message Type	1	h62	PC_TO_RDR_ICCPOWERON
2	Data Length	4	h00000000	
6	Slot	1		Identifies the slot number for this command
7	Sequence	1		Sequence number for command
8	Power Select	1	h00	Automatic Voltage Selection only
9	RFU	2	h0000	*Reserved for future use*

The response to this command is the RDR_to_PC_DataBlock message (§ 6.3.1); the Data returned is the card's ATR (Answer To Reset).

5.3.2. PC_To_RDR_IccPowerOff

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h02	CCID_COMM_BULK_PC_TO_RDR
1	Message Type	1	h63	PC_TO_RDR_ICCPOWEROFF
2	Data Length	4	h00000000	
6	Slot	1		Identifies the slot number for this command
7	Sequence	1		Sequence number for command
8	RFU	3	h000000	*Reserved for future use*

The response to this command is the RDR_to_PC_SlotStatus message (§ 6.3.2).

5.3.3. PC_To_RDR_GetSlotStatus

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h02	CCID_COMM_BULK_PC_TO_RDR
1	Message Type	1	h65	PC_TO_RDR_GETSLOTSTATUS
2	Data Length	4	h00000000	
6	Slot	1		Identifies the slot number for this command
7	Sequence	1		Sequence number for command
8	RFU	3	h000000	*Reserved for future use*

The response to this command is the RDR_to_PC_SlotStatus message (§ 6.3.2).

5.3.4. PC_To_RDR_XfrBlock

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h02	CCID_COMM_BULK_PC_TO_RDR
1	Message Type	1	h6F	PC_TO_RDR_XFRBLOCK
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Identifies the slot number for this command
7	Sequence	1		Sequence number for command
8	BWI	1	h00	*Not used*
9	Level Parameter	2	h0000	*Not used*
11	Data	Var.		Data block sent to the card. Only a maximum length of 262 bytes is supported

The response to this command is the RDR_to_PC_DataBlock message (§ 6.3.1); the Data returned is the response from the card.

5.3.5. PC_To_RDR_Escape

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h02	CCID_COMM_BULK_PC_TO_RDR
1	Message Type	1	h6B	PC_TO_RDR_ESCAPE
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Identifies the slot number for this command
7	Sequence	1		Sequence number for command
8	RFU	3	h000000	*Reserved for future use*
11	Data	Var.		Data block sent to the RDR. Only a maximum length of 262 bytes is supported

The response to this command is the RDR_to_PC_Escape message (§ 6.3.3); the Data returned is the response from the RDR.

6. Command Layer – Bulk-In Endpoint (RDR to PC messages)

All Bulk-In responses described in this chapter are documented with more details in the CCID Specifications [CCID].

6.1. List of supported/unsupported Bulk-In messages

ID	Response message	See	In answer to these commands	Supported
h80	PC_To_RDR_DataBlock	6.3.1	PC_To_RDR_IccPowerOn RDR_To_PC_XfrBlock	✓
h81	RDR_To_PC_SlotStatus	6.3.2	PC_To_RDR_IccPowerOff PC_To_RDR_GetSlotStatus PC_To_RDR_Abort	✓
h82	RDR_To_PC_Parameters			✗
h83	RDR_To_PC_Escape	6.3.3	PC_To_RDR_Escape	✓
h84	RDR_To_PC_DataRateAnd ClockFrequency			✗

6.2. Binding to the Transport Layers

a. *Binding to a Serial Transport*

For a Serial communication, the Bulk-In message is sent by the RDR in the following format:

START BYTE	ENDPOINT	CCID response	CHECKSUM
$_h$CD	$_h$81	10 + (0 to 262) bytes	1 byte

b. *Binding to a TCP Transport*

For a TCP communication, the Bulk-Out message is sent by the RDR in the following format:

ENDPOINT	CCID response
$_h$81	10 + (0 to 262) bytes

6.3. Details

6.3.1. RDR_To_PC_DataBlock

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$81	CCID_COMM_BULK_RDR_TO_PC
1	Message Type	1	$_h$80	RDR_TO_PC_DATABLOCK
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Status	1		Slot Status as defined in § 6.4.1
9	Error	1		Slot Error as defined in § 6.4.2
10	RFU	1	$_h$00	*Reserved for future use*
11	Data	Var.		Data block retrieved from the card. Only a maximum length of 262 bytes is supported

