IFX I2C Protocol

Protocol Specification

About this document

Scope and purpose

The scope of this document is the IFX I2C Protocol and covers the necessary information to integrate a Infineon I2C device with a system hosting it.

Intended audience

This document addresses the audience: customers, solution providers, and system integrators.

Please read the Important Notice and Warnings at the end of this document

Table of Contents

Figures

Tables

1 Introduction

The I2C Device Protocol Specification specifies the format and behavior of the protocol, and its possible variations as well. The protocol specified within this document is used for IFX Platform Security I2C Device if not differently required by a particular project.

1.1 References

This section provides references to information used in this document or information useful becoming familiar with in the context of the technology specified within this document.

Term	Definition
[I2CBus]	A 2-wire serial bus designed by Philips to allow easy communication between components that reside on the same circuit board.
[ProtRep]	André A.S. Danthine: Protocol Representation with Finite-State Models, IEEE Transactions on Communications, Vol. COM-28, No. 4, PP 632-643, April 1980
[RFC1549]	Request For Comments: 1549: PPP in HDLC Framing
[RFC1662]	Request For Comments: 1662: PPP in HDLC-like Framing (obsoletes [RFC1549])
[RFC5246]	The Transport Layer Security (TLS) Protocol, Version 1.2, August 2008
[SP800-38C]	Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf
[SP800-38A]	Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf

Table 1-1 References

1.2 Definitions

The terms defined here are collected from different application and technology domains to maintain a common terminology used throughout the documents associated with the Protocol Specification project. They are generally in alphabetical order to provide the greatest accessibility.

1.2.1 Abbreviations

API	**A**pplication **P**rogramming **I**nterface
APDU	**Ap**plication **D**ata **U**nit
App	**App**lication
BBD	**B**lock **D**efinition **D**iagram
CSUM	**C**heck **Sum**
CMD	**C**o**m**man**d**
FCTR	**F**rame **C**on**tr**ol
NVM	**N**on-**V**olatile **M**emory
PCTR	**P**acket **C**on**tr**ol
RFU	**R**eserved for **F**uture **U**se

STA	**Sta**tus
UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator

1.3 Terminology Used in This Document

Parts of this document contain specification requirements. The use of the words “must,” “should,” “may,” and “reserved” in these specifications have the following meanings:

“Must” means that the specification is an absolute requirement.

“Must not” means that the specification is an absolute prohibition.

“Should” means that there may be valid reasons in particular circumstances to ignore the specification, but their full implications must be understood and carefully weighed before choosing to do so.

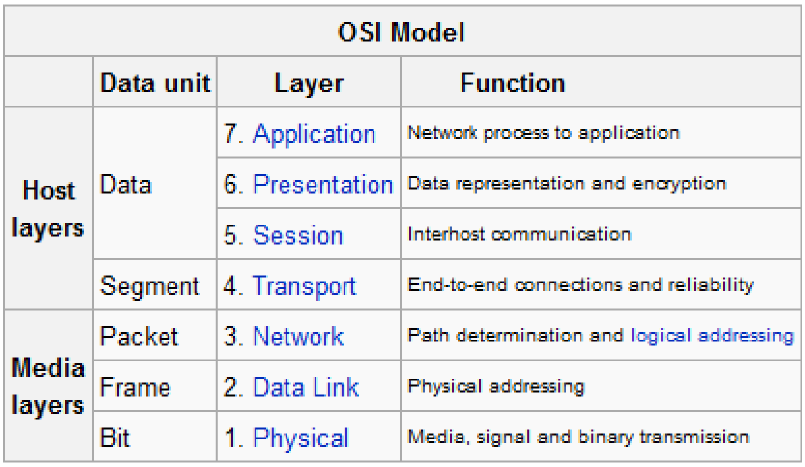
“Should not” means that there may be valid reasons in particular circumstances that make the specified action or feature acceptable, but their full implications must be understood and carefully weighed before choosing to include it.

“May” means that the indicated action or feature does not contravene this specification.

When a data field is marked “reserved,” accessories writing to it must set it to 0 and accessories reading it must ignore its value

1.4 Overview

The ISO OSI (open system interconnection) reference model is used to describe the protocol and its various layers. The OSI reference model has 7 layers, the incarnation based on the IFX Standard Protocol Extension has 6 (1, 2, 3, 4, 6, 7).



OSI Model

	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation and encryption
		5. Session	Interhost communication
	Segment	4. Transport	End-to-end connections and reliability
Media layers	Packet	3. Network	Path determination and logical addressing
	Frame	2. Data Link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

Figure 1-1 OSI reference model Communication

The protocol is defined for point-to-point connection with low failure rate. It is optimized for minimum RAM usage and minimum overhead to achieve maximum bandwidth, but also offers error handling and flow control.

The following sections describe the OSI layers of the protocol specified by this document.

- Section 2 Physical Layer
- Section 3 Data Link Layer
- Section 4 Network Layer
- Section 5 Transport Layer
- Section 6 Presentation Layer
- Section 7 Application Layer

2 Physical Layer

The Standardized Physical Layer is entirely defined in [UM10204.pdf]. Only a subset of those definitions is used for this protocol:

- Support of 7-Bit Addressing only (only 1 Address value)
- Single-Master / Multi-Slave configuration
- Speed (Fast Mode (Fm) up to 400 KHz; optional (Fm+) up to 1000 KHz)
- Optional Clock Stretching

To avoid losing communications on the Physical Layer the „Late Acknowledge Algorithm“ (refer to Figure 2-2: 'Write with Late Acknowledge' and Figure 2-3: 'Read with Late Acknowledge') is used to synchronize master and slave in case the I2C Master attempts to access the I2C slave which is not yet ready. Those cases could be e.g. leaving the sleep mode or preparing the data to be read. In case the HW implementation supports clock stretching the "Late Acknowledge Algorithm" might not be used.

Figure 2-1: 'BDD Overview' depicts the communication nodes supported by this protocol. This UML block definition diagram is provided to maintain a common terminology throughout this section, in particular the lifelines of the subsequent sequence diagrams are reusing the same blocks.

