CHERRY SECUREBOARD 1.0

Software Developer’s Guide

June 2019
Document Version: 0.4
Document State: Draft

Contents

About this Guide

This Developer Guide is for developers integrating ISO/IEC 7816 ICCs or ISO/IEC14443 PICCs using the CHERRY KC 1000 SC.

Contact

Please provide the following information about the device when you make an enquiry:

- Item and serial no. of the product
- Name and manufacturer of your system
- Operating system and, if applicable, installed service pack version

Cherry GmbH
Cherrystraße
91275 Auerbach/OPf.
Germany

Internet: www.cherry.de

1 Overview

The CHERRY SECUREBOARD 1.0 opens new market opportunities for system integrators seeking simple reader integration and development using standard interfaces, such as CCID (Circuit Card Interface Device). This reader generally works without installing or maintaining device drivers; only an operating system driver is necessary.

1.1 Features

- **ISO/IEC 7816-3** T=1, T=0 Interface
- **ISO/IEC 14443-3/4** T=Cl Contactless
- **Support for various NFC Tags** (e.g.: Mifare Classic, Mifare Ultralight, NTAG, Felica, ISO 15963 Tags, …)
- **Secure PIN Entry** according to CCID and PC/SC specification
- **True Plug and Play** without special driver installations
- **Randomized keyboard scan** to hardens against attackers aiming to capture key strokes using EMV signals sent bey the device during key scanning.
- **USB Keylogger Protection** by allowing to encapsulate HID messages within a TLS 1.3 channel protected by ECDSA Signatures, Client and Server Certificates, ECDH Key Exchange, CHACHA20POLY1305 Cipher.
- **Fully USB2.0 Compliant** including Low Power Consumption while the host is turned off. As well as Remote Wakeup features on Card Insertion and Keyboard activity.

2 Getting Started

Driver Installation: As stated no drivers need to be installed.

3 Host Interface

3.1 USB

CHERRY SECUREBOARD 1.0 supports USB 2.0 Full Speed (12 Mbits/s) interface.

The device enumerates as a composite device. USB protocol stack implements the following device classes:

- CCID (Integrated Circuit Cards Interface Device, v1.1).
- HID Keyboard.
- DFU (Device Firmware Update) – In DFU mode only.

3.1.1 Endpoints Assignment in Operational Mode

The table below lists the USB protocol stack endpoints, their parameters and functional assignment:

Endpoint	Description	Type	Max Packet Size
EP 0	Control Endpoint	Bulk IN/OUT	64
Interface 0: Keyboard			
EP 1	HID Reports	Interrupt IN	64
Interface 1: Media and Power Reports			
EP 2	HID Reports	Interrupt IN	64
Interface 2: Dual Function CCID Interface (Contact, Contactless)			
EP 3	ICC Data	Bulk IN	64
EP 3	ICC Data	Bulk OUT	64
EP 4	Card Presence	Interrupt IN	64
Interface 3: Secure Keyboard Reports			
EP 5	Encrypted Data	Bulk IN	64
EP 5	Encrypted Data	Bulk OUT	64

3.1.2 Endpoint Assignment in DFU Mode

Endpoint	Description	Type	Max Packet Size
EP 0	Control Endpoint	Bulk IN/OUT	64
Interface 0: DFU Interface			
	No Endpoints used		

4 Human Interface

4.1 Keyboard

The CHERRY SECURE BOARD 1.0 supports keyboard interface according to USB Device Class Definition for Human Interface Devices (HID Specification) version 1.11. It implements two USB protocol stack interfaces:
• Keyboard with BIOS support
• Media & Power Keys interface according to Advanced Configuration and Power Interface

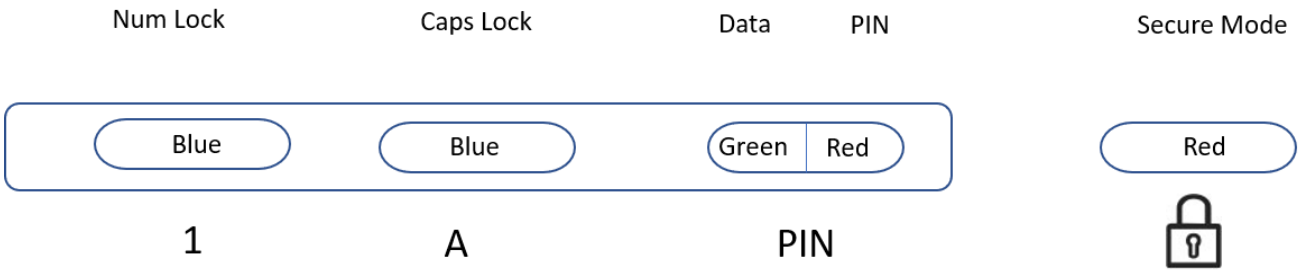
Specification (ACAPI)

4.2 LEDs

There are 5 LEDs to indicate the status of the keyboard, the card reader and the secure channel.

- Num Lock
- Caps Lock
- Data
- PIN
- Secure Mode

4.2.1 LED position and inscription



4.2.2 Num Lock

Indicates current function of numeric keypad. Its state is controlled by operating system.

4.2.3 Caps Lock

Indicates input mode in which all typed keys are uppercase. Its state is controlled by the operating system.

4.2.4 Data

Indicates the status of the card reader contact and contactless.

Function	**Green**
Contact card activated	on
Data transfer contact card	flickering
Reader is waiting for answer from card	slowly flashing (1Hz)
Data transfer contactless card	flickering

4.2.5 PIN

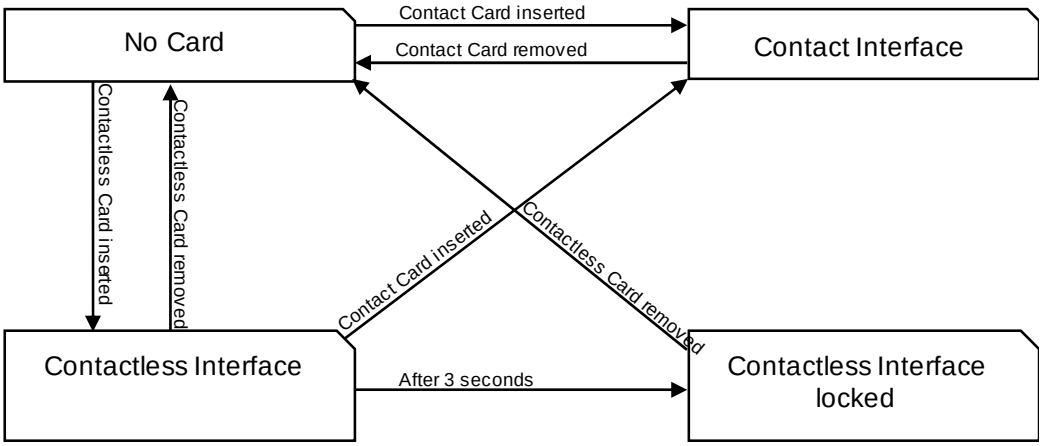
The PIN LED indicates that the secure PIN entry mode is on. In this mode user is expected to enter a PIN on the numeric keypad. All other keys are ignored. The keys pressed by the user on the numeric keypad are not reported to the PC.
The Secure PIN entry mode is started by one of secure pin commands. For more details see Secure Pin Entry Support chapter.
There is no possibility to control this LED other than starting secure pin entry session.

4.2.6 Secure Mode

Indicates the status of the secure channel between the keyboard and the system as described in section 9.1.

5 Dual Function CCID Interface

CHERRY SECUREBOARD 1.0 uses one logical interface for both contact and contactless communication. Only one ICC can be active at any time. The CCID selects the physical interface according to the following rules:



6 Contact Cards Interface

CHERRY SECUREBOARD 1.0 is compliant with CCID specification and features the following functionality:

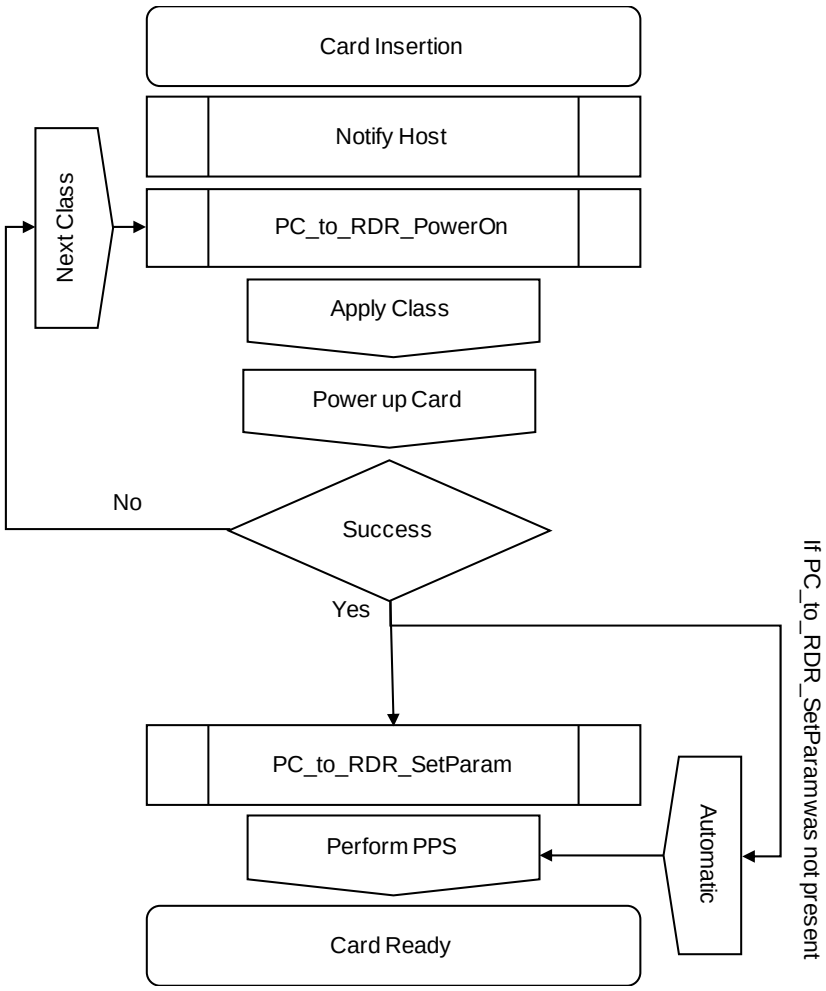
- Configurable Voltage Selection
- Extended APDU Exchange (T=1 Only)
- Secure Pin Entry

6.1 Card Activation

Before the card is ready for data exchange it must be properly activated. The diagram on the following page shows the activation sequence:

- Card is inserted into a contact slot.
- CCID notifies host
- CCID waits for PC_to_RDR_PowerOn
- CCID activates ICC according to configured voltage sequence
- CCID waits for PC_to_RDR_SetParameters
- CCID performs PPS
- Card is ready

If no *PC_to_RDR_SetParameters* was received before the first data packet the CCID selects proper parameters and performs PPS.