6.3.2. RDR_To_PC_SlotStatus

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h81	CCID_COMM_BULK_RDR_TO_PC
1	Message Type	1	h81	RDR_TO_PC_SLOTSTATUS
2	Data Length	4	h00000000	
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Status	1		Slot Status as defined in § 6.4.1
9	Error	1		Slot Error as defined in § 6.4.2
10	ClockStatus	1	h00	Other values are not supported

6.3.3. RDR_To_PC_Escape

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	h81	CCID_COMM_BULK_RDR_TO_PC
1	Message Type	1	h83	RDR_TO_PC_ESCAPE
2	Data Length	4		Size of the Data field of this message
6	Slot	1		Same as last PC_TO_RDR message
7	Sequence	1		
8	Status	1		Slot Status as defined in § 6.4.1
9	Error	1		Slot Error as defined in § 6.4.2
10	RFU	1	h00	*Reserved for future use*
11	Data	Var.		Data block sent by the RDR. Only a maximum length of 262 bytes is supported

6.4. Values of the Status and Error fields

Each RDR_To_PC message contains a Slot Status and a Slot Error field.

6.4.1. Slot Status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Command Status		*Reserved for future use*				**Card Status**	
See below		0	0	0	0	*See below*	

a. Card Status field

Bit 1	Bit 0	Description
Card Status		
0	0	A Card is present and active (powered ON)
0	1	A Card is present and inactive (powered OFF or hardware error)
1	0	No card present (slot is empty)
1	1	*Reserved for future use*

b. Command Status field

Bit 7	Bit 6	Description
Command Status		
0	0	Command processed without error
0	1	Command failed (error code is provided in the Slot Error field)
1	0	Time Extension is requested
1	1	*Reserved for future use*

6.4.2. Slot Error

This field is valid only if Command Status = $_b$01 in the Slot Status field.

Error code	Error name	Possible cause
$_h$FF	CMD_ABORTED	The PC has sent an ABORT command
$_h$FE	ICC_MUTE	Time out in Card communication
$_h$FD	XFR_PARITY_ERROR	Parity error in Card communication
$_h$FC	XFR_OVERRUN	Overrun error in Card communication
$_h$FB	HW_ERROR	Hardware error on Card side (over-current?)
$_h$F8	BAD_ATR_TS	Invalid ATR format
$_h$F7	BAD_ATR_TCK	Invalid ATR checksum
$_h$F6	ICC_PROTOCOL_NOT_SUPPORTED	Card's protocol is not supported
$_h$F5	ICC_CLASS_NOT_SUPPORTED	Card's power class is not supported
$_h$F4	PROCEDURE_BYTE_CONFLICT	Error in T=0 protocol
$_h$F3	DEACTIVATED_PROTOCOL	Specified protocol is not allowed
$_h$F2	BUSY_WITH_AUTO_SEQUENCE	RDR is currently busy activating a Card
$_h$E0	CMD_SLOT_BUSY	RDR is already running a command
$_h$00		Command not supported

7. Command Layer – Interrupt Enpoint (RDR to PC notifications)

In order to ease the implementation of the receiving logic, the CCID Interrupt frame is extended to reach the same length as the bulk messages.

7.1. List of supported Interrupt messages

ID	Response message	See	Supported
h50	PC_To_RDR_NotifySlotChange	7.3.1	✓
h51	PC_To_RDR_HardwareError		✗

7.2. Binding to the Transport Layers

a. Binding to a Serial Transport

For a Serial communication, the Interrupt message is sent by the RDR in the following format:

START BYTE	ENDPOINT	CCID notification	CHECKSUM
hCD	h83		1 byte

b. Binding to a TCP Transport

For a TCP communication, the Interrupt message is sent by the RDR in the following format:

ENDPOINT	CCID notification
h83	

7.3. Details

7.3.1. PC_To_RDR_NotifySlotChange

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$83	CCID_COMM_INTERRUPT_TO_PC
1	Message Type	1	$_h$50	
2	Data Length	4		Size of the SlotICCState array
6	RFU	5	$_H$00 … $_h$00	*Reserved for future use*
11	SlotICCState			This field is reported on byte granularity. The size is (2 bits * number of slots) rounded up to the nearest byte. Each slot has 2 bits. The least significant bit reports the current state of the slot (0 = no ICC present, 1 = ICC present). The most significant bit reports whether the slot has changed state since the last RDR_to_PC_NotifySlotChange message was sent (0 = no change, 1 = change). If no slot exists for a given location, the field returns 00 in those 2 bits. Example: A 3 slot CCID reports a single byte with the following format: Bit 0 = Slot 0 current state Bit 1 = Slot 0 changed status Bit 2 = Slot 1 current state Bit 3 = Slot 1 changed status Bit 4 = Slot 2 current state Bit 5 = Slot 2 changed status Bit 6 = 0 Bit 7 = 0

8. Mapping PC/SC calls to PC_To_RDR / RDR_To_PC messages

The PC/SC specification and Microsoft's reference implementation define a short set of functions to work with PC/SC Couplers (and with Cards through a PC/SC Coupler).

This chapter explains how the very basic PC/SC functions shall be implemented.

8.1.1. SCardStatus

In PC/SC, the **SCardStatus** function has two roles:

- Check whether there's a Card in a Coupler or not,
- If a Card is present in the Coupler, return its ATR.

The first role is directly available from CCID, using the PC_To_RDR_GetSlotStatus message:

1. Send the **PC_To_RDR_GetSlotStatus** command (§ 5.3.3),
2. The Coupler returns a **RDR_To_PC_SlotStatus** response (§ 6.3.2),
3. Observe the **Card Status field** in the response (§ 6.4.1.a) to know whether a Card is present and powered, present and not powered, or absent.

To retrieve the Card's ATR, the PC must activate the Card explicitely – this is done using the PC_To_RDR_IccPowerOn message when a Card is present:

4. Send the **PC_To_RDR_IccPowerOn** command (§ 5.3.1),
5. The Coupler returns a **RDR_To_PC_Datablock** response (§ 6.3.1),
6. Observe the **Status and Error fields** in the response (§ 6.4.1). If both fields are zero, communication with the Card is OK, and **the Data field contains the Card's ATR** (Answer To Reset)[3].

8.1.2. SCardConnect

In PC/SC, the **SCardConnect** function has two roles:

- Open a handle to communicate with the Card – the computer's PC/SC subsystem manages concurrent or exclusive access, and take care to release the handle if the application stops for any reason,
- Tells the Coupler which protocol shall be used to communicate with the Card (T=0 or T=1).

[3] For a Contactless Card (PICC/VICC), the ATR is constructed according to PC/SC v2 chapter 3 rules.

None of these two roles are significant for “low level” communication with a CCID Coupler[4]. The actual “Card power up” role is provided by PC_To_RDR_IccPowerOn, which has already been invoked by **SCardStatus** to retrieve the ATR.

8.1.3. SCardTransmit

In PC/SC, the **SCardTransmit** function is the very function that implements the exchange of APDUs (Application Protocol Datagram Units) with a contact or contactless Smart Card.

1. Send your **C-APDU** (application-level Command) in the **Data field** of a **PC_To_RDR_XfrBlock** command (§ 5.3.4),
2. The Coupler returns a **RDR_To_PC_Datablock** response (§ 6.3.1),
3. Observe the **Status and Error fields** in the response (§ 6.4.1). If both fields are zero, communication with the Card is OK, and **the Data field contains the Card's R-APDU** (application-level Response)

> *Note that for a wired logic contactless Card (PICC not compliant with ISO 14443-4 or VICC), the SCardTransmit function doesn't actually communicate with the Card, but with the Coupler's APDU Interpreter which is responsible for translating the “standard” APDU into a low-level, proprietary command specific to the Card. This is the same here, the PC_To_RDR_XfrBlock command goes through the Coupler's APDU Interpreter and not directly to a wired-logic PICC/VICC.*

8.1.4. SCardDisconnect

In PC/SC, the **SCardDisconnect** function has two roles:

- Close the handle that has been opened by **SCardConnect**,
- Tells the Coupler to power down the Card.