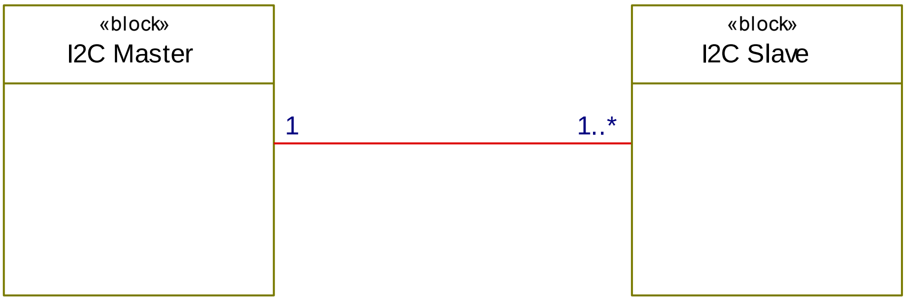


Figure 2-1 BDD Overview

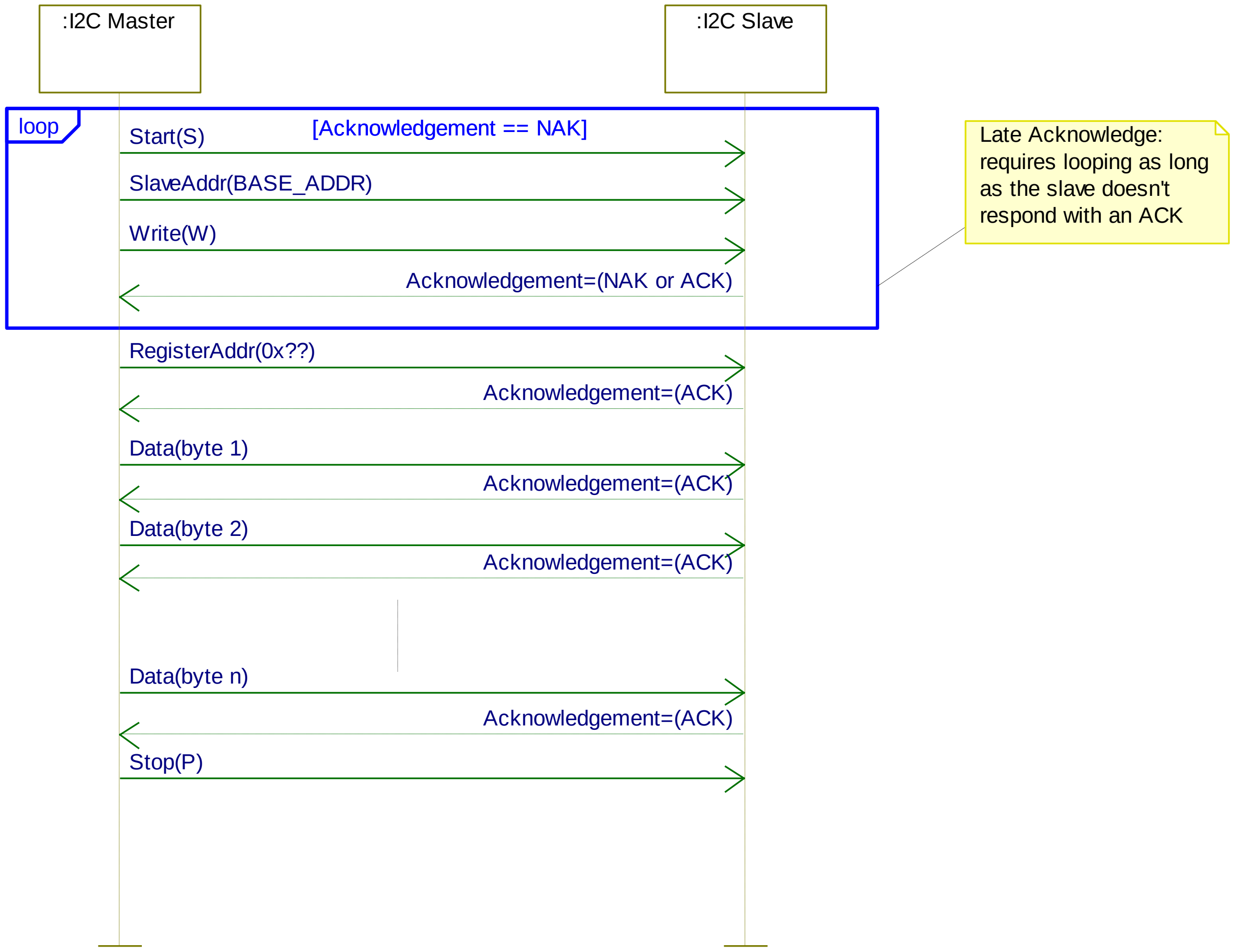


Figure 2-2 Write with Late Acknowledge

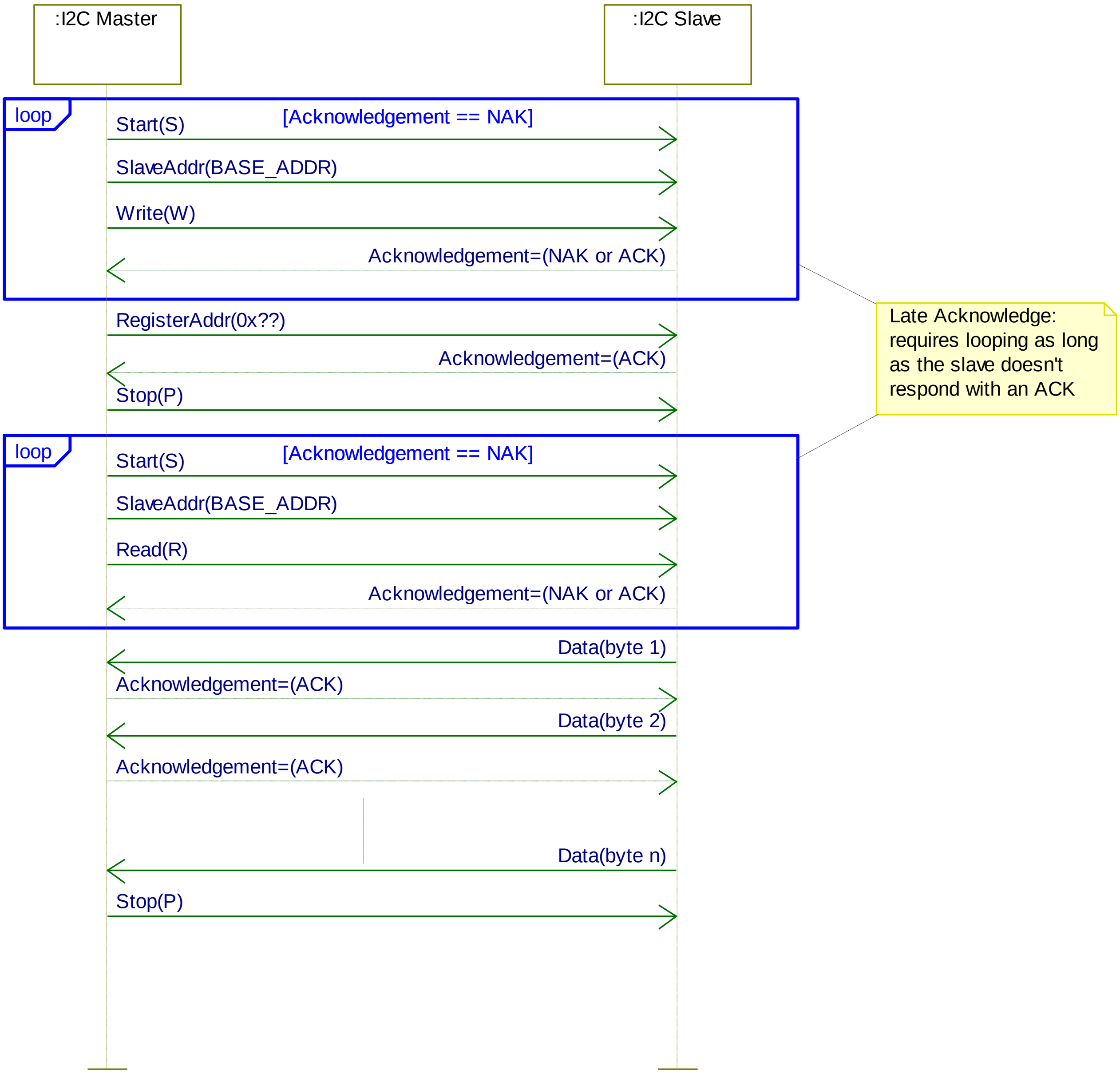


Figure 2-3 Read with Late Acknowledge

2.1 IFX Physical Layer Extension

The IFX Physical Layer is used to establish several sub-addresses under a single I2C base address as defined by **[I2CBus]**. The sub-addresses are IFX proprietary and defined as follows:

The I2C slave uses different address locations for status, control and data communication registers.

Note 1: *Multibyte numeric values are stored and send over the line in big-endian order; for example, the first byte in a two-byte register is the MSB of the stored value and the second byte is its LSB.*

Note 2: *In case the master reads beyond the end of a register or reads from an invalid register address the slave returns 0xFF on the data line.*

Register Address	Name	Size in Bytes	Description	Master Access
0x80	DATA	DATA_REG_LEN	This is the location where data shall be read from or written to the I2C slave	Read / Write
0x81	DATA_REG_LEN	2	This register holds the maximum data register (Addr 0x80) length. The allowed values are 0x0010 up to 0xFFFF. After writing the new data register length it becomes effective with the next I2C master access. However, in case the slave could not accept the new length it indicates its maximum possible length within this register. Therefore, it is recommended to read the value back after writing it to be sure the I2C slave did accept the new value. *Note: the value of MAX_PACKET_SIZE is derived from this value or vice versa (MAX_PACKET_SIZE= DATA_REG_LEN-5)*	Read / Write
0x82	I2C_STATE	4	Bits 31:24 of this register provides the I2C state in regards to the supported features (e.g. clock stretching …) and whether the device is busy executing a command and/or ready to return a response etc. Bits 15:0 defining the length of the response data block at the physical layer.	Read only
0x83	BASE_ADDR	2	This register holds the I2C base address as specified by Table 2-2. If not differently defined by a particular project the default value at reset is 0x20. After writing a different address the new address become effective with the next I2C master access. In case the bit 15 is set in addition to the new address (bit 6:0) it becomes the new default address at reset (persistent storage).	Write only
0x84	MAX_SCL_FREQU	4	This register holds the maximum clock frequency in KHz supported by the I2C slave. The value gets adjusted to the register I2C_Mode setting. Fast Mode (Fm): The allowed values are 50 up to 400. Fast Mode (Fm+): The allowed values are 50 up to 1000.	Read
0x85	GUARD_TIME[1]	4	For details refer to Table 'List of protocol	Read only

[1] In case the register returns 0xFFFFFFFF the register is not supported and the default values specified in Table 'List of protocol variations' shall be applied.

Register Address	Name	Size in Bytes	Description	Master Access
			variations'	
0x86	TRANS_TIMEOUT[1]	4	For details refer to Table 'List of protocol variations'	Read only
0x87	PWR_SAVE_TIMEOUT[1]	4	This optional register holds the minimum timeout [ms] after which the device enters power save mode. This timeout gets retriggered with any valid slave address. Once the power save state is initiated the addressing of the slave terminates it and the device returns to normal operation. The new value becomes effective with the next addressing of the slave. However, in case the slave could not accept the new value it indicates its maximum (the new value is too high to be accepted) or minimum (the new value is too low to be accepted) possible value within this register. Therefore, it is recommended to read the value back after writing it to be sure the slave did accept the new value.	Read / Write