7 Contactless Interface

CHERRY SECUREBOARD 1.0 is compliant with CCID specification and features the following functionality:

- Extended APDU Exchange (T=1 Only)
- 14443-L4 exchange
- Various NFC Tags

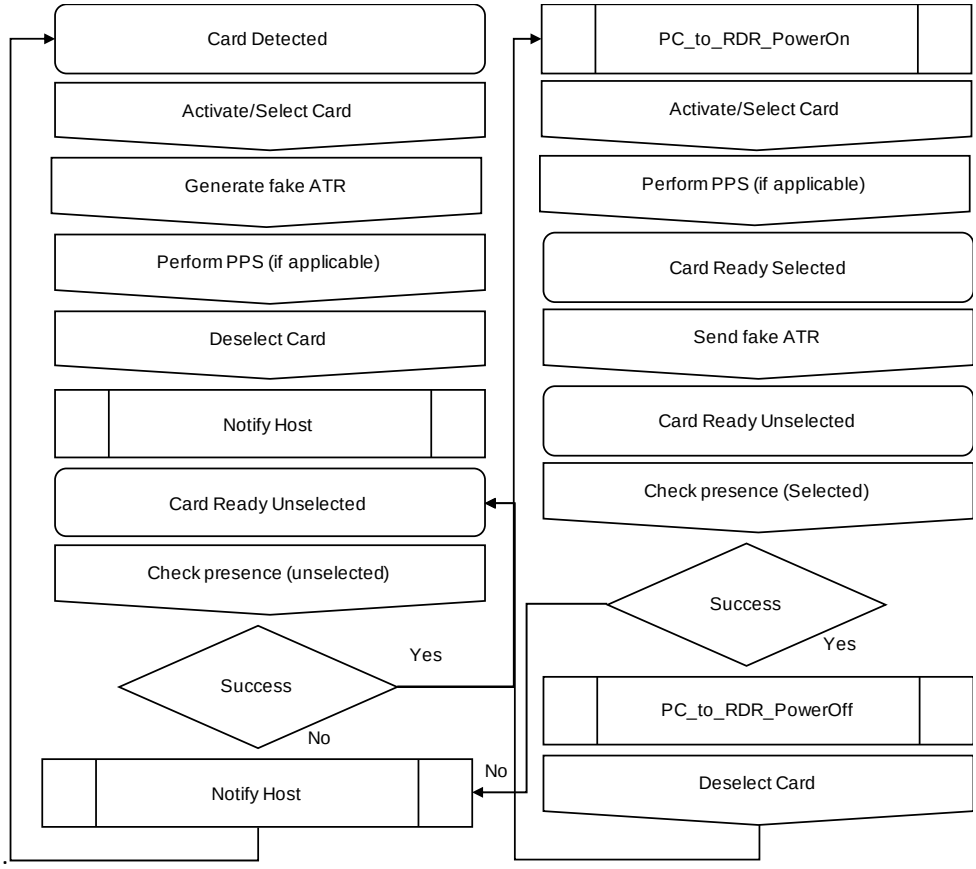
7.1 Card Activation

Before the card is ready for data exchange it must be properly activated.

The diagram on the following page shows the activation sequence:

- Card is detected in the Field.
- CCID activates, selects the PICC
- CCID generates ATR
- CCID notifies the host
- CCID polls the PICC to test if it is still in field
- CCID waits for PC_to_RDR_PowerOn
- CCID activates, selects the PICC
- CCID polls the PICC to test if it is still in field
- CCID waits for host commands
- On PC_to_RDR_PowerOn the ICC is deselected

All PICC Parameters are selected automatically by the CCID



7.2 Storage Cards

Storage Cards can be access as defined in PCSC Part3

7.2.1 Get Data

Return the UID (P1=0) or the Historical Bytes (P1=1) of the PICC

CLA	INS	P1	P2	Lc	Data In	Le
FFh	CAh	0/1	0	–	–	xx

Output:

Data Out
Data + SW1 SW2

Supported Cards:

Data + SW1 SW2

7.2.2 Read Binary

Read data from the PICC.

CLA	INS	P1	P2	Lc	Data In	Le
FFh	B0h	Address MSB	Address LSB	–	–	xx
FFh	B0h	0	0	Lc	Service, Blocks, Block Number[1]	xx

Output:

Data Out
Data + SW1 SW2

[1] Service and blocklist information are interpreted as common with Felica PICCs.

Supported Cards:

Mifare and similar, Felica, 15963 PICCs

Example Felica:

```
APDU, read MC:
  FF B0 00 00 06
    01            // 1 Service
    0B 00.        // Service
    01.           // 1 Block
    80 88         // Blocks
  00

Result:
  ff ff ff 00 ff 00 00 00
  00 00 00 00 00 00 00 00
  90 00
```

Example Mifare Classic

```
APDU read block 21
  FF B0 00 15 10

Result
  01 02 03 04 05 06 07 08
  09 0A 0B 0C 0D 0E 0F 15 90 00
```

7.2.3 Update Binary

Update data on the PICC.

CLA	INS	P1	P2	Lc	Data In	Le
FFh	D6h	Address MSB	Address LSB	–	Data to write	–
FFh	D6h	0	0	Lc	Service, Blocks, Block Number, Data[2]	Le

Output:

Data Out
SW1 SW2

[2] Service and blocklist information are interpreted as common with Felica PICCs.

Supported Cards:

Mifare and similar, Felica, 15963 PICCs

Example Felica:

```
APDU, update SPAD 10:
  FF D6 00 00 16
    01            // 1 Service
    09 00         // Service
    01            // 1 Block
    80 0a         // Block
    // Data to Write
    01 02 03 04 05 06 07 08
    09 0A 0B 0C 0D 0E 0F 15
  00

Result:
  90 00
```

Example Mifare Classic

```
APDU update block 21
  FF D6 00 15 10
    01 02 03 04 05 06 07 08
    09 0A 0B 0C 0D 0E 0F 15

Result
  90 00
```

7.2.4 Load Key

Load a Key into the readers volatile or non-volatile memory (see details in PCSC Part 3).

CLA	INS	P1	P2	Lc	Data In	Le
FFh	82h	Key Structure	Key ID	xx	key	–

Output:

Data Out
SW1 SW2

Supported Cards:

Mifare Classic

Example Mifare Classic

```
APDU Load non-volatile key slot 0
  FF82 2000 06 xxxxxxxxxxxx
Result
```

```
  90 00

APDU Load non-volatile key slot 1
  FF82 2001 06 xxxxxxxxxxxx
Result
  90 00

APDU Load volatile key slot 80h
  FF82 0080 06 xxxxxxxxxxxx
Result
  90 00
```

7.2.5 Key Store

SECUREBORAD1.0 maintain a key store for 3 keys.

Slot ID	Technology	Key Size	Memory Type
00h	Mifare Classic	6 Bytes	non-volatile
01h	Mifare Classic	6 Bytes	non-volatile
80h	Mifare Classic	6 Bytes	volatile

7.2.6 General Authenticate

Authenticate with a key (see details in PCSC Part 3).

CLA	INS	P1	P2	Lc	Data In	Le
FFh	86h	0	0	5	Data	–

Data

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
01	Address MSB	Address LSB	KEY_A 60h or KEY_B 61h	Key Number (Slot ID)

Output:

Data Out
SW1 SW2

Supported Cards:

Mifare Classic

Example Mifare Classic

```
APDU Authenticate Block 14h with KEY_A stored in Slot 1.
  FF 86 00 00 05
    01 00 14 60 01
Result
   90 00
```

7.2.7 Transparent Exchange

Issue a low-level command to a PICC

CLA	INS	P1	P2	Lc	Data In	Le
FFh	C2	0	1	Lc	Data Object	Le

See PCSC Part 3 Supplement 2 for details. Note that all data objects need to be DER encoded (PCSC Part 3 state BER).

Supported Data Objects: TIMER, TRANSCEIVE

Output:

Data Out
data SW1 SW2

Supported Cards:

Mifare Ultralight and similar

Example Mifare Classic

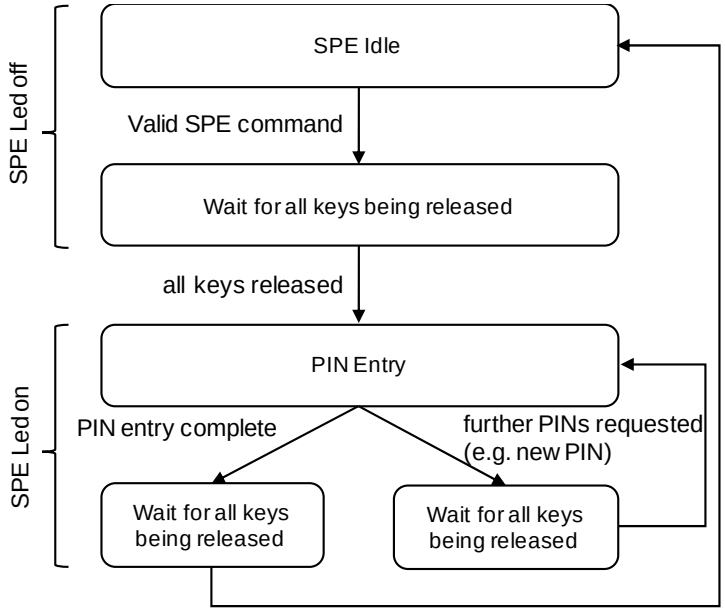
```
APDU, GET_VERSION + Delay 310ms:
  FF C2 00 01 0A            // CLA INS P1 P2 Lc
     5F 46 04 F0 BA 04 00   // TIMER Data Object 310000us
     95 01 60               // Transceive 60h
  00                        // Le

Result
  C0 03 00 90 00            // Success Data Object
  92 01 00                  //.Bit Framing Data Object
  96 02 00 00               // Return Status Data Object
  97 08 00 04 04 02 01 00 13 03
                            // Response Data Data Object
  90 00                     // SW1 SW2
```

8 Secure Pin Entry Support

CHERRY SECUREBOARD1.0 implements a method of secure PIN entry called SPE. This feature uses PCSC specification: Part 10. IFDs with Secure PIN Entry Capabilities (PCSC-10-SPE). SPE is available only when an ISO/IEC7816 Contact ICC is inserted in the card slot (There is no sense in keeping the pin within the device in order to transmit it on radio).

During SPE the following state machine is processed:



8.1 Feature Execution by Pseudo-APDU

SPE is commanded by special APDUs, called Pseudo APDU (PPDU). The Pseudo-APDU command is in a data format which has much resemblance with an APDU for cards:

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	C2h	01h	*Feature Number*	Lc	*Feature Command Data*	Le

Any valid Pseudo-APDU command will always generate a response:

Response Data	Response Status SW1/SW2	Description
Feature Response Data	90 00	Feature executed successful; Feature Response Data is present
-empty-	6A 86	Incorrect value for P2 (requested feature not present)

The list of Feature Numbers and Feature Command Data implemented by the CHERRY KC 1000 SC is defined in the following table:

Feature	Feature Number	Feature Command Data	Feature Response Data
GET_FEATURE_REQUEST	0x00	-empty-	byte array: each byte in this array represents a feature number present in this reader
FEATURE_VERIFY_PIN_START	0x01	PIN_VERIFY structure (see below)	-empty-
FEATURE_VERIFY_PIN_FINISH	0x02	-empty-	2-byte status
FEATURE_MODIFY_PIN_START	0x03	PIN_MODIFY structure (see below)	-empty-
FEATURE_MODIFY_PIN_FINISH	0x04	-empty-	2-byte status
FEATURE_GET_KEY_PRESSED	0x05	-empty-	single byte with the following code: 0x2B – 0-9 valid key 0x1B – cancel 0x08 – backspace 0x0d – enter 0x0e – timeout 0x00 – no key 0x40 – aborted
FEATURE_VERIFY_PIN_DIRECT	0x06	PIN_VERIFY structure (see below)	2-byte status
FEATURE_MODIFY_PIN_DIRECT	0x07	PIN_MODIFY structure (see below)	2-byte status
FEATURE_IFD_PIN_PROPERTIES	0x0A	-empty-	PIN_PROPERTIES structure (see below)

During active SPE transactions (indicated by the PIN LED being LIT/ON) only the following features are allowed:

- FEATURE_VERIFY_PIN_FINISH
- FEATURE_MODIFY_PIN_FINISH
- FEATURE_GET_KEY_PRESSED

PIN_VERIFY structure as defined in PCSC Part 10 specification (Section 2.5.2):