The first role is not relevant here.

The second role is taken by the PC_To_RDR_IccPowerOff message:

1. Send the **PC_To_RDR_IccPowerOff** command (§ 5.3.2),
2. The Coupler returns a **RDR_To_PC_SlotStatus** response (§ 6.3.2),
3. Observe the **Card Status field** in the response (§ 6.4.1.a) to know whether the Card has already been removed, or is still present in the Coupler.

[4] All SpringCard couplers feature automatic protocol activation, which mean that the protocol is always selected by the Coupler, not by the application.

8.1.5. SCardControl

In PC/SC, the **SCardControl** function is used to send “low level” commands to the Coupler, to the Coupler's driver, or to the PC/SC subsystem.

In our case, only sending “low level” commands to the Coupler makes sense. This is done using the PC_To_RDR_Escape message.

1. Send the low level command in the **Data field** of a **PC_To_RDR_Escape** command (§ 5.3.5),
2. The Coupler returns its answer in a **RDR_To_PC_Escape** response (§ 6.3.3).

> *Sending an escape sequence through PC_To_RDR_Escape is exactly the same as sending a “legacy command” to a **SpringCard** coupler running in **legacy (SpringProx) mode**.*

9. Secure communication mode over TCP

9.1. Abstract

Since any PC that connects to the RDR and sends a valid SET CONFIGURATION command will disconnect any currently connected PC, an authentication scheme must be introduced to make sure that only on of the 'expected' PC takes control of the RDR.

More than that, reading and writing a smartcard may imply exchanging 'sensitive' data on the CCID channel, which is, for a TCP RDR, a potentially large Network. Therefore, the communication between the PC and the RDR could be encrypted when needed.

9.2. Cryptographic background

The RDR uses the **AES block cipher** (Rijndael) . AES has a fixed 128-bit (16 bytes) block size. The RDR supports **128-bit keys** (16 bytes) only.

In the following paragraphs,

- E (*K* , *P*) means “AES encrypt operation (cipher) over block *P* using the key *K*”,
- D (*K* , *C*) means “AES decrypt operation (decipher) over block *C* using the key *K*”.

Note that the size of blocks *P* and *C* must exactly 16 bytes.

When more than one block are involved, the encrypt and decrypt operations are performed **in CBC (cipher block chaining) mode**.

In the following paragraphs,

- E_{CBC} (*K* , *V* , *P*) means “AES encrypt operation (cipher) over buffer *P* using the key *K* and the Init Vector *V*”,
- D_{CBC} (*K* , *V* , *C*) means “AES decrypt operation (decipher) over buffer *C* using the key *K* and the Init Vector *V*”.

Note that the size of buffers *P* and *C* must be a multiple of 16 bytes. As a consequence, a padding is generally involved.

9.3. Configuration

9.3.1. Configuring the authentication mode

If the RDR is configured for enforced authentication mode, the authentication is mandatory. RDR drops the connection if it receives a SET CONFIGURATION command that doesn't ask to perform the authentication.

If the RDR is not configured for enforced authentication mode, the authentication remains optional.

9.4. 3-pass authentication

The 3-pass authentication is initiated by the PC after retrieving the RDR's descriptors, using a special SET CONFIGURATION command.

9.4.1. Step 0, PC to RDR

The PC sends to the RDR a SET CONFIGURATION command which asks to perform the authentication.

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$00	CCID_COMM_CONTROL_TO_RDR
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000000	*Data field is empty*
6	Value	2		$_h$0001 to start the RDR
8	Index	2	$_h$0000	*Not used*
10	Option	1		$_h$10 authenticated mode, plain communication $_h$30 authenticated mode, secure communication

9.4.2. Step 1, RDR to PC

- The RDR activates its **secret key K_{AUTH},**
- The RDR generates a random challenge (C_R) on 16 bytes,
- The RDR sends to the PC a block containing E (K_{AUTH}, C_R) in a SET CONFIGURATION response

This SET CONFIGURATION response is formatted as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000010	16 bytes of data
6	Value	2	$_h$0000	*Don't care*
8	Index	2	$_h$0000	*Don't care*
10	Status	1	$_h$00	*Don't care*
11	Challenge	16		E (K_{AUTH}, C_R)

The Init Vector of the AES cipher is cleared to (00..00) before computing E (K_{AUTH}, C_R).
1 block is crypted, no padding is applied.