0x88	SOFT_RESET	2	Writing to this register will cause a device reset. This feature is optional	Write only
0x89	I2C_MODE	2	This register holds the current I2C Mode as defined by Table 2-3. The default mode is SM & FM (011B).	Read / Write
0x90	APP_STATE_0	4	This register holds application specific information (definition of content and behavior is up to the application). The default value at reset is 0x00000000. This register is bound to the application which is linked to by channel 0 (default) of the network layer.	Read only
0x9X	APP_STATE_X	4	This register holds application specific information (definition of content and behavior is up to the application). The default value at reset is 0x00000000. This register is bound to the application which is linked to by channel X (X = 0x1 … 0xf) of the network layer.	Read only
0xA0	FOR_IFX_USE_1	4	This register can be used for IFX specific purpose during development and evaluation. The content doesn't impact the protocol execution. For Product releases this register is RFU.	Read / Write
0xA1	FOR_IFX_USE_2	4	Refer to register 0xA0	Read / Write

Table 2-1 IFX standard register set

[1] In case the register returns 0xFFFFFFFF the register and its functionality is not supported.

15	14	13	12	11	10	9	8
DEF_ADDR	RFU						
7	6	5	4	3	2	1	0
RFU	BASE_ADDR						

Field	Bits	Value	Description
DEF_ADDR	15	0 1	Volatile address setting by bit 6:0, lost after reset. Persistent address setting by bit 6:0, becoming default after reset.
BASE_ADDR	6:0	0x00-0x7F	I^{2}C base address specified by Section 2.1

Table 2-2 Definition of BASE_ADDR

15	14	13	12	11	10	9	8
DEF_MODE	RFU						
7	6	5	4	3	2	1	0
RFU					Mode		

Field	Bits	Value	Description
DEF_MODE	15	0 1	Volatile mode setting by bit 2:0, lost after reset. Persistent mode setting by bit 2:0, becoming default after reset. This bit is always read as 0.
MODE[1]	2:0	001 010 011 100 other values	Sm Fm SM & Fm (fab out default) Fm+ not valid; writing will be ignored

Table 2-3 Definition of I2C_MODE

[1] This mode defines the adherence of the bus signals to the electrical characteristics according [I2CBus]

31	30	29	28	27	26	25	24
BUSY	RESP_RDY	RFU		SOFT_RESET	CONT_READ	REP_START	CLK_STRETCHING
23	22	21	20	19	18	17	16
PRESENT_LAYER	RFU						
15-0							
Length of data block to be read							

Field	Bit(s)	Value	Description
BUSY	31	0 1	Device is not busy Device is busy executing a command
RESP_RDY	30	0 1	Device is not ready to return a response Device is ready to return a response
SOFT_RESET	27	0 1	SOFT_RESET not supported SOFT_RESET supported
CONT_READ	26	0 1	Continue Read not supported Continue Read supported
REP_START	25	0 1	Repeated start not supported Repeated start supported
CLK_STRETCHING	24	0 1	Clock stretching not supported Clock stretching supported
PRESENT_LAYER	23	0 1	Presentation Layer not supported Presentation Layer supported

Table 2-4 Definition of I2C_STATE

2.1.1 OSI level standard use

For IFX standard data communication at the DATA register, the OSI Layer 2,3,4,6,7 as in this document are used. Figure 2-4 depicts the overview of the protocol stack structure.