Byte Offset	Field	Type	Description
0	bTimeOut	BYTE	timeout in seconds (00 means use default timeout)

Byte Offset	Field	Type	Description
1	bTimeOut2	BYTE	timeout in seconds after first key stroke
2	bmFormatString	BYTE	formatting options USB_CCID_PIN_FORMAT_xxx
3	bmPINBlockString	BYTE	bits 7-4 bit size of PIN length in APDU bits 3-0 PIN block size in bytes after justification and formatting
4	bmPINLengthFormat	BYTE	bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits, bits 3-0 PIN length position in system units
5	wPINMaxExtraDigit	USHORT	XXYY, where XX is minimum PIN size in digits, YY is maximum
7	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete: 01h Max size reached 02h Validation key pressed 04h Timeout occurred
8	bNumberMessage	BYTE	Number of messages to display for PIN verification
9	wLangId	USHORT	Language for messages
11	bMsgIndex	BYTE	Message index (should be 00)
12	bTeoPrologue	BYTE[3]	T=1 I-block prologue field to use (fill with 00)
15	ulDataLength	ULONG	length of Data to be sent to the ICC
19	abData	BYTE[]	Data to send to the ICC

PIN_MODIFY structure as defined in PCSC Part 10 specification (section 2.5.3):

Byte Offset	Field	Type	Description
0	bTimeOut	BYTE	timeout in seconds (00 means use default timeout)
1	bTimeOut2	BYTE	timeout in seconds after first key stroke
2	bmFormatString	BYTE	formatting options USB_CCID_PIN_FORMAT_xxx
3	bmPINBlockString	BYTE	bits 7-4 bit size of PIN length in APDU bits 3-0 PIN block size in bytes after justification and

Byte Offset	Field	Type	Description
			formatting
4	bmPINLengthFormat	BYTE	bits 7-5 RFU, bit 4 set if system units are bytes clear if system units are bits, bits 3-0 PIN length position in system units
5	bInsertionOffsetOld	BYTE	Insertion position offset in bytes for the current PIN
6	bInsertionOffsetNew	BYTE	Insertion position offset in bytes for the new PIN
7	wPINMaxExtraDigit	USHORT	XXYY, where XX is minimum PIN size in digits, YY is maximum
9	bConfirmPIN	BYTE	Flags governing need for confirmation of new PIN
10	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete: 01h Max size reached 02h Validation key pressed 04h Timeout occurred
11	bNumberMessage	BYTE	Number of messages to display for PIN verification
12	wLangId	USHORT	Language for messages
14	bMsgIndex1	BYTE	Index of 1st prompting message
15	bMsgIndex2	BYTE	Index of 2nd prompting message
16	bMsgIndex3	BYTE	Index of 3rd prompting message
17	bTeoPrologue	BYTE[3]	T=1 I-block prologue field to use (fill with 00)
20	ulDataLength	ULONG	length of Data to be sent to the ICC
24	abData	BYTE[]	Data to send to the ICC

PIN_PROPERTIES structure as defined in PCSC Part 10 specification (section 2.5.5):

Byte Offset	Field	Type	Description
0	wLcdLayout	USHORT	Display characteristics 0000h = No LCD
2	bEntryValidationCondition	BYTE	Conditions under which PIN entry should be considered complete:

Byte Offset	Field	Type	Description
			01h = Max size reached 02h = Validation key pressed 04h = Timeout occurred
3	bTimeOut2	BYTE	0 = IFD does not distinguish bTimeOut from bTimeOut2 1 = IFD distinguishes bTimeOut from bTimeOut2

GET_KEY structure as defined in PCSC Part 10 specification (paragraph 2.5.11):

Byte Offset	Field	Type	Description
0	wWaitTime	USHORT	Time in seconds to wait for a key to be hit
2	bMode	BYTE	Display mode of key N/A, set 00h
3	bPosX	BYTE	Column (starting at 0) N/A, set 00h
4	bPosY	BYTE	Row (starting at 0) N/A, set 00h

8.2 SPE Examples

8.2.1 Get Feature Request

P-APDU (GET_FEATURE_REQUEST):

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	C2h	01h	0x00	0x00	*-empty-*	*-empty-*

Response:

Response Data	Response Status SW1/SW2	Description
01 02 03 04 05 06 07 0A 10	90 00	The list of features

8.2.2 Get PIN_PROPERTIES Structure

P-APDU (FEATURE_IFD_PIN_PROPERTIES):

CLA	INS	P1	P2	Lc	Data Field	Le
FFh	C2	01	0A	00	*-empty-*	*-empty-*

Response:

Response Data	Response Status SW1/SW2	Description
00 00 07 01	90 00	PIN_PROPERTIES structure: wLcdLayout = 0x0000 (no display supported) bEntryValidationCondition = 0x07 (all validation condition supported) bTimeOut2 = 0x01 (IFD distinguishes bTimeOut from bTimeOut2)

8.2.3 Direct PIN Verification

P-APDU (FEATURE_VERIFY_PIN_DIRECT):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	06	21	0A 05 46 08 00 08 04 06 FF 00 00 00 00 00 00 0E 00 00 00 00 20 00 00 09 FF FF FF FF FF FF FF FF PIN_VERIFY structure: bTimeOut=0A bTimeOut2=05 bmFormatString=46 bmPINBlockString=08 bmPINLengthFormat=00 wPINMaxExtraDigit=0408 bEntryValidationCondition=06 bNumberMessage=FF wLangId=0000 bMsgIndex=00 bTeoPrologue=000000 ulDataLength=0000000E abData=0020000009FFFFFFFFFFFFFFFF	00

	FF

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN verification OK

8.2.4 Direct Pin Modification

P-APDU (FEATURE_MODIFY_PIN_DIRECT):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	07	29	0A 0A 01 0C 00 00 00 04 04 00 02 FF 00 00 00 00 00 00 00 00 11 00 00 00 00 20 00 00 0C FF FF FF FF FF FF FF FF FF FF FF PIN_MODIFY structure: bTimeOut=0A bTimeOut2=0A bmFormatString=01 bmPINBlockString=0C bmPINLengthFormat=00 bInsertionOffsetOld=00 bInsertionOffsetNew=00 wPINMaxExtraDigit=0404 bConfirmPIN=00 bEntryValidationCondition=02 bNumberMessage=FF wLangId=0000 bMsgIndex1=00 bMsgIndex2=00 bMsgIndex3=00 bTeoPrologue[3]=000000 ulDataLength=00000011 abData=002000000CFFFFFFFFFFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN modification OK

8.2.5 Indirect PIN Verification

P-APDU (FEATURE_VERIFY_PIN_START):

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	01	21	0A 05 46 08 00 08 04 06 FF 00 00 00 00 00 00 0E 00 00 00 00 20 00 00 09 FF FF FF FF FF FF FF FF PIN_VERIFY structure: bTimeOut=0A bTimeOut2=05 bmFormatString=46 bmPINBlockString=08 bmPINLengthFormat=00 wPINMaxExtraDigit=0408 bEntryValidationCondition=06 bNumberMessage=FF wLangId=0000 bMsgIndex=00 bTeoPrologue=000000 ulDataLength=0000000E abData=0020000009FFFFFFFFFFFFFFFF	00

Response:

Response Data	Response Status SW1/SW2	Description
-empty-	90 00	

P-APDU (FEATURE_GET_KEY_PRESSED):

This P-APDU should be repeated until one of the termination codes has been returned.

CLA	INS	P1	P2	Lc	Data Field	Le

FF	C2	01	01	21	-empty-	-empty-

Response:

Response Data	**Response Status SW1/SW2**	**Description**
2B	90 00	Valid key pressed

P-APDU (FEATURE_VERIFY_PIN_FINISH):

CLA	**INS**	**P1**	**P2**	**Lc**	**Data Field**	**Le**
FF	C2	01	02	00	-empty-	-empty-

Response:

Response Data	**Response Status SW1/SW2**	**Description**
90 00	90 00	PIN verification ok

8.2.6 Indirect PIN Modification

P-APDU (FEATURE_MODIFY_PIN_START):

CLA	**INS**	**P1**	**P2**	**Lc**	**Data Field**	**Le**
FF	C2	01	03	29	0A 0A 01 0C 00 00 00 04 04 00 02 FF 00 00 00 00 00 00 00 00 11 00 00 00 00 20 00 00 0C FF FF FF FF FF FF FF FF FF FF FF PIN_MODIFY structure: bTimeOut=0A bTimeOut2=0A bmFormatString=01 bmPINBlockString=0C bmPINLengthFormat=00 bInsertionOffsetOld=00 bInsertionOffsetNew=00 wPINMaxExtraDigit=0404 bConfirmPIN=00 bEntryValidationCondition=02 bNumberMessage=FF	00

	wLangId=0000 bMsgIndex1=00 bMsgIndex2=00 bMsgIndex3=00 bTeoPrologue[3]=000000 ulDataLength=00000011 abData=002000000CFFFFFFFFFFFFFFFF FFFFFFFF

Response:

Response Data	Response Status SW1/SW2	Description
-empty-	90 00	

P-APDU (FEATURE_GET_KEY_PRESSED):

This P-APDU should be repeated until one of the termination codes has been returned.

CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	01	21	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
2B	90 00	Valid key pressed

P-APDU (FEATURE_MODIFY_PIN_FINISH):

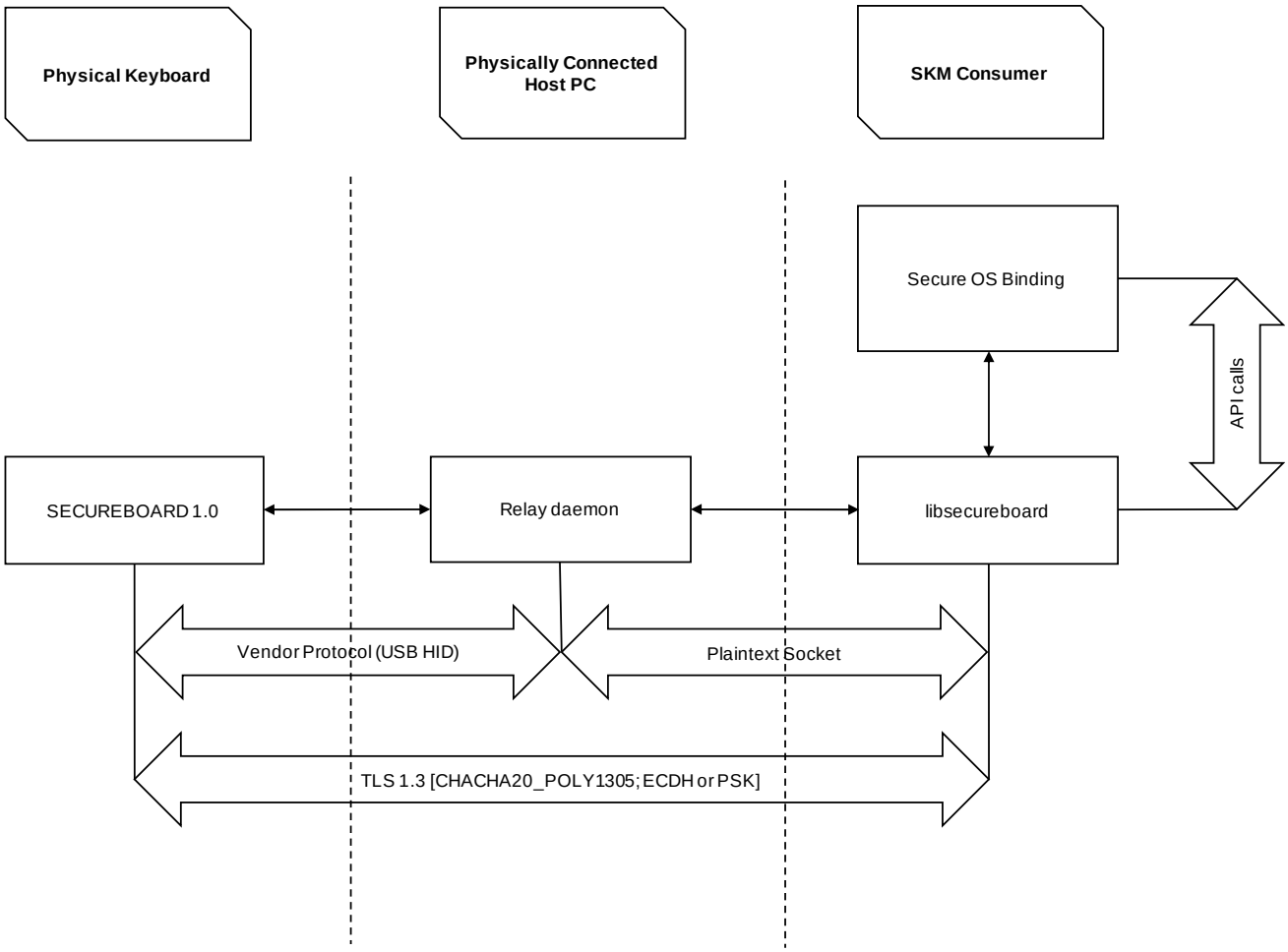
CLA	INS	P1	P2	Lc	Data Field	Le
FF	C2	01	02	00	-empty-	-empty-