9.4.3. Step 2, PC to RDR

- The PC activates the secret key K_{AUTH},
- The PC decrypts the payload received from the RDR, and retrieves C_R,
- The PC computes $C_R' = C_R << 1 \,||\, C_R >> 127$ (shift left with carry),
- The PC generate a random challenge (C_H) on 16 bytes,
- The PC sends to the RDR a block containing E (K_{Auth}, $C_H \,||\, C_R'$) in a new SET CONFIGURATION command

This SET CONFIGURATION command is formatted as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$00	CCID_COMM_CONTROL_TO_RDR
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000020	32 bytes of data
6	Value	2	$_h$0000	*Don't care*
8	Index	2	$_h$0000	*Don't care*
10	Option	1	$_h$00	*Don't care*
11	Challenge	32		E (K_{AUTH}, $C_H \,||\, C_R'$)

The Init Vector of the AES cipher is cleared to (00..00) before computing E (K_{AUTH}, $C_H \,||\, C_R'$).
2 blocks are crypted in CBC mode, no padding is applied.

9.4.4. Step 3, RDR to PC

- The RDR decrypts the payload received from the PC, and retrieves C_H and C_R',
- The RDR checks that C_R' is valid. This is the proof that the PC knows the secret key,
- The RDR computes $C_H' = C_H << 1 \,||\, C_H >> 127$ (shift left with carry),
- The RDR sends to the PC a block containing E (K_S, C_H') in a SET CONFIGURATION response

This SET CONFIGURATION response is formatted as follow:

Offset	Field	Size	Value	Description / remark
0	ENDPOINT	1	$_h$80	CCID_COMM_CONTROL_TO_PC
1	Request number	1	$_h$09	SET_CONFIGURATION
2	Data Length	4	$_h$00000020	32 bytes of data
6	Value	2	$_h$0000	*Don't care*
8	Index	2	$_h$0000	*Don't care*
10	Status	1	$_h$01	R is running
11	Challenge	32		E (K_{AUTH}, C_H') on 16 bytes

The Init Vector of the AES cipher is cleared to (00..00) before computing E (K_{AUTH}, C_H').
1 block is crypted, no padding is applied.

9.5. Secure communication

If the PC enables the Secure communication mode (Option byte set to $_h$30 in the SET CONFIGURATION command), all the exchanges taking place over the Bulk-Out, Bulk-In and Interrupt endpoint are secure.

The communication is secured by the combination of:

- A **8-byte CMAC** computed over the plain-text using a session **CMAC key** K_{CMAC},
- The **AES-CBC encryption** of the plain-text plus the CMAC using a session **cipher key** K_{SESS},
- The synchronisation the synchronisation of the **Init Vectors (IV)** between sender and receiver, to prevent any kind of injection.
- Both K_{CMAC} and K_{SESS} are “disposable” keys, generated from the random challenges exchanged during the authentication process. The static authentication key (K_{AUTH}) is never used again after the authentication.

The exchanges over the Control Endpoint (GET STATUS as keep-alive typically) stay in plain mode.

9.5.1. Generation of the disposable keys

a. CMAC Key

Let C_R be the Device's random challenge (§ 9.4.2). C_R is a 16-byte value ($C_R[0]$ ... $C_R[15]$).

Let C_H be the Host's random challenge (§ 9.4.3). C_H is a 16-byte value ($C_H[0]$ ... $C_H[15]$).

Let K_{AUTH} be the key used for authentication.