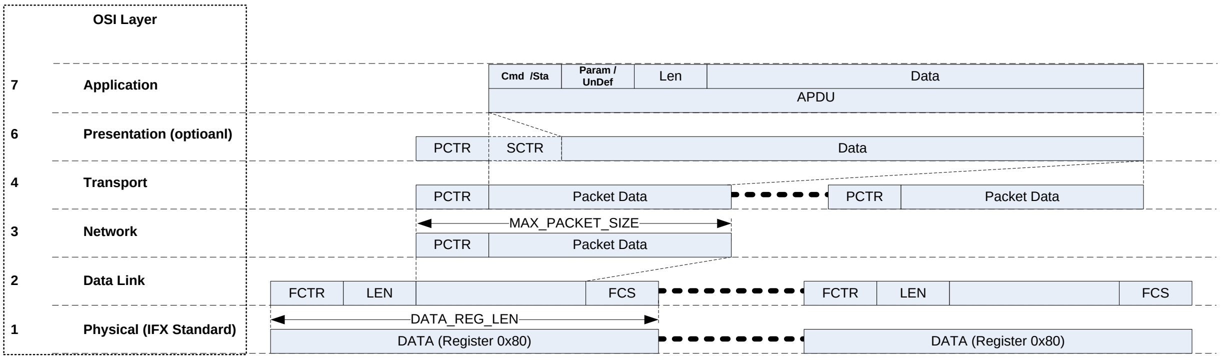


Figure 2-4 Overview Protocol Stack on IFX Physical Layer Extension (Standard)

3 Data Link Layer

The main task of the data link layer is to take a raw transmission facility and transform it into a line that appears free of transmission errors for higher layers. It accomplishes this task by taking packets of a maximum size from the sender, put a frame around, transmit the frames sequentially, and process the acknowledgment frames sent back by the receiver. To detect destroyed or incomplete frames a checksum is added. The packets are passed to higher levels of the receiver in the same order as they are provided by the sender though the frames may be transmitted in a different order over the physical line.

3.1 Structure of a frame

The figure below shows the structure of a frame:

1 byte	2 bytes	≤ MAX_PACKET_SIZE bytes	2 bytes
FCTR	LEN	packet	FCS

FCTR: Frame Control
LEN: Length of packet data
FCS: Frame Checksum

Figure 3-1 Structure of a frame

A frame starts with the frame type including the frame number (FCTR) followed by the packet length (LEN) and the packet data. For the structure of the packet data, see Section 4. After the packet data a 2 byte checksum (FCS) is added to the end. The maximum size of the packet data is limited to DATA_REG_LEN Bytes.

3.2 Frame types

There are two types of frames:

1. Control frames and
2. Data frames

Control frames are used to control the flow of data frames and synchronization and are not numbered (and not acknowledged). Control frames doesn't carry data (LEN = 0).

Data frames also carry control information (piggybacking) and are numbered. The frame control information is stored in the field FCTR at the beginning of a frame.

7	6	5	4	3	2	1	0
FTYPE	SEQCTR		RFU	FRNR		ACKNR	

Field	Bits	Value	Description
FTYPE	7	0	Data frame including control information
		1	Control frame
SEQCTR	6:5	00	ACK: Acknowledge received frame. The frame number is defined by ACKNR

		01 10 11	NAK: No acknowledge for the frame defined by ACKNR. Reset frame counter for synchronization. Use only for control frames. RFU
RFU	4	x	undefined
FRNR	3:2	0-3	Frame number of the data frame. Not used for control frames: set to 00 in this case
ACKNR	1:0		Number of frame to be acknowledged / not acknowledged

Table 3-1 Definition of FCTR

A complete list of encoding of FCTR is shown in section 8.2 “Common encoding tables”.

To optimally use the physical bandwidth a sliding window algorithm is applied. The window size *WIN_SIZE* can be 1 up to 2 frames. This leads to 2* *WIN_SIZE* buffers of the maximum packet size MAX_PACKET_SIZE which have to be available on the Master and Slave (one half for transmission and the other for reception).

There are also *WIN_SIZE*+1 software timers necessary: *WIN_SIZE* for transmission (retransmit timer: time-out for frame repetition, if no acknowledge is received) and one additional for the receiver (acknowledge timer; in case no data frame is sent, carrying the acknowledge for the last received block, it triggers sending a control frame to acknowledge the last correct received block. To avoid elapsing the retransmit timer on the transceiver side this timer must elapse earlier).

Valid data frames are acknowledged by the receiver either by a control frame (if a data frame in the other direction is not available after a receiver time out) or a data frame.

In implementations with a window size of 2 frames up to one frame can be unacknowledged while another frame is sent in the same direction.

Example for *WIN_SIZE* = 2: consider frame n and n+1 are sent by the transmitter, but not acknowledged by the receiver yet. Before frame n+2 can be sent frame n (or n+1, see next example) has to be acknowledged.

The receiver always acknowledges all correct received frames.

Example for *WIN_SIZE* = 2: the last acknowledged frame is n-1. Now frame n and n+1 are sent, before frame n can be acknowledged. An acknowledge for frame n+1 is also valid for frame n.

A NAK is sent by the receiver if an invalid data frame is received (incorrect checksum or frame number is not within the window)

The frame n is repeated by the transmitter if a corresponding NAK is received from the other side or a time out for frame n.

Note: NAK is only for performance optimization, the protocol works also without. NAK should only be sent once for each frame and immediately after the error is detected (but not after receiver time-out).

There is a special control frame to reset (and synchronize) the frame counters of Master and Slave (set ACKNR to 0 for this frame).

3.3 Frame checksum

At the end of the packet data a 16-bit frame checksum (FCS) is added. The used checksum is a cyclic redundancy code (CRC). The standard CCITT generator polynomial:

$x^{16} + x^{12} + x^{5} + 1$

is used.