Response:

Response Data	Response Status SW1/SW2	Description
90 00	90 00	PIN verification ok

9 Secure Keyboard Mode

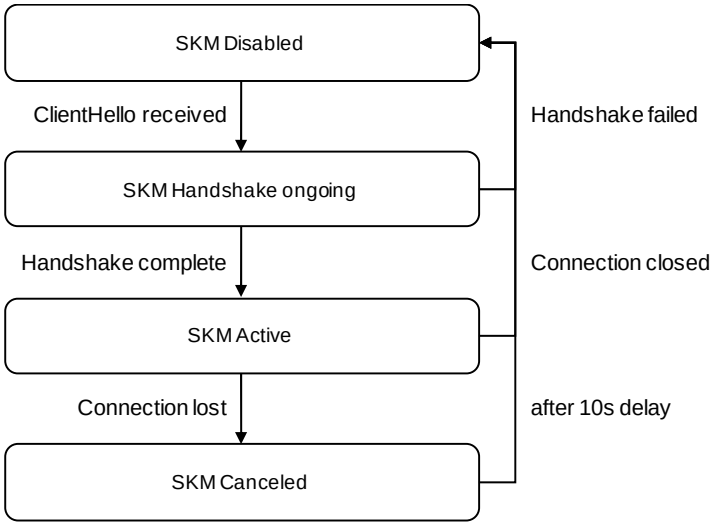
SECUREBOARD 1.0 allow to enter a Secure Keyboard Mode. In Secure keyboard mode all key strokes are transmitted securely using a TLS 1.3 channel to the host. The components are depicted in the chart below.



- ***SECUREBOARD 1.0*** implements the necessary subset TLS 1.3 of functions required by a TLS 1.3 server (RFC 8446). TLS Records are encapsulated within USB HID Vendor Reports. The following subset is implemented:
  - TLS_CHACHA20_POLY1305_SHA256.
  - Handshake using ECDSA/ECDH over NIST Curve P-256.
  - Handshake using PSK.
  - Client certificate authentication with user defined CA.
  - Static Factory (unique per SECUREBOARD 1.0) device certificate.
  - User Server Certificate provisioning option.
- ***Relay Daemon*** connects the SECUREBOARD 1.0 to a network socket. It manages physical devices and relays records between SECUREBOARD 1.0 and Secure OS Binding. As depicted above it does neither de- nor encrypt data for the TLS 1.3 connection.
- ***libsecureboard*** is the connecting part for consumer software. It uses openssl to establish a TLS 1.3 connection with the SECUREBOARD 1.0 and provides API calls for communication protocol encapsulated within the TLS 1.3 channel. The library documentation is provided within the source release.
- ***Secure OS Binding*** is operating system dependent and needs to be designed by the system integrator. This piece of software uses libsecureboard to communicate with a SECUREBOARD 1.0 and calls appropriate OS functions to feed keyboard events into the OS. An example is provided as part of libsecureboard.

9.1 Secure Keyboard Mode Led

The Secure Keyboard Mode (SKM) Led is used to provide an indication of the SKM state to the user. The following chart depicts the possible state. The table below describes the Led indications.



LED State	State	Description
off	SKM Disabled	SECUREBOARD 1.0 operates in normal mode. Key strokes are delivered insecurely using classic HID protocol
on	SKM Active	SECURECOARD1.0 operates in secure mode. Key strokes are delivered securely within the TLS 1.3 channel.
flashing 1Hz	SKM Handshake	SECUREBOARD 1.0 is currently executing an TLS 1.3 handshake. No Key strokes are transmitted at all
Flashing 20Hz	SKM Canceled	The TLS 1.3 channel has been canceled (e.g. connection loss with libsecureboard). This state is used to indicate that the SECUREBOARD 1.0 is disabling the secure connection in 10 seconds.

9.2 Certificate Store

The device stores the following items in a certificate store:

Item	changeable	default	Format	Purpose
Device Certificate	no	unique certificate issued during production	DER encoded x509 certificate	Server Certificate

Device Private Key	no	unique private key created during production	Key over curve NIST P-256	Server Certificate
User Certificate	yes	empty	DER encoded x509 certificate	Server Certificate
User Private Key	yes	empty	PEM encoded key over curve NIST P-256	Server Certificate
User Root CA	yes	empty	SHA256 Fingerprint of the Root CA	Verify Client Certificates

User certificates can be provisioned using sb-tool described in Section 9.4 or by using libsecureboard (see sections C.1.5, C.1.6 and C.1.7). Once an item is provisioned it can only be changed after resetting the complete key-store. For security reasons this action requires physical access to the device. The procedure is described in Section 9.2.1.

9.2.1 Resetting the Certificate Store

To reset the Certificate Store, one requires physical access to the device.

The procedure is as follows:

- Disconnect the SECUREBOARD 1.0
- While Reconnecting SECUREBOARD 1.0 hold:

  D+J+RGUI *Note:* RGUI *denotes the right Windows key*

- On success all LEDs flash (>10Hz) for about 1 second, the keyboard starts normally, and the certificate store is reset to the state described in the table above.

9.3 Relay Daemon (sb-relayd)

Note: The relay-daemon is currently under development and the API might change. Refer to the documentation of the relay-daemon itself to watch for changes.

The latest state of the software can be downloaded at:

https://github.com/secureboard10/libsecureboard

sb-relayd is a service the relays TLS1.3 records between SECUREBOARD 1.0 and *libsecureboard*. It needs to be started **after** a SECUREBOARD 1.0 is connected. The corresponding *systemd* service and *udev* rule flies are distributed along with *sb-relayd*. *sb-relayd* is part of libsecureboard.

The following files need to be installed at the proper locations to work with *systemd/udev*:

```
/sbin/sb-relayd
/etc/sb-relayd.conf
/etc/udev/rules.d/95-secureboard.rules
/lib/systemd/system/sb-relayd@.service
```

After installing the files and reload the rules with:

```
sudo systemctl daemon-reload
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Once completed, after reconnecting a SECUREBOARD1.0, the a service should show up. To verify execute:

```
# sudo systemctl | grep SECUREBOARD
sb-relayd@00000002JS0405948N5LI0OTHA.service   loaded active running   "SECUREBOARD1.0
```

```
Relay Daemon 00000002JS0405948N5LI0OTHA"
```

To verify the socket name execute:

```
$ netstat -lnp | grep SECUREBOARD
unix 2 [ ACC ] STREAM LISTENING 42128 - @SECUREBOARD1.0-00000002JS0405948N5LI0OTHA
```

Once the relay daemon is running, the secure keyboard mode can be activated with (see <path-to>/secureboard -h for further options):

```
sudo <path-to>/sb-tool --ca-dir <path-to-ca> -u @SECUREBOARD1.0-<device-serial>
```

or, to enable SKM Linux OS binding, by:

```
sudo <path-to>/sb-tool --ca-dir <path-to-ca> -u @SECUREBOARD1.0-<device-serial> --evdev
```

9.4 libsecureboard example software

Note: The libsecureboard is currently under development and the API might change. Refer to the documentation of the libsecureboard itself to watch for changes.

The latest state of the software can be downloaded at:

https://github.com/secureboard10/libsecureboard

Along with libsecureboard, also a sample application is provided. This application can further be used to establish a TLS 1.3 channel and to personalize the device.

```
SYNOPSIS: sb-tooltool <options>
        Connection Management
                -c <host>:<port>      (optopt)   Host and Port of the relay-daemon
                -u <unix-socket>      (opt)   Unix Socket path (use @ as first character
                                              for abstract namespace)
                                     Note: either -c or -u must be given
                --ca-dir <ca-dir>     (req)   Directory to CA used to verify certificates
                -s <sessionfile>      (opt)   Session file uses to store PSK key
                --sni <servername>    (opt)   Use <servername> as server name indication in
                                              ClientHello message
                --client-cert <cert>  (opt)   Use <cert> as client certificate if requested
                                              (PEM encoded)
                                              If used -client-key must also be present
                --client-key <key>    (opt)   Use <key> as client key if requested (PEM
                                              encoded)
                                              If used -client-cert must also be present
                --evdev               (opt)   Send Reports to evdev
        Device Personalization **ENSURE CORRECT ENCODINGS**
                --user-cert           (opt)   Update User Certificate and terminate (DER
                                              encoded)
                --user-priv-key       (opt)   Update User Private Key (must match User
                                              Certificate) (PEM encoded)
                --user-root-ca        (opt)   Update User Root Certificate Authority
                                              (PEM encoded)
        Other Options
                --verbose, -v         (opt)   Be verbose
```

When neither *--user-cert, --user-priv-key* nor *--user-root-ca* is present *sb-tool* acts in *Virtual Keyboard Mode* and terminates when the user hits *^C*, or *^D*. This mode is only available after Device Personalization (see Section 9.5) and if not connecting using --sni.

In *Virtual Keyboard Mode* received reports are either

- printed to the console (if *--evdev* is absent) or
- connected to virtual input device (if *--evdev* is present).

The *--sni* can be used to select the identity selected by the SECUREBOARD 1.0. If *--sni* is not used the SECUREBOARD 1.0

prefers the *User Certificate* over the *Device Certificate* (see Section 9.5). If used and the *<servername>* is the device serial, the SECUREBOARD 1.0 uses the identity provisioned during production.

In any case, when a SECUREBOARD 1.0 identifies itself with the *Device Certificate* the *Virtual Keyboard Mode* is not available since SECUREBOARD 1.0 will not send any keyboard reports.

9.5 Device Personalization

Before using the Secure Keyboard Mode, SECUREBOARD1.0 must be personalized with your company's device certificate. Ensure to do the following steps in a secure environment.

As long as the device is not personalized, the SECUREBOARD1.0 will not send any key strokes in *Secure Keyboard Mode* to the host connected. Basically, the command *Set Running* (see Appendix C.1.4) is disabled for un-personalized devices.

Note 1: All Certificates and Key Pairs must be X509 Version 3 *using* ECDSA *over* NIST Curve prime256v1 *with corresponding keys.*

9.5.1 Create a User Certificate Authority