Construct T, a 16-byte buffer, as follow:

- $T[0] = C_H[7]$
- $T[1] = C_H[8]$
- $T[2] = C_H[9]$
- $T[3] = C_H[10]$
- $T[4] = C_H[11]$
- $T[5] = C_R[7]$
- $T[6] = C_R[8]$
- $T[7] = C_R[9]$
- $T[8] = C_R[10]$
- $T[9] = C_R[11]$
- $T[10] = C_H[0] \oplus C_R[0]$
- $T[11] = C_H[1] \oplus C_R[1]$
- $T[12] = C_H[2] \oplus C_R[2]$
- $T[13] = C_H[3] \oplus C_R[3]$
- $T[14] = C_H[4] \oplus C_R[4]$
- $T[15] = {}_h22$

Compute $K_{CMAC} = E (K_{AUTH}, T)$.

b. Encryption Key

Let C_R be the Device's random challenge (§ 9.4.2). C_R is a 16-byte value ($C_R[0]$ ... $C_R[15]$).

Let C_H be the Host's random challenge (§ 9.4.3). C_H is a 16-byte value ($C_H[0]$ ... $C_H[15]$).

Let K_{AUTH} be the key used for authentication.

Construct T, a 16-byte buffer, as follow:

- $T[0] = C_H[11]$
- $T[1] = C_H[12]$
- $T[2] = C_H[13]$
- $T[3] = C_H[14]$
- $T[4] = C_H[15]$
- $T[5] = C_R[11]$
- $T[6] = C_R[12]$
- $T[7] = C_R[13]$
- $T[8] = C_R[14]$
- $T[9] = C_R[15]$

- $T[10] = C_H[4] \oplus C_R[4]$
- $T[11] = C_H[5] \oplus C_R[5]$
- $T[12] = C_H[6] \oplus C_R[6]$
- $T[13] = C_H[7] \oplus C_R[7]$
- $T[14] = C_H[8] \oplus C_R[8]$
- $T[15] = {}_h11$

Compute $K_{SESS} = E (K_{AUTH}, T)$.

9.5.2. Format of the secure blocks

a. Bulk-Out Endpoint (PC to RDR commands)

If secure communication is enabled, the Bulk-Out message is sent by the PC in the following format:

	ENDPOINT	CCID message	CMAC	Padding
Plain	h02	10 to 272 bytes	8 bytes	8 to 270 bytes
Ciphered	h02	Encrypted payload – 288 bytes exactly		

b. Bulk-In Endpoint (RDR to PC responses)

If secure communication is enabled, the Bulk-In message is sent by the RDR in the following format:

	ENDPOINT	CCID message	CMAC	Padding
Plain	h81	10 to 272 bytes	8 bytes	8 to 270 bytes
Ciphered	h81	Encrypted payload – 288 bytes exactly		

c. Interrupt Endpoint (RDR to PC notifications)

If secure communication is enabled, the Interrupt message is sent by the RDR in the following format:

	ENDPOINT	CCID message	CMAC	Padding
Plain	h83	11 to 15 bytes	8 bytes	6 to 9 bytes
Ciphered	h83	Encrypted payload – 32 bytes exactly		

9.5.3. CMAC

a. Sequence numbers

The calculation of the CMAC includes a 32-bit Sequence number, to protect against the injection or the removal of frames. Both the PC and the RDR shall maintain 2 Sequence numbers for the CMAC:

- **SEQ_H is used by the PC to compute its outgoing CMAC**, and by the RDR to verify the its incoming CMAC. SEQ_H is incremented every time the PC sends a frame on the Bulk-Out Endpoint.
- **SEQ_R is used by the RDR to compute its outgoing CMAC**, and by the PC to verify the its incoming CMAC. SEQ_R is incremented every time the RDR sends a frame either on the Bulk-In or the Interrupt Endpoint.

SEQ_H and SEQ_R are not related to the Sequence *field present in the CCID header.*

Both SEQ_H and SEQ_R are cleared at the end of the authentication process (§ 9.4) and evolve independently afterwards.

b. Computing the CMAC

Let P be the (plain) CCID message. P is an arbitrary-length buffer between 10 and 272 bytes.