The 16-bit CRC checksum is able to detect:

all errors with odd bit number (like parity),

all double bit failures,

all failure burst of 16 bits or less and

most of longer failure bursts.

The order of bytes in the FCS filed is low byte || high byte.

Note: the CRC checksum with the above polynomial should not be used for more than 4093 bytes.

The checksum is calculated over the frame control field, the length and the packet data.

To avoid confusion regarding the CRC (e.g. bit order) a reference implementation of the checksum algorithm and also an optimized implementation are shown in Section 8.1.

3.4 Error handling

3.4.1 Receiver

Condition	Action
Layer 2 framing error	Send NAK: frame number is last correct frame received + 1
Layer 1 receive buffer overflow error	Send NAK: frame number is last correct frame received + 1
FCTR field SEQCTR or RFU has invalid value	Send NAK: frame number is last correct frame received + 1
Data frame number not expected	Send ACK: frame number is last correct frame received
Checksum incorrect	Send NAK: frame number is last correct frame received + 1
Control frame has invalid size	Frame is discarded
Data frame size has invalid value	Send NAK: frame number is last correct frame received + 1
Receive timer overflow	Send ACK: frame number is last correct frame received. This is no error condition.

Table 3-2 Error handling on receiver

If an error is detected a NAK is responded, the frame is discarded. The receiver waits for the next frame.

Note: A NAK is sent with the next data frame or a control frame. It is not delayed (e.g. by time-out of receive timer).

3.4.2 Transmitter

Condition	Action
NAK received	Retransmit frame specified by NAK
Unexpected NAK received	A NAK for an already acknowledged frame is received: frame is discarded
Unexpected ACK received	An ACK for an already acknowledged frame is received: frame is discarded
Transmission timer overflow for frame n	Re-transmit frame n

Table 3-3 Error handling on transmitter

3.5 Petri net models of frame handling

This section gives Petri net models of the interaction of a transmitter, receiver and an imperfect transmission medium concerning frame handling. The symbol definition is taken from **[ProtRep]**. For details about protocol modeling by Petri nets see also **[ProtRep]**.

3.5.1 Overview

Figure3-2 below shows the overall interaction between transmitter (left) and receiver (right) with the transmission medium (physical layer) in between. There are some restrictions / simplifications to reduce the complexity of the figure:

- data transfer in one direction only. As the protocol is symmetric the back channel is identical.
- no piggybacking, therefore no acknowledge timer.
- error handling: only loss of a complete frame modeled.
- no handling of corrupted frames (NAK).
- no handling of unsynchronized transmitter/receiver.
- More detailed sub models are described in the next sections.
- Places are illustrated by circles and correspond to states (transmitter and receiver) or messages (downlink and uplink). Transitions are depicted by thick lines, directed arcs connect places with transitions and vice versa.
- F0-F3 represent data frames with the corresponding frame numbers, ACK0-ACK3 the acknowledges for the corresponding frames (control frame or ACKNR field of a data frame in the opposite direction).
- PAKTR means that a new packet is available to be transmitted to the receiver, PAKRC stands for packet received.
- Places A0-A3 accept the corresponding data frame, S0-S3 mean that the corresponding data frame is sent and acknowledged correctly, W0-W3 wait for the corresponding data frame to be acknowledged.

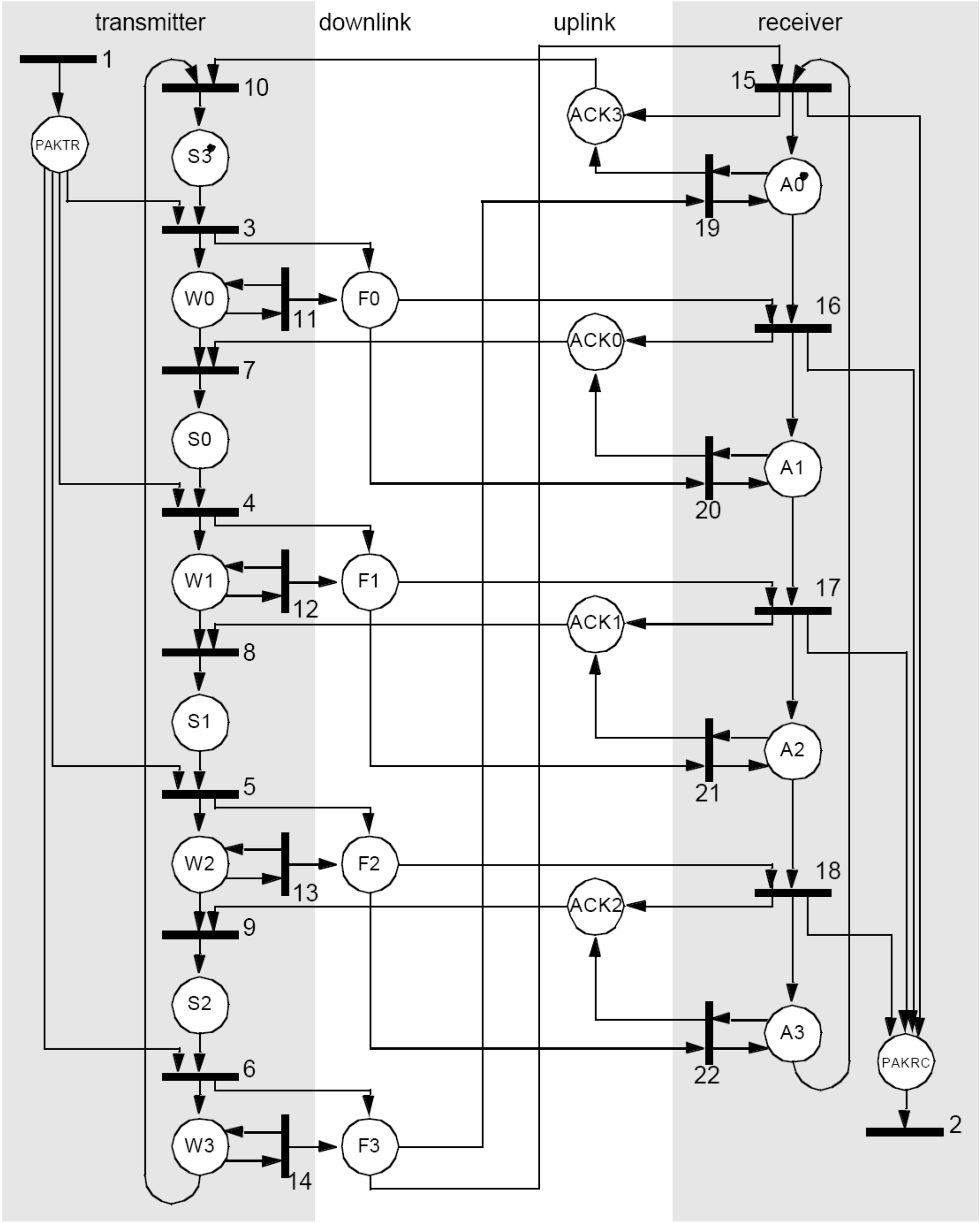


Figure 3-2 **Petri net model of frame handling (overview)**

Transition 1 is the interface to the next higher layer in the transmitter. It puts a token into PAKTR when a packet is ready for transmission. Transition 2 takes an error free received packet and passes it to layer 3 of the receiver.