```
export CADIR=<path-to-your-ca>
export KEYDIR=<path-to-a-secure-storage>
export CONFIGFILE=<path-to-openssl.cnf>
export USER_DEVICE_ROOT_CA=device_root_ca
cd $CADIR
openssl req -days 3650 -config $CONFIGFILE -sha256 -new -x509 \
   -newkey ec:<(openssl ecparam -name prime256v1) \
   -subj "/C=AT/O=User Company/CN=SB Root" \
   -keyout $KEYDIR/$USER_DEVICE_ROOT_CA-key.pem -out $USER_DEVICE_ROOT_CA.pem
c_rehash $CADIR
```

Keep the generated key safe and secure. It is used to sign the certificates provisioned in your SECUREBOARDS 1.0. The certificate should be stored at a location available to users, it does not contain secret information.

9.5.2 Create a User Certificate

```
export TMPDIR=/tmp
export USER_DEVICE=device01
cd $TMPDIR
openssl req -config $CONFIGFILE -sha256 -new \
    -newkey ec:<(openssl ecparam -name prime256v1) \
    -subj "/C=AT/O=User Company/CN=$USER_DEVICE" \
    -keyout $USER_DEVICE-key.pem -out $USER_DEVICE-csr.pem
openssl x509 -sha256 -days 3650 -req -in $USER_DEVICE-csr.pem \
    -CA $CADIR/$USER_DEVICE_ROOT_CA.pem \
    -CAkey $KEYDIR/$USER_DEVICE_ROOT_CA-key.pem \
    -extfile $(dirname $CONFIGFILE)/production.ext \
    -CAcreateserial -out $USER_DEVICE.der --outform DER
# certificate signing request is no longer required
rm $TMPDIR/$USER_DEVICE-csr.pem
```

Verify the size of *$USER_DEVICE.der*. **It must be less than 572 bytes**. This should not be a to high restriction unless you use a too long subject. Use the following command to verify the size:

9.5.3 Device Genuineness

During production each SECUREBOARD 1.0 is initialized with a unique certificate and key pair signed by the Secureboard CA. When connecting to the SECUREBOARD 1.0 the client can verify the keychain to prove genuineness. The certificates can be downloaded at:

https://github.com/secureboard10/secureboard-ca

Once downloaded execute

```
c_rehash <path-to-secureboard-ca>
```

to initialize the symlinks for openssl.

As anchor of trust the fingerprints of the root certificate is depicted below:

```
$ openssl x509 -noout -fingerprint -sha256 -inform pem -in SecureboardRootCA.pem
SHA256 Fingerprint=2E:1E:CB:35:76:EF:D4:AF:77:0C:91:0B:C3:48:00:9B: \
                   F7:BF:E2:1C:DB:EC:41:08:8D:6B:28:94:13:6C:38:BE
$ openssl x509 -noout -fingerprint -sha1 -inform pem -in SecureboardRootCA.pem
SHA1 Fingerprint=88:A0:59:94:FD:E8:EE:D0:C9:24:EA:A0:F1:F5:01:24:64:E2:B9:D1
```

9.5.4 Provision the User Certificate and User Key

Preconditions:

- Before running this step ensure that sb-relayd is installed and running (see Section 9.3).
- Before running this step fetch the *secureboard_ca*. Execute *c_rehash* within the *secureboard_ca* to create proper openssl symbolic links.

```
<path-to-sb-tool>  \
   -u @SECUREBOARD1.0-<device-serial> \
   --ca-dir <path-to-secureboard-ca> \
   --user-cert $TMPDIR/$USER_DEVICE.der \
   --user-priv-key $TMPDIR/$USER_DEVICE-key.pem
# certificate and key are no longer required
rm $TMPDIR/$USER_DEVICE.der $TMPDIR/$USER_DEVICE-key.pem
```

This steps also verifies that the device can sign data using a genuine key pair. After this step the SECUREBOARD1.0 uses the new certificate when establishing a TLS1.3 channel.

9.5.5 Provision the Client Root CA (optional, but highly recommended)

The steps above guarantee that, later, when you establish a connection to the SECUREBOARD 1.0 you are indeed connecting to a SECUREBOARD 1.0 and that the data received is genuine. However, the SECUREBOARD 1.0 has no indication that you are allowed to receive data. Without this step, anyone can connect to the SECUREBOARD 1.0 and the SECUREBOARD 1.0 is happy to send input data to anyone.

To mitigate this behavior SECUREBOARD 1.0 provide authentication through client certificates. To enable this feature, you must provision the fingerprint of the User Root CA of which your authorized client certificates are signed with.

Note, the example below already uses the CA you provisioned with the steps before.

```
<path-to-sb-tool>  \
   -u @SECUREBOARD1.0-<device-serial> \
   --ca-dir $CADIR \
   --user-root-ca $CADIR/$USER_DEVICE_ROOT_CA.pem
```

From now on you can only connect to the SECUREBOARD 1.0 using a certificate and key signed with this CA. The User Root CA can be the same as the User Certificate Authority (see Section 9.5.1), but it can also be a new one.

Note 1: Check the size of the certificate in DER encoded form with:

```
openssl x509 -in $CADIR/$USER_DEVICE_ROOT_CA.pem -inform PEM -outform DER | wc -c
```

*It must be **less than 475 bytes** (see Section 9.5.6).*

9.5.6 Create a Client Certificate (optional, but highly recommended)

```
export CLIENT_NAME=bob
cd $CADIR
openssl req -config $CONFIGFILE -sha256 -new \
   -newkey ec:<(openssl ecparam -name prime256v1) \
   -subj "/C=AT/O=User Company/CN=$CLIENT_NAME" \
   -keyout $KEYDIR/$CLIENT_NAME-key.pem -out $CLIENT_NAME-csr.pem
openssl x509 -sha256 -days 365 -req \
   -in $CLIENT_NAME-csr.pem -CA $CADIR/$USER_DEVICE_ROOT_CA.pem \
   -CAkey $KEYDIR/$USER_DEVICE_ROOT_CA-key.pem \
   -extfile $(dirname $CONFIGFILE)/device.ext \
   -CAcreateserial -out $CADIR/$CLIENT_NAME.pem
rm $CLIENT_NAME-csr.pem
c_rehash $CADIR
```

To use the client certificate along with sb-tool, add the following parameters:

```
<path-to-sb-tool> […] --client-cert $CADIR/bob.pem --client-key $KEYDIR/bob-key.pem
```

Note 1: When establishing a TLS1.3 channel the client sends all certificates (DER encoded) in the chain to the SECUREBOARD 1.0 in which the memory is constraint. SECUREBOARD 1.0 temporarily allocates 475 bytes of memory for each certificate. All certificates in the client chain must be ***smaller than 475 bytes when encoded in DER****.*

```
openssl x509 -in $CADIR/$CLIENT_NAME.pem -inform PEM -outform DER | wc -c
```

Note 2: The length of the chain is basically unrestricted. But take into consideration that the SECUREBOARD 1.0, must verify a signature per item in the chain. Which takes about 2 seconds per signature. When connecting using a PSK signature computation is omitted.

9.6 Security Considerations

SECUREBOARD 1.0 implements a resource constrained TLS1.3 server. While after establishing a connection, data is transmitted is encrypted according to RFC8446, restricting access to the SECUREBOARD 1.0 Secure Keyboard Mode interface is out of scope of this product.

Especially when dealing with sockets created by sb-relayd, other measures like *operating systems permissions, firewall, VPN, ssh tunnels* need to be taken further restrict external access to the interface.

It is further important to provision a the Client Root CA, since this counters that unauthorized personal can turn on the SKM Indicator LED.

9.7 Secure Keyboard Mode TLS 1.3 details.

SECUREBOARD 1.0 implements the following messages:

9.7.1 ClientHello

- The *ClientHello* message is structured as defined in RFC8446.. The following restrictions apply:
- The total message size must be less than 475 bytes.
- The cipher suite TLS_CHACHA20_POLY1305_SHA256 must be offered
- If no PSK is provided the Key Share extension must be present.
- The proprietary extension *MicroFragmentation* must be offered (see SectionSection 9.7.10)
- The SNI extension can be used to force the device to select a specific server certificate in the following *ServerHello* message.

9.7.2 ServerHello

As response to a correct ClientHello SECUREBOARD 1.0 sends a *ServerHello* as defined in RFC8446.

9.7.3 EncryptedExtensions

As defined in RFC 8446 the server provides further connection details within the *EncrypedExtensions* message.

9.7.4 Certificate

In order to authenticate the SECUREBOARD 1.0 itself, during handshake the server certificate is sent. The certificate is selected by the following rules:

- If the SNI extension in the ClientHello message matches the device certificate Common Name, the device certificate is used.
- Else if present the user certificate is used.
- Else the device certificates used.
- When sending certificates to the device the following restrictions apply for each certificate:
- The certificate must be an x509 certificate in Version 3.
- The certificate in DER encoded form must be less than **475 bytes**. Check with:

```
openssl x509 -in <certificate>.pem --outform DER | wc -c
```

9.7.5 CertifcateRequest

If the handshake is established using ECDSA/ECDH and if a User CA has been provisioned and if the handshake is established using the User Certificate, SECUREBOARD 1.0 request the user to authenticate itself using a client certificate issued in a chain of trust by the provisioned user CA. This is done be sending the *CertificateRequest* message.

SECUREBOARD expects to receive to complete chain up to and including the provisioned CA. During handshake all signature are verified. The root CA must match with the provisioned fingerprint.

9.7.6 CertificateVerify

- See RFC 8446 for details

9.7.7 Finished

- See RFC 8446 for details

9.7.8 KeyUpdate

- See RFC 8446 for details

9.7.9 NewSessionTicket

Verifying ECDSA signatures is a hard task for a low power MCU and takes about 1 second on SECUREBOARD 1.0. To avoid asymmetric cryptographic operations, a previously established connection can be resumed. RFC 8446 applies.

SECUREBOARD 1.0 offer a new session ticket after receiving a close notification as last message before replying with its own close notification.

- The PSK is stored in the device volatile memory and is valid as long as the device is powered for at most 7 days.

9.7.10 Micro Fragmentation Extension

Although RFC 6066 specifies the extension *max_fragment_length* to limit transmitted fragment size. The possible minimum size possible with *max_fragment_length* is 512 bytes. This is still too large for SECUREBOARD 1.0.

Extend the *ExtensionType* enumeration in TLS 1.3 by

```
enum {
  micro_fragmentation(65408), (65535)
}
```

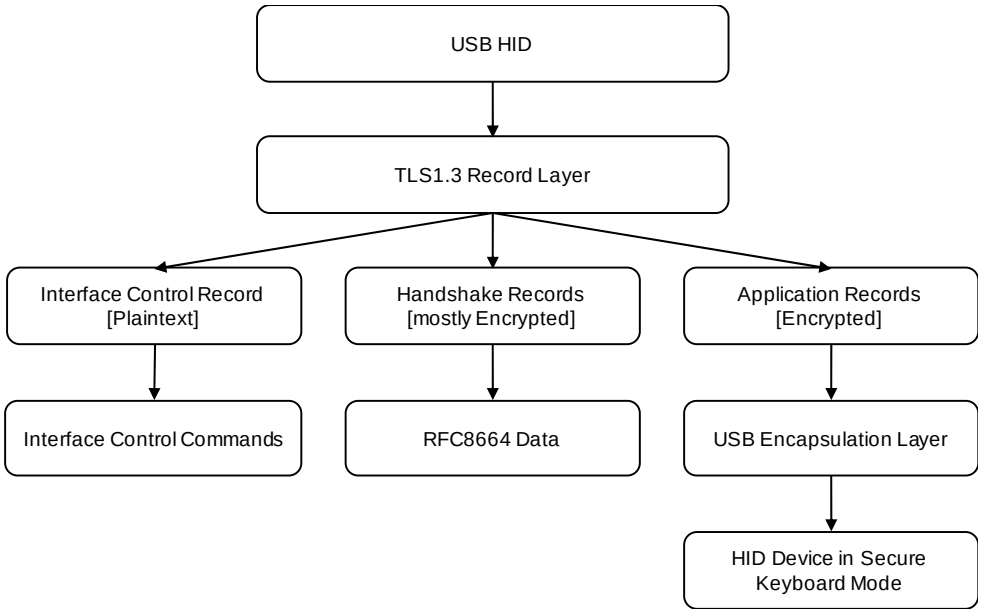
When contained in a client hallo the extension_data field shall be empty. When contained in the EnryptedExtension message the extension data shall contain:

```
enum {
  2^1-1 (1), 2^2-1 (2), 2^3-1 (3), 2^4-1 (4), 2^5-1 (5), 2^6-1 (6), 2^7-1 (7), 2^8-1 (8),
(255)
} MicroFragmentLength
```

Servers that receive an extended client hello containing a *micro_fragmentation* extension may request a maximum micro fragment length of plaintext records (before encryption) from the client by including the extension within the *EncryptedExtension* message. From there on the client must fragment all messages into records with the maximum payload indicated by the extension sent by the server.

9.8 Software Stack in Secure Keyboard Mode

The SW stack in Secure Keyboard Mode can be outlined as depicted below:



- USB HID is used for low level data transport using vender defined HID reports.
- Data sent to the SECUREBOARD 1.0 is encapsulated within the TLS 1.3 Record Layer defined in RFC 8446.
- Interface Control Records are used by the Relay Daemon to control the SKM Interface, using Interface Control Commands.
- Handshake records defined in RFC 8446 are used by libsecureboard to establish a secure TLD 1.3 Channel.
- Application Records are used to transport messages of the USB Encapsulation Layer.
- The USB Encapsulation takes care of providing an USB like data transport layer.
- HID Device in Secure Keyboard Mode is the final interface used by the Secure OS Binding to provide services to the OS.

Details on the specific parts are included within the Appendix A, Appendix B and Appendix C.

10 Vendor Specific Commands

Configuration Interface

Item	Type	Tag	Get/Set	Example Result	APDU
Vendor	String	100_h	GET	Cherry GmbH	FF 70 04 6A 06 E0 04 DF 82 00 00 00
Hardware Revision	Integer	101_h	GET	1	FF 70 04 6A 06 E0 04 DF 82 01 00 00
Firmware Label	String	102_h	GET	<complete label>	FF 70 04 6A 06 E0 04 DF 82 02 00 00
Firmware Version	4 Octets	103_h	GET	1.0.0.4	FF 70 04 6A 06 E0 04 DF 82 03 00 00
Firmware Source Reference	20 Octets	105_h	GET	SHA1	FF 70 04 6A 06 E0 04 DF 82 05 00 00
Start Bootloader (DFU Mode)	1 Octet Possible Values: 1	$1FF_h$	SET		FF 70 04 6A 07 E1 05 DF 83 7F 01 01 00
Bootloader Label	String	202_h	GET	<complete label>	FF 70 04 6A 06 E0 04 DF 84 02 00 00
Bootloader Version	4 Octets	203_h	GET	1.0.0.4	FF 70 04 6A 06 E0 04 DF 84 03 00 00
Bootloader Source Reference	20 Octets	205_h	GET	SHA1	FF 70 04 6A 06 E0 04 DF 84 05 00 00
Reset Reader Config	1 Octet Possible Values: 1	300_h	SET		FF 70 04 6A 07 E1 05 DF 86 00 01 01 00
CT Activation Sequence	1 Octet	400h	SET and		Set ($C6_{h)}$: APDU: FF 70 04 6A 07 E1 05 DF 88 00 01 C6 00
			GET	$C6_h$	Get: FF 70 04 6A 06 E0 04 DF 88 00 00 00

When reading data, the response is encapsulated within an ASN.1 TLV using the same Tag as defined in the table above. For example, the response to the Vendor query is:

```
DF 82 00 0B 43 68 65 72 72 79 20 47 6D 62 48 90 00
```

More specifically this means:

```
DF 82 00                              // ASN.1 encoded Tag 100h = 256d
0B                                    // ASN.1 encoded Length 0Bh = 11d bytes
43 68 65 72 72 79 20 47 6D 62 48      // ASCII Representations for Cherry GmbH
90 00                                 // SW1 SW2
```

10.1 Vendor

Retrieve the Vendor name *Cherry GmbH*.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 82 00 00 00
<- DF 82 00 0B 43 68 65 72 72 79 20 47 6D 62 48 90 00
```

10.2 Hardware Revision

Retrieve the Hardware Revision of the device.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 82 01 00 00
<- DF 82 01 01 01 90 00
```

10.3 Firmware Label

Retrieve the Firmware Label as String. The example below states:

SECUREBOARD1.0-1.0.0.6-183-E07ABA9DD17E3F0D3874FE04E84741A2362AF45C

The components denote

- SECUREBOARD1.0: Product description
- 1.0.0.6: Firmware Version
- 183: Build Number
- E07ABA9DD17E3F0D3874FE04E84741A2362AF45C: Source Hash.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 82 02 00 00
<- DF 82 02 43 53 45 43 55 52 45 42 4F 41 52 44 31
   2E 30 2D 31 2E 30 2E 30 2E 36 2D 31 38 33 2D 45
   30 37 41 42 41 39 44 44 31 37 45 33 46 30 44 33
   38 37 34 46 45 30 34 45 38 34 37 34 31 41 32 33
   36 32 41 46 34 35 43 90 00
```

10.4 Firmware Version

Retrieve the Firmware Version. The example below states *Version 1.0.0.6*.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 82 03 00 00
<- DF 82 03 04 01 00 00 06 90 00
```

10.5 Firmware Source Reference

Retrieve the Firmware Source Reference (SHA1 Hash).

Example:

```
-> FF 70 04 6A 06 E0 04 DF 82 05 00 00
```

```
<- DF 82 05 14 E0 7A BA 9D D1 7E 3F 0D 38 74 FE 04 E8 47 41 A2 36 2A F4 5C 90 00
```

10.6 Bootloader Label

Retrieve the Firmware Label as String. The example below states:

BL-1.0.0.5-182-E16BB753D009B95C8A206132C62C87F312A1FD0A

The components denote

- BL: Product description
- 1.0.0.5: Bootloader Version
- 182: Build Number
- E16BB753D009B95C8A206132C62C87F312A1FD0A: Source Hash.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 84 02 00 00
<- DF 84 02 37 42 4C 2D 31 2E 30 2E 30 2E 35 2D 31
   38 32 2D 45 31 36 42 42 37 35 33 44 30 30 39 42
   39 35 43 38 41 32 30 36 31 33 32 43 36 32 43 38
   37 46 33 31 32 41 31 46 44 30 41 90 00
```

10.7 Bootloader Version

Retrieve the Firmware Version. The example below states *Version 1.0.0.5*.

Example:

```
-> FF 70 04 6A 06 E0 04 DF 84 03 00 00
<- DF 84 03 04 01 00 00 05 90 00
```

10.8 Bootloader Source Reference

Retrieve the Firmware Source Reference (SHA1 Hash).

Example:

```
-> FF 70 04 6A 06 E0 04 DF 84 05 00 00
<- DF 84 05 14 E1 6B B7 53 D0 09 B9 5C 8A 20 61 32 C6 2C 87 F3 12 A1 FD 0A 90 00
```

10.9 Reset Reader Config

Set all non-volatile settings to their corresponding default values.

Example:

```
-> FF 70 04 6A 07 E1 05 DF 86 00 01 01 00
<- 90 00
```

10.10 Voltage Selection

CHERRY SECUREBOARD 1.0 contact card interface supports all classes listed in ISO/IEC 7816-3. It is possible to set the sequence followed during voltage selection process. This can be used for a card that supports more than one class or to accelerate activation time. The value is stored in the non-volatile configuration memory. Voltage selection sequence is encoded in one byte as following:

Bit	7	6	5	4	3	2	1	0

Sequence	4	3	2	1
Voltage	*00_b: 5V0*	*00_b: 5V0*	*00_b: 5V0*	*00_b: 5V0*
	01_b: 3V0	*01_b: 3V0*	*01_b: 3V0*	*01_b: 3V0*
	10_b: 1V8	*10_b: 1V8*	*10_b: 1V8*	*10_b: 1V8*
	11_b: Stop	*11_b: Stop*	*11_b: Stop*	*11_b: Stop*

Stop means that all following items in Sequence are ignored (e.g.: $0C_h$ can be used to limit the selection to 5V0).

The default value is: $11\text{-}10\text{-}01\text{-}00_b = E4_h$ = 5V0, 3V0, 1V8, Stop.

To change the sequence, use the Vendor Specific Command *CT Activation Sequence* defined in Section 10.

Example to set to *1V8, 3V0, 5V0*, *Stop* (Value $11\text{-}00\text{-}01\text{-}10_b = C6_h$)

```
-> FF 70 04 6A 07 E1 05 DF 88 00 01 C6 00
<- 90 00
```

Example to retrieve the current setting.

```
-> FF 70 04 6A 06 E0 04 DF 88 00 00 00
<- DF 88 00 01 C6 90 00
```

10.11 Start Bootloader

This command can be used to bring the device into DFU Mode (see Firmware Update in Section 11)

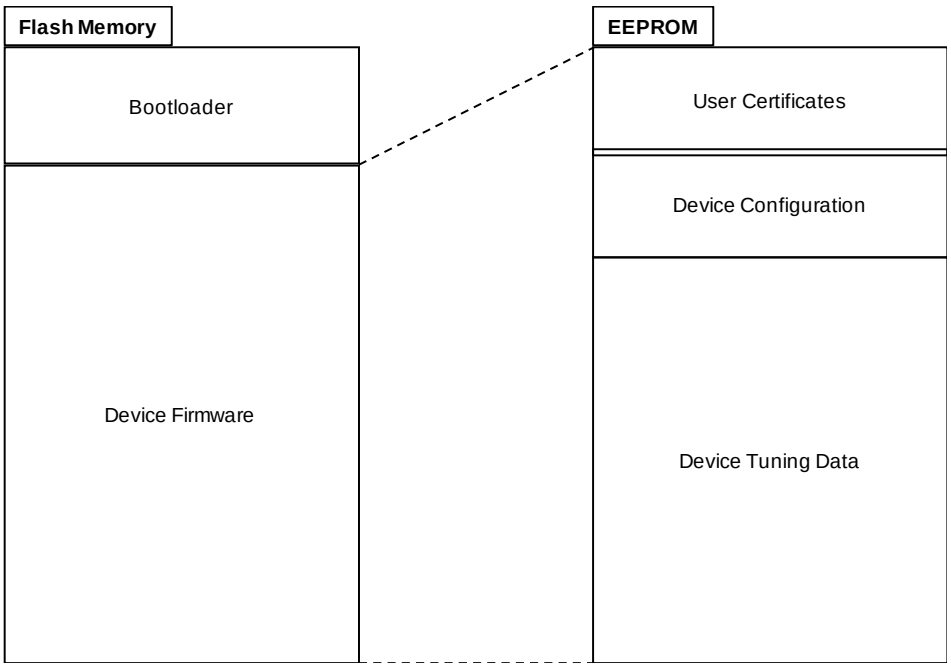
Example to enter DFU Mode.

```
-> FF 70 04 6A 07 E1 05 DF 83 7F 01 01 00
<- 90 00
```

11 Firmware Update

11.1 Memory Layout

SECUREBOARD 1.0 incorporates of a Flash and an EEPROM Memory. The memories and their use within the device are depicted in the chart below.



- The **Flash memory** stores the bootloader and the device firmware.
- The **EEPROM** stores device tuning data, device configuration and user certificates.

This section describes how to update the flash memory. In order to change the EEPROM data follow the instructions in the corresponding sections.