Construct H, an 8-byte buffer, as follow:

- H[0] to H[3] = SEQ_H or SEQ_R (sequence number of the sender), expressed in MSB-first format
- H[4] = ENDPOINT of the block ($_h$02, $_h$81 or $_h$83)
- H[5] = $_h$CD
- H[6] to H[7] = length of the (plain) CCID message, expressed in MSB-first format

Construct T = H || P

If size of T is not a multiple of 16 bytes, padd T as follow:

- T = T || $_h$80
- While size of T is not a multiple of 16 bytes, T = T || $_h$00

Compute $C = E_{CBC}(K_{CMAC} , {}_h00 .. {}_h00 , T)$

(T encrypted in CBC mode, using K_{CMAC} and an all-zero Init Vector)

Keep C_{LAST} , the last 16 bytes of C.

Extract CMAC, a 8-byte buffer, as follow:

- CMAC[0] = C_{LAST}[0]
- CMAC[1] = C_{LAST}[2]
- CMAC[2] = C_{LAST}[4]
- CMAC[3] = C_{LAST}[6]
- CMAC[4] = C_{LAST}[8]
- CMAC[5] = C_{LAST}[10]
- CMAC[6] = C_{LAST}[12]
- CMAC[7] = C_{LAST}[14]

c. New payload

The AES-CBC encryption will now be applied over P' = P || CMAC

9.5.4. AES-CBC encryption

a. Init Vectors

All operation are performed in CBC mode. The Init Vector is preserved among all operations on both sides. Both the PC and the RDR shall maintain 2 Init Vectors for the encryption/decryption:

- **IV_H is used by the PC to send** (encrypt), and by the RDR to receive (decrypt),
- **IV_R is used by the RDR to send** (encrypt), and by the PC to receive (decrypt).

Both IV_H and IV_R are cleared at the end of the authentication process (§ 9.4) and evolve independently afterwards.

b. Padding

The AES-CBC encryption could be performed only if the size of the text is a multiple of 16 bytes. A padding is always applied.

Let P' be the packet's payload (P' = P || CMAC as per § 9.5.3.c).

Construct P'' by adding zeros ($_h$00) to the right of P' until the specified length is reached:

- For Bulk-Out and Bulk-In messages, the length P'' is fixed 288 bytes (leading to a block of 289 bytes),
- For Interrupt message, the length of P'' is fixed to 32 bytes (leading to a block of 33 bytes).

c. Encryption

Compute C = E_{CBC} (K_{SESS} , IV_H or IV_R , P'')

(P'' encrypted in CBC mode, using K_{SESS} and using either the Init Vector of the sender)

The final block preserves the value of the ENDPOINT byte, but the block's content (including the actual length of the data) is crypted.

9.5.5. Receiving

When receiving secure block, the receiver must follow the reverse path:

1. Observe the ENDPOINT byte and check that the actual length of the block is equal to the expected length (289 or 33),
2. Retrieve $P'' = D_{CBC} (K_{SESS} , IV , C)$

   (remember to use the sender's Init Vector)
3. Extract the Length field in the recovered CCID header, and check that the value is between 0 and 262,
4. From the Length, locate the Padding field and verify that every byte in this field is $_h00$. Drop the padding to recover P' from P'',
5. Extract P and CMAC from P', verify the CMAC using K_{CMAC} and the sender's sequence number. Increment the sender's sequence number.

9.6. New authentication – Generation of a new session key

The PC may require a new authentication at any time, by sending a new SET CONFIGURATION command as specified in § 4.4 . During the authentication process, all features of the RDR are halted.

10. Configuration registers for a TCP coupler

10.1. Security options

Name	Tag	Description	Size
SEC	$_h$6E	Security option bits. See table a below	1

Security option bits

Bits	Value	Meaning
7	0	Telnet server is disabled
	1	Telnet server is enabled
6-0		*RFU (set to 0000000)*

Default value: $_b$10000000

10.2. Network configuration

10.2.1. IPv4 address, mask, and gateway

Name	Tag	Description	Size
IPA	$_h$80	IPv4 configuration bytes, see table below	4, 8 or 12