Transitions 3-6 send a data frame and start the retransmit timer of the transmitter. Transitions 7-10 stop the retransmit timer and update the frame counter.

Transitions 11-14 represent an overflow of the retransmit timer: the frame is sent again and also timer is restarted. These transitions are used if either a data frame or an acknowledge frame got lost.

Transitions 15-18 update the frame counter in the receiver part and accept error free packets.

Last but not least transitions 19-22 acknowledge the previous frame in the case that the previous frame is already accepted, the acknowledge was sent, but dropped by the physical layer, and the frame is resent by a retransmit time-out.

The tokens in the places S3 and A0 mark the reset state.

3.5.2 Detailed model of transmitter

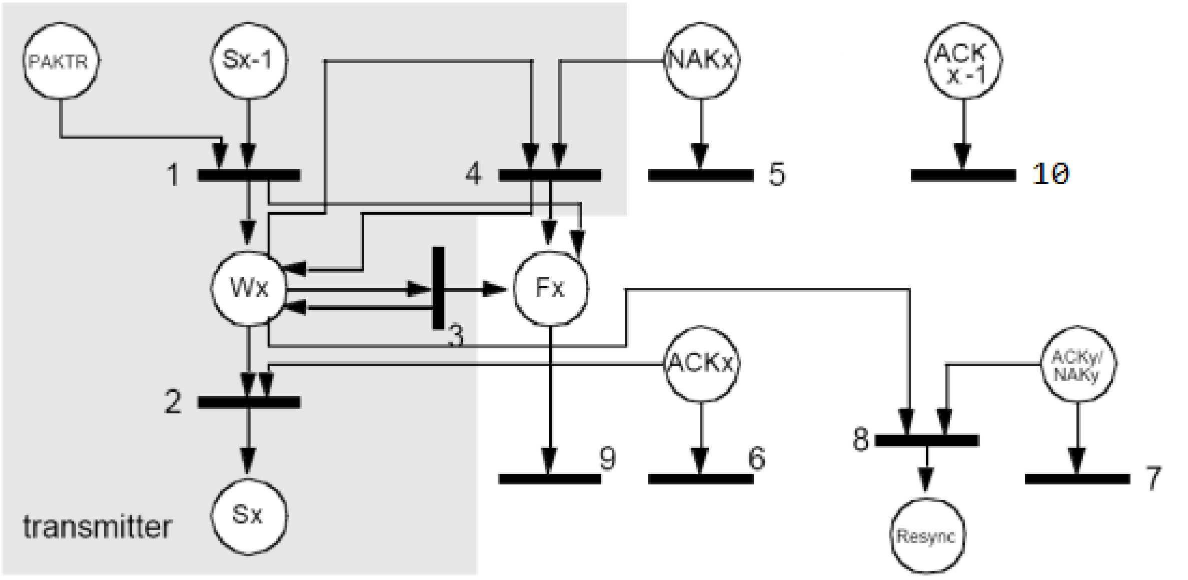


Figure 3-3 Detailed Petri net of transceiver

Figure 3-3 shows a section of the transmitter of Figure 3-2 in detail. It includes error handling. Possible errors are loss or corruption of ACK or NAK frames and unexpected ACK/NAK frames.

Loss or corruption of the data frame are handled by the receiver, see Section 0.

x stands for the frame number (0-3) currently processed, y for a different frame number, i.e. x!=y. Frame number arithmetic is done modulo 4.

Transitions 1-3 are already described in Section 3.5.1: transition 1 sends frame x, and starts the retransmit timer, transition 2 is triggered by a valid ACK of frame x and stops the retransmit timer. Transition 3 is triggered by a retransmit time out, frame x is sent again and the timer is restarted.

Transition 4 is triggered by a NAK frame for frame number x: the data frame x is sent again immediately. Also, the retransmit timer is restarted.

Transitions 5-7 represent loss or corruption of ACK or NAK frames. Corrupted but CRC correct control frames are simply discarded. A time out of the retransmit timer will force a new transmission of the data frame x followed by an ACK.

Transition 8: A valid ACK or NAK for an unexpected frame number can be used as a hint for unsynchronized transmitter and receiver frame counters. The synchronization (place resync) can be forced by a special control frame, see Section 3.2.

Transition 9 illustrate the loss of a data frame. The frame is transmitted again after time out of the retransmit timer.

Transition 10 illustrate the full duplex case when the new frame is transmitted and a new frame containing the already received ACK x-1 is received in parallel. No action is required.