11.2 DFU Mode

Firmware updates are executed, in DFU Mode (Device Firmware Update). DFU is specified within USB-IF (https://www.usb.org/sites/default/files/DFU_1.1.pdf), therefore and tools are widely available in most common operating systems. We recommend the opens source tool *dfu-util* Version 0.9. *dfu-util* is included in most Linux distributions. Further binaries (e.g.: Windows) as well as the source is available at (https://sourceforge.net/projects/dfu-util/files/). While in DFU Mode keyboard, and CCID interfaces are deactivated.

During DFU operations the device executes a couple of steps to verify firmware genuineness (e.g.: Image Header, Image Signature, …). If possible, the checks are executed before any memory changes are made. However, some checks require the parts of the image being stored in the device before the update process completes.

If a check for done before changing the flash memory is fails. After device reboot, the SECUREBOARD 1.0 continues operating with the old firmware. If a problem occurs after updating parts of the firmware the device stops within the bootloader and enters DFU Mode to allow repeating the process with a correct firmware image.

11.2.1 Entering the DFU Mode

Before attempting to update the device firmware the device has to be brought into DFU mode. Either one of the following choices can be used.

- Disconnect the device
  Hold the following key combination while powering the device:

D+F+RGUI *Note: RGUI denotes the right Windows key*

- Send the Vendor Escape APDU:

```
FF 70 04 6A 07 E1 05 DF 83 7F 01 01 00
```

- Send the Vendor Command (with (P)ICC is inserted):

```
FF 70 04 6A 07 E1 05 DF 83 7F 01 01 00
```

If either of the procedure above is successful de device enters DFU mode and indicates this by showing the following LED pattern from left to right (blue, blue, orange, red).

Notes for Linux, Mac:

The device is now ready to accept a new firmware image.

Notes for Windows:

- The host detects the device as WinUSB device in the Device Manager: The device is now ready to accept a new firmware image.
- In rare cases the device is not detected properly as WinUSB device. In this case execute the following steps:
  - Disconnect the SECUREBOARD 1.0
  - Open regedit.exe
  - Navigate into the following folder:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\usbflags
```

  - Delete all subfolders starting with 046A01A2.
  - Reconnect the SECUREBOARD 1.0.
  - The device is detected as WinUSB.

11.2.2 Updating the Firmware

Read the complete section before starting the update.

DO NOT REMOVE POWER nor RECONNECT UNLESS INSTRUCTED.

Step 1. Enter DFU Mode as described above.

- Ensure that the device is in DFU Mode and that the LEDs show the following pattern from left to right (blue, blue, orange, red).

Step 2. Update the device firmware.

Linux, MAC: Open a **root** command line and execute the following command:

```
dfu-util -D <kc1000mc-<version>.dfu>
```

or Windows: Open an **administrator** command line and execute the following command:

```
dfu-util.exe -D < kc1000mc-<version>.dfu>
```

The output should look something like this:

```
# sudo dfu-util -D kc1000mc-1.0.0.6-183-e07aba9dd17e3f0d3874fe04e84741a2362af45c.dfu
dfu-util 0.9
```

Example:

```
Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2016 Tormod Volden and Stefan Schmidt
```

```
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Match vendor ID from file: 046a
Match product ID from file: 01a2
Opening DFU capable USB device...
ID 046a:01a2
Run-time device DFU version 0101
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0101
Device returned transfer size 64
Copying data from PC to DFU device
Download        [=========================] 100%       127040 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

Step 3. Reboot: After reconnecting the device, it starts the new firmware.

Troubleshooting:

- Ensure you have root/administrator privileges.

Possible side effects:

- On error the firmware gets inaccessible and the device stays in DFU mode.
  - Reasons: Invalid Image, Flash/EEPROM damaged.
  - Fix: Repeat the steps described in this section using a proper image.
- User certificates get deleted.
  - Reasons: Old user certificates are incompatible with the new firmware.
  - Fix: Provision new certificates.
- User configuration defaults to factory settings.
  - Reasons: Configuration data is incompatible with the new firmware.
  - Fix: Update setting to you needs.

11.2.3 Updating the Bootloader

Read the complete section before starting the update.

DO NOT REMOVE POWER nor RECONNECT UNLESS INSTRUCTED.

Step 1. Enter DFU Mode as described above.

- Ensure that the device is in DFU Mode and that the LEDs show the following pattern from left to right (blue, blue, orange, red).

Step 2. Load the bootloader updater firmware.

Linux, MAC: Open a **root** command line and execute the following command:

```
dfu-util -D <boot-dfu-kc1000mc-<version>.dfu>
```

or Windows: Open an **administrator** command line and execute the following command:

```
dfu-util.exe -D <boot-dfu-kc1000mc-<version>.dfu>
```

Example:

```
# sudo dfu-util -D boot-dfu-kc1000mc-1.0.0.5-182-
e16bb753d009b95c8a206132c62c87f312a1fd0a.dfu
dfu-util 0.9

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2016 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Match vendor ID from file: 046a
Match product ID from file: 01a2
Opening DFU capable USB device...
ID 046a:01a2
Run-time device DFU version 0101
Claiming USB DFU Interface...
Setting Alternate Setting #0 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0101
Device returned transfer size 64
Copying data from PC to DFU device
Download        [=========================] 100%        24768 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

Step 3. Start the bootloader update process by reconnecting the device.

- The following Led patterns are indicated.
- The following Led pattern indicated that the update firmware started.
  (blue, blue, green, off)
- After approximately one second the following pattern indicates that the bootloader is currently updating.
  (blue, blue, off, flashing red).

DO NOT REMOVE POWER nor RECONNECT WHILE THE SKM LED IS FLASHING.

Step 4. Once all LEDs are turned off the update was successful.

Step 5. Reconnect the device to start new bootloader.

Step 6. Update Firmware as described above.

Troubleshooting:

- Ensure you have root/administrator privileges.

Possible side effects:

- On error the firmware gets inaccessible and the device stays in DFU mode.
  - Reasons: Invalid Image, Flash/EEPROM damaged
  - Fix: Repeat the steps described in this section using a proper image
- Old Firmware is deleted.
  - Reasons: Expected during bootloader updates.
  - Fix: Update Firmware as described above.

Appendix A Secure Keyboard Mode USB Interface

The SKM USB Interface exposes an USB HID interface with one IN and one OUT vendor report without IDs and each with a length of 64 bytes. The reports are used to transport TLS record including the 5-byte header defined in RFC8446. The *ContentType* is extended as defined below.

```
enum {
  invalid(0),
  change_cipher_spec(20),
  alert(21),
  handshake(22),
  application_data(23),
  interface_control(254),
  (255)
ContentType;
```

All *ContentTypes* except *interface_control* are treated as defined in RFC 8446 *interface_control* is used by the relay daemon to control the SKM interface.

A.1 Record Transmission

Records are split into chunks of 64 bytes without modification and transmitted on after each other onto the interface pipes. If the record size is not a multiple of 64 bytes, the last chunk is padded with (FF_h).

A.2 Interface Control

Interface Control Records use the *ContentType interface_control,* are sent as plaintext and can be transmitted at any time. The record data contains a 2-byte header and optional data appended.

```
InterfaceControl Header {
  uint8 Length
  uint8 Command
}
```

Length	Length of the data appended to the header (without size of header)
Command	Command ID of command to execute

A.2.1 Interface Control: CANCEL_CONNECTION

Length: 00_h
Command: 00_h.

When a SECUREBOARD 1.0 receives this message, it resets the SKM state to disabled. If needed the SKM Canceled state is entered for 10 seconds during this process.

Appendix B USB Encapsulation Layer

As usual in USB all multibyte integers are transmitted in Little-Endian Format. The USB Encapsulation Packets are encrypted and sent within TLS 1.3 application records.

The USB Encapsulation Layer does not support nor required USB signaling, addressing, or similar. Once the TLS 1.3 channel is established requests can be sent.

B.1 Message Header

USB Encapsulation messages have a 3-byte header. The Header is defined as follows:

Byte	0								1								2							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Field	Length (Bits 7..0)								S	Length (Bits 14..8)							D	Endpoint						

- Length: Number of bytes (excluding header) following the header.
- S: Bit indicating a stall condition, when sent by the SECUREBOARD 1.0. Unused (set to 0) when sent by the host.
- D: Direction Bit: 0 [OUT] indicates transfer from host to SECUREBOARD 1.0; 1 [IN] indicates transfer from SECUREBOARD 1.0 to host
- Endpoint: Endpoint Number of this transfer. The special value $7F_h$ indicates a SETUP transfer.

B.2 Control transfers

- Similar to USB the USB Encapsulation layer also provides USB Control transfers.

Each Control Transfer is Initiated by the host by sending a Setup Packet. The setup packet has the same field and semantics as usual in USB.

Byte	0	1	2	3	4	5	6	7
Field	bmRequestType	bRequest	wValue		wIndex		wLength	

bmRequestType Bits

Bit	7	6	5	4	3	2	1	0
Field	Direction	Type		Recipient				

- bmRequestType.Direction:

  0 -> Host to SECUREBOARD 1.0;
  1 -> SECUREBOARD 1.0 to Host

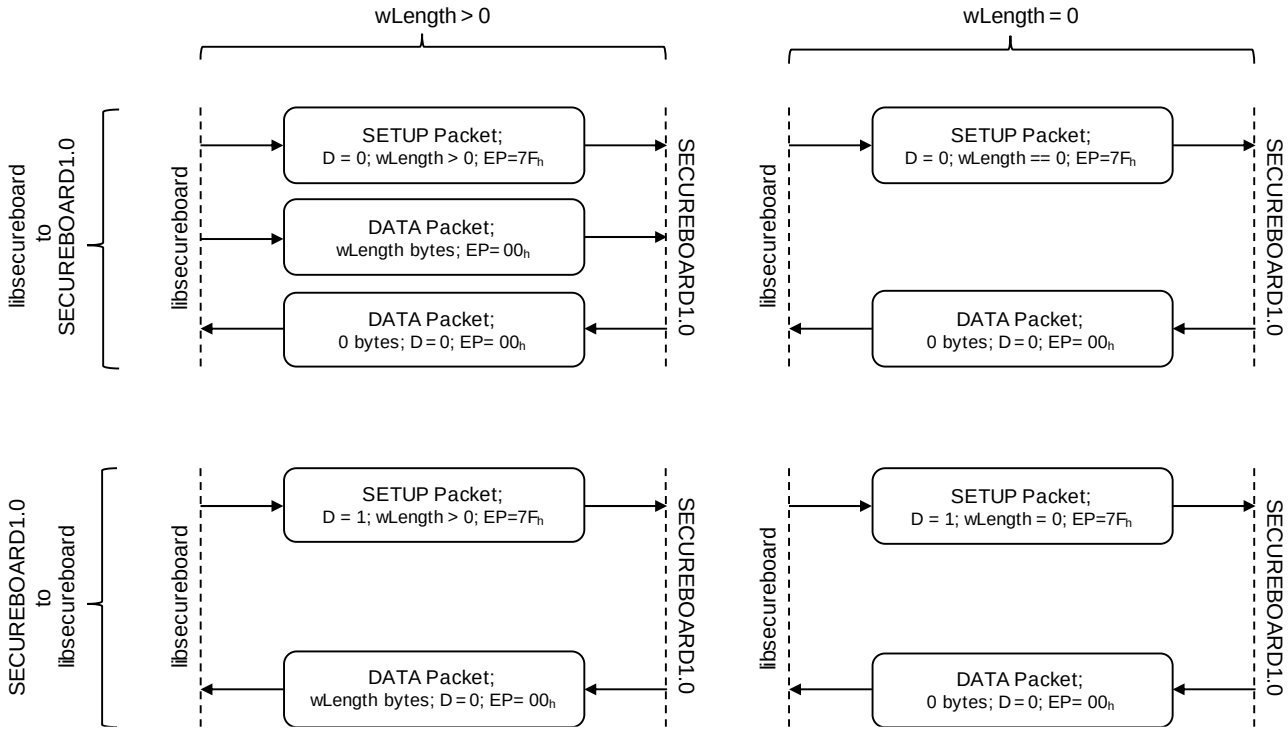
- bmRequestType.Type

  0 -> Standard request
  1 -> Class request
  2 -> Vendor Request

- bmRequestType.Recipient

  0 -> Device
  1 -> Interface
  2 -> Endpoint

- bRequest: Request
- wValue: Value
- wIndex: Index
- wLength: Number of bytes of the transfer in the Data Phase
- The chart below depicts the packets sent during control transfers within the USB Encapsulation Layer.



Example:

```
# Control Transfer: Type VENDOR, INTERFACE REQUEST to Device; 4 Bytes of Data
# 1 Setup
-> 08 00 7F    41 00 00 00 00 00 04 00
# 2 Data
-> 04 00 00    01 02 03 04
# 3 Response
<- 00 00 80
```

Appendix C Encapsulated SECUREBOARD 1.0

This section describes the supported request of SECUREBOARD 1.0 within the Encapsulated Channel.

C.1 Control Transfers

C.1.1 Get Descriptor

This request is used to retrieve device descriptor.

Setup Packet	Value	Description
bmRequestType	1 00 00000_b	STANDARD, DEVICE, DEVICE_to_HOST
bRequest	6	GET_DESCRIPTOR
wIndex	0	RFU / Set 0
wValue	Type (high byte), Index(low byte)	Supported Types: DEVICE_DESCRIPTOR: 1 CONFIGURATION_DESCRIPTOR: 2
wLength	Maximum length to be received	The device truncates the data if wLength is shorter than the Descriptor.

C.1.2 Get Firmware Version

This request is used to retrieve firmware version.

Setup Packet	Value	Description
bmRequestType	1 00 00000_b	VENDOR, DEVICE, DEVICE_to_HOST
bRequest	0	GET_FW_VERSION
wIndex	0	RFU / Set 0
wValue	0	RFU / Set 0
wLength	4	

C.1.3 Set Report

Set report has the same semantic as SET_REPORT for HID 1.11 devices (see HID 1.11 for details).

Setup Packet	Value	Description
bmRequestType	0 01 00001_b	CLASS, INTERFACE, HOST_to_DEVICE
bRequest	9	SET_REPORT
wIndex	0	0

wValue	0	0
wLength	1	Same value as in normal mode.

C.1.4 Set Running

This request is to bring the device into scanning mode. By default, the device will not generate any HID reports. Reports are generated only after the device is brought into running mode.

Set running is only available when connected with the user device certificate.

Setup Packet	Value	Description
bmRequestType	0 10 00001$_b$	VENDOR, INTERFACE, HOST_to_DEVICE
bRequest	0	SET_RUNNING
wIndex	0	RFU / Set 0
wValue	0	0: stopped 1: running
wLength	0	

C.1.5 Set User Certificate

Update the user device certificate. The payload is the DER encoded x509 certificate in Version 3. The allow maximum size 572 bytes.

Setup Packet	Value	Description
bmRequestType	0 10 00001$_b$	VENDOR, INTERFACE, HOST_to_DEVICE
bRequest	1	SET_USER_CERT
wIndex	0	0
wValue	0	0
wLength	Certificate size.	Maximum size 572 bytes.

C.1.6 Set User Private Key

Update the fingerprint of the user root CA. The payload is a 32 Byte private key that matches the User Certificate.

Setup Packet	Value	Description
bmRequestType	0 10 00001$_b$	VENDOR, INTERFACE, HOST_to_DEVICE
bRequest	2	SET_USER_PRIVATE_KEY
wIndex	0	0

wValue	0	0
wLength	1	32

C.1.7 Set User Root CA

Update the fingerprint of the user root CA. The payload is the 32 Byte SHA256 fingerprint of the user root CA.

Setup Packet	Value	Description
bmRequestType	0 10 00001$_b$	VENDOR, INTERFACE, HOST_to_DEVICE
bRequest	3	SET_USER_ROOT_CA
wIndex	0	0
wValue	0	0
wLength	32	Size of SHA256 Digest

C.2 HID Reports

In Secure Keyboard Mode the device generates reports the same way as in normal mode. The report is formatted as defined in HID1.11 with the following report descriptor:

```
    0x06, 0x00, 0xFF,  // Usage Page (Vendor Defined 0xFF00)
    0x09, 0x01,        // Usage (0x01)
    0xA1, 0x01,        // Collection (Application)
    0x19, 0x00,        //   Usage Minimum (0x00)
    0x29, 0x3f,        //   Usage Maximum (0x1e)
    0x15, 0x00,        //   Logical Minimum (0)
    0x26, 0xFF, 0x00,  //   Logical Maximum (255)
    0x75, 0x08,        //   Report Size (8)
    0x95, 0x30,        //   Report Count (48)
    0x82, 0x00, 0x01,  //   Input (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null
                       //   Position)
    0x95, 0x40,        //   Report Count (64)
    0x19, 0x00,        //   Usage Minimum (0x00)
    0x29, 0x39,        //   Usage Maximum (0x39)
    0x92, 0x00, 0x01,  //   Output (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null
                       //   Position,Non-volatile)
    0xC0,              // End Collection
```

The report is received within the USB Encapsulation layer on EP1.

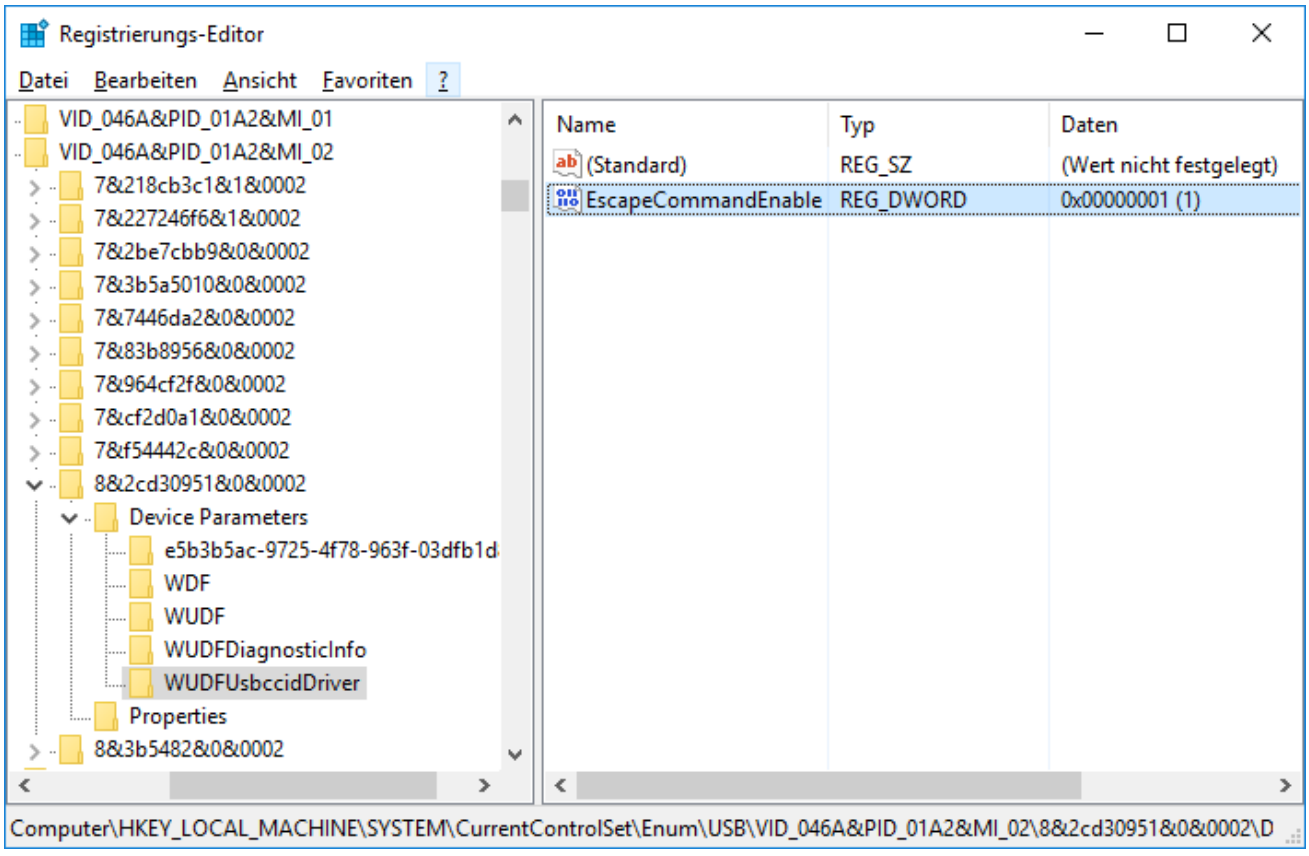
Appendix D Enabling Escape CCID Commands

- In order to send or receive an Escape command to a reader using Microsoft's CCID driver, add the DWORD registry value **EscapeCommandEnable** and set to a non-zero value under one of the following keys:
- **Windows 7 to 10** HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\ VID_046A&PID_01A2&MI_02\<serial number>\Device Parameters\WUDFUsbccidDriver
- **Prior Windows 7** HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\ VID_046A&PID_01A2&MI_02\<serial number>\ Device Parameters

Then the vendor IOCTL for the Escape command is defined as follows:

#define IOCTL_CCID_ESCAPE SCARD_CTL_CODE(3500).

For details see http://msdn.microsoft.com/en-us/windows/hardware/gg487509.aspx.



If a reader with a different serial number is connected to the computer the operation must be repeated.

Appendix E Definitions, Abbreviations and Symbols

APDU	Application Protocol Data Unit
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CHACHA20	Stream Cipher
CHACHA20POLY1305	Stream Cipher with MAC
CLA	Class byte of an APDU
DER	Distinguished Encoding Rules
EC	Elliptic Curve
ECDH	Elliptic Curve Diffie Hellman Key Exchange
ECDSA	Elliptic Curve Digital Signing Algorithm
ICC	Integrated Circuit Card
IETF	Internet Engineering Task Force
IFD	Interface Device (for accessing ICC card)
MSDN	Microsoft® Developer Network
PC/SC	Personal Computer/Smart Card
PICC	Proximity Integrated Circuit Card
PSK	Pre-Shared Key
RFC	Document System hosted by IETF
TLS	Transport Layer Security

Appendix F References

ISO 7816-3	ISO 7816-3 Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Electrical interface and transmission protocols Third edition - 2006-11-01
ISO 7816-4	ISO 7816-4 Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for Interchange Second edition - 2005-01-15
ISO 8825	ISO/IEC8825 ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) Fourth edition 2008-12-15*, or* X.690 Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
PCSC-3-Sup-CL	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Supplemental Document for Contactless ICCs Revision 2.02.00
PCSC-3	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Requirements for PC-Connected Interface Devices Revision 2.01.09
PCSC-3-Sup	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Supplemental Document Revision 2.01.09
PCSC-3-AMD	Interoperability Specification for ICCs and Personal Computer Systems Part 3. Requirements for PC-Connected Interface Devices - AMENDMENT 1 Revision 2.01.09
PCSC-10-SPE	Interoperability Specification for ICCs and Personal Computer Systems Part 10. IFDs with Secure Pin Entry Capabilities Revision 2.02.08
CCID	Specification for Integrated Circuit(s) Cards Interface Devices Revision 1.1 April 22nd, 2005
RFC 8446	The Transport Layer Security (TLS) Protocol Version 1.3