IPv4 configuration bytes

Bytes	Contains	Remark
0	Address, 1st byte	Device's IPv4 Address. If these bytes are missing, the default IP Address $_h$C0 A8 00 FA (192.168.0.250) is taken.
1	Address, 2nd byte	
2	Address, 3rd byte	
3	Address, 4th byte	
4	Mask, 1st byte	Network Mask. If these bytes are missing, the default Mask $_h$FF FF FF FF (255.255.255.0) is taken.
5	Mask, 2nd byte	
6	Mask, 3rd byte	
7	Mask, 4th byte	
8	Gateway, 1st byte	Default Gateway. If these bytes are missing, the value $_h$00 00 00 00 (0.0.0.0) is taken, meaning that there's no Gateway.
9	Gateway, 2nd byte	
10	Gateway, 3rd byte	
11	Gateway, 4th byte	

Default value: $_h$C0 A8 00 FA FF FF FF 00 00 00 00 00

(address = 192.168.0.250, mask = 255.255.255.0, no gateway)

10.2.2. TCP server port

Name	Tag	Description	Size
IPP	$_h$81	Listen TCP port for the server (2 bytes, MSB-first)	2

Default value: $_h$0F 9F (server TCP port = 3999)

10.2.3. Authentication and secure communication

Name	Tag	Description	Size
IPS	$_h$84	Server security settings bits, see table below	1

Security settings bits

Bits	Value	Meaning
7-2		*RFU (set to 000000)*
1-0	00	No authentication is required
	01	The authentication is mandatory, the secure communication is optional
	10	*RFU, do not use*
	11	Both the authentication and the secure communication are mandatory

Default value: $_b$00000000

10.2.4. Authentication Key

Name	Tag	Description	Size
IPK	$_h$85	K_{AUTH}	16

Default value: $_h$00 … $_h$00

10.2.5. Ethernet configuration

Name	Tag	Description	Size
ETC	$_h$8D	Ethernet configuration bits. See table a below	1

Ethernet configuration bits

Bits	Value	Meaning
7-1		*RFU (set to 0000000)*
0	0	Use auto-configuration (10/100Mbps, half or full-duplex)
	1	Force bitrate = 10Mbps, half-duplex

Default value: $_b$00000000

10.2.6. Info / Location

Name	Tag	Description	Size
ILI	$_h$8E	Info / Location string	Var. 0-30

Default value: empty

The **Info / Location** string is a text value (ASCII) that appears

- When someone tries to connect on Telnet,
- In the NDDU software.

Use this string as a reminder of where your RDR is installed, or what is its role in your access-control system.

10.2.7. Password for Telnet access

Name	Tag	Description	Size
ITP	$_h$8F	Password for Telnet access string	Var. 0-16

Default value: “springcard”

The **Password for Telnet access** string is a text value (ASCII) that protects the access to the RDR using Telnet protocol.

The password is mandatory. If this registry is not set, default value “springcard” is used.

Disclaimer

This document is provided for informational purposes only and shall not be construed as a commercial offer, a license, an advisory, fiduciary or professional relationship between SPRINGCARD and you. No information provided in this document shall be considered a substitute for your independent investigation.

The information provided in document may be related to products or services that are not available in your country.

This document is provided "as is" and without warranty of any kind to the extent allowed by the applicable law. While SPRINGCARD will use reasonable efforts to provide reliable information, we don't warrant that this document is free of inaccuracies, errors and/or omissions, or that its content is appropriate for your particular use or up to date. SPRINGCARD reserves the right to change the information at any time without notice.

SPRINGCARD doesn't warrant any results derived from the use of the products described in this document. SPRINGCARD will not be liable for any indirect, consequential or incidental damages, including but not limited to lost profits or revenues, business interruption, loss of data arising out of or in connection with the use, inability to use or reliance on any product (either hardware or software) described in this document.

These products are not designed for use in life support appliances, devices, or systems where malfunction of these product may result in personal injury. SPRINGCARD customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify SPRINGCARD for any damages resulting from such improper use or sale.

Copyright notice

Editor's information

SPRINGCARD SAS company with a capital of 227 000 €

RCS EVRY B 429 665 482

Parc Gutenberg, 2 voie La Cardon

91120 Palaiseau – FRANCE

Contact information

For more information and to locate our sales office or distributor in your country or area, please visit

www.springcard.com