3.5.3 Detailed model of receiver

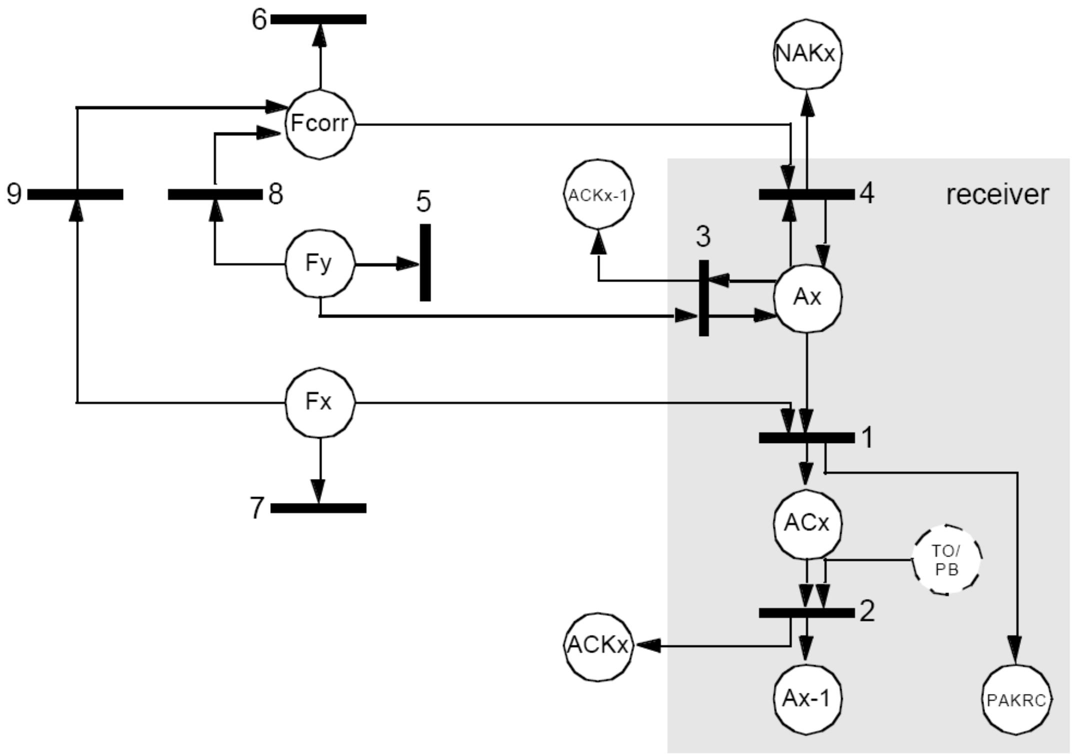


Figure 3-4 Detailed Petri net of receiver

Figure 3-4 shows a section of the receiver of Figure 3-2 in detail. It includes error handling and piggybacking. Possible errors are unexpected data frame numbers and loss or corruption of data frames.

The variables x and y are analog to the previous section.

The places A0-A3 in Figure 3-2 are split in two places: the first Ax accepts the corresponding frame x (also x-1, see below) and transfers the packet to the next higher layer in the protocol (transition 1). It also starts the acknowledge timer. ACx waits for sending the ACK for frame x.

Sending of the acknowledge depends on two different other conditions illustrated by the place TO/PB:

TO: time out of the acknowledge timer or

PB: an event triggered by the next higher layer either to request the transmission of a data frame in the opposite direction (piggybacking) or to request the immediate transmission of the acknowledge for the received frame through a control frame (e.g. if the first or an intermediate frame inside a transport layer chain was received and data in the opposite direction is not expected).

In the first case the ACK is sent as a control frame, in the second the ACK is sent either together with a data frame (if data muss be sent in the opposite direction) or as a control frame (if the immediate acknowledgement of the received frame was requested by the next higher layer). In both cases the acknowledge timer is stopped (transition 2).

Transition 3 handles the response of the receiver to an unexpected data frame: an acknowledge for frame x-1 is sent. If y==x-1 (mod 4) the acknowledge enables the transmitter to go on if a previous ACKx-1 was dropped by the line (see also Section 3.5.1) otherwise it can be a hint for the transmitter that the frame counter of transmitter and receiver are not synchronized, see Section 3.5.2.

A corrupted data frame (see Table 3-2 for detection of corrupted frames) is answered by a NAK for frame x (transition 4). Note that the frame number of the corrupted frame may be also incorrect. NAKx can be a hint for the transmitter that the frame counter of transmitter and receiver are not synchronized, see Section 3.5.2.

Loss of data frames is actually handled by the transmitter: the frame is transmitted again after a time out.

Transitions 5-7 illustrate loss of data frames, transitions 8 and 9 represent the corruption of data frames.

4 Network Layer

The purpose of the network layer is to route packets to different channels. A packet consists of the packet control byte (PCTR) and up to MAX_PACKET_SIZE data bytes. The PCTR (bits 7:4) is RFU and set to zero.

4.1 Channels

The first byte of the packet data is the packet control byte (PCTR). Channel information is coded in the upper bits of PCTR.

7	6	5	4	3	2	1	0
CHAN				not used by network layer			

Field	Bits	Value	Description
CHAN	7:4	0	Default channel to route the packet data to. This value is set in case just a single channel is used.
		1-15	Optional channels. Not valid in case of a single channel “network” connection

Table 3-4 Definition of PCTR regarding the network layer

5 Transport Layer

As there is a point to point connection between a master and a slave the data link layer takes care that packages are received in the same order as sent, the only function of the transport layer in this protocol is packet chaining.

5.1 Chaining

To simplify chaining a bit-field in the PCTR is used to mark the first, intermediate and last packet of a multi-packet APDU. APDUs smaller than MAX_PACKET_SIZE bytes are transmitted with bit-field CHAIN cleared (no chaining).

For longer APDUs a chaining method is used: the APDU is split in consecutive packets. All packets except the last must have a size of MAX_PACKET_SIZE bytes (including PCTR) and the value 001 (first) or respectively 010 (intermediate) for CHAIN. The last packet has a packet size of 2 to MAX_PACKET_SIZE bytes and the value 100 for CHAIN.

7	6	5	4	3	2	1	0
not used by transport layer					CHAIN		

Field	Bits	Value	Description
CHAIN	2:0	000	No chaining. Single frame.
		001	First packet of a chain
		010	Intermediate packet(s) of a chain.
		100	Last packet in a chain
		111	Chaining error

Table 5-1 Definition of PCTR regarding the transport layer

5.2 Error handling

There are three different errors possible in this layer:

- CHAIN has the value 001 or 010, but packet size != MAX_PACKET_SIZE bytes.
- CHAIN has value 000 or 001, but the previous APDU is not entirely received yet (can be detected by value 100 for CHAIN).
- A packet is smaller than MAX_PACKET_SIZE bytes, but the previous APDU is not entirely received yet (can be detected by value 100 for CHAIN).

In all cases the receiver sends a single byte packet (consisting of PCTR only) to the corresponding channel of the transmitter with the value 111 in field CHAIN (signaling an error in the CMD field is recommended). The transmitter can repeat the chain (if the information is still available) or pass the error to a higher level.

Note: There are only two sources of errors in this layer:

- wrong implementation of the protocol or

- an error in a lower layer which is not detected there (e.g. frame is corrupt, but checksum is still ok).

In the second case the error can also be passed to the data link layer.

6 Presentation Layer

The Presentation Layer provides secure channel capabilities. This layer is optional and is only present in case a master/slave network channel requires confidentiality and/or integrity services. Once the presentation layer is enabled (PRESENCE bit is set in PCTR), the presentation layer becomes mandatory for both directions “master to slave” and “slave to master” for all network channel until the next reset of the device.

6.1 Presence Indication

The bit 3 of the PCTR is indicating the absence or presence of the Presentation Layer as shown in Table 6-1. Only the first packet in a chain is indicating the presence of the Presentation Layer. The PRESENCE flag of all other packets is not considered (don’t care).

7	6	5	4	3	2	1	0
not used by presentation layer				PRESENCE	not used by presentation layer		

Field	Bits	Value	Description
PRESENCE	3	0	Presentation Layer absent
		1	Presentation Layer present

Table 6-1 Definition of PCTR regarding the presentation layer

6.1.1 Presentation Layer Abbreviations

SCTR	**S**ecurity **C**on**tr**ol
PVER	Supported/chosen **p**rotocol **ver**sion
MasterEncKey	**Master Enc**ryption **key**
MasterEncNonce	**Nonce** for **Master Enc**ryption
MCTXT	**M**aster calculated **c**ipher**t**e**xt**
MPUBKEY	**M**aster **Pub**lic **Key**
MRND	**M**aster **R**a**nd**om Number
MSEQ	**M**aster **Seq**uence number
RND	**R**a**nd**om Number
SCTXT	**S**lave calculated **c**ipher**t**e**xt**
SlaveEncKey	**Slave Enc**ryption **key**
SlaveEncNonce	**Nonce** for **Slave Enc**ryption
SRND	**S**lave **R**a**nd**om Number
SSEQ	**S**lave **Seq**uence number

6.2 Security Control Byte

The SCTR byte controls the security services of the Presentation Layer. Those services are:

- Establishing of a secure channel (Handshake).
- Protected communication transparent for the Application Layer.
- Managing the session context (e.g. save/ restore).
- Sending alert messages.

7	6	5	4	3	2	1	0
PROTOCOL			MESSAGE			PROTECTION	

Field	Bits	Value	Description
PROTOCOL	7:5	000	Handshake
		001	Record Exchange
		010	Alert
		011	Manage Context
		100-111	RFU
MESSAGE	4:2		Handshake
		000	Hello
		001	Key Agreement
		010	Finished
		011-111	RFU
			Record Exchange
		000-111	RFU
			Alert
		000	Fatal Error[1]
		001	Integrity violated[2]
		010-111	RFU
			Manage Context
		000	Save context
		001	Context saved
		010	Restore context
		011	Context restored
		100-111	RFU
PROTECTION	1:0		All message types
		x0	Master Payload unprotected
		x1	Master Payload protected
		0x	Slave Payload unprotected
		1x	Slave Payload protected

Table 6-2 Definition of SCTR

[1] Re-establishing of the secure channel is required.

[2] The Application Layer shall retransmit the regarded command or response max. TRANS_REPEAT times. Upon retransmitting the regarded sequence number (MSEQ / SSEQ) gets increased. The ultimate solution is re-establishing the secure channel.

6.3 Handshake messages

6.3.1 Supported protocol versions

The presentation layer supports multiple protocol versions in terms of message format and cryptographic schemes defined for it. The master is announcing its supported versions as part of the handshake message and the slave returns the version it has chosen. The version (PVER) is defined as a bit field, where each bit is representing a dedicated protocol version (e.g. version 1 = 0000 0001; version 1 & 2 = 0000 0011).

PVER (bit)	Key derivation scheme	Key agreement scheme	Authentication scheme	Communication protection scheme
0	TLS PRF SHA256[1]	Pre-shared secret[2]	AES128-CCM8[3]	AES128-CCM8
1	TLS PRF SHA256	ECDHE with pre-generated[4] ECC NIST P256 key pair	AES128-CCM8	AES128-CCM8
2-7	RFU	RFU	RFU	RFU

Table 6-3 List of by presentation layer supported protocol versions

6.3.2 Key derivation scheme details TLS PRF SHA256 for AES128-CCM8

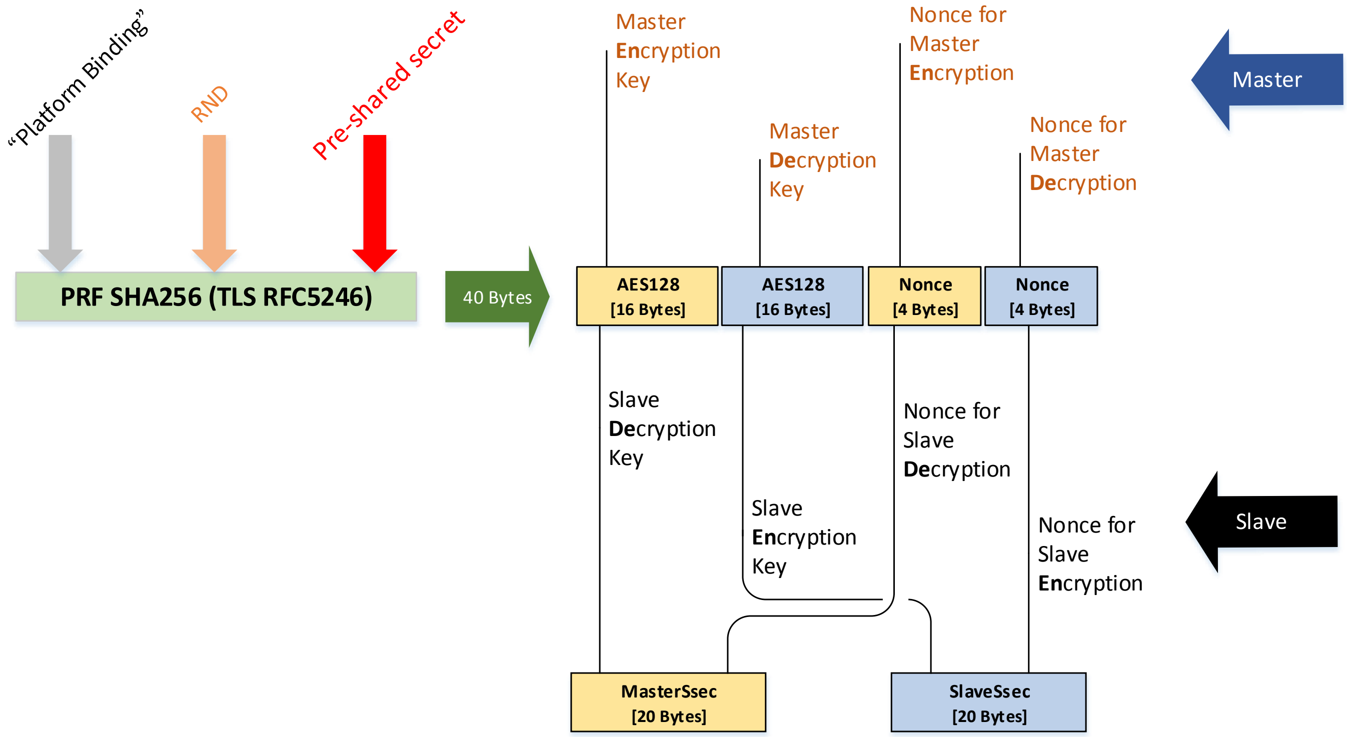


Figure 6-1 Key derivation scheme details AES128-CCM8

[1] TLS PRF SHA256 according to [RFC5246]

[2] During pairing a shared secret gets generated and kept in non-volatile memory at master and slave for later use.

[3] According to [SP800-38C] Section 6.1 and 6.2

[4] During pairing the slave device generates a key pair from which the private key is kept in its non-volatile memory and the public key gets exported to the master for non-volatile storage and later use.

6.3.3 AES128-CCM8 based finished message ciphertext calculation

Figure 6-2 and Figure 6-3 illustrating the cryptogram (output) calculation[1] details at master and slave side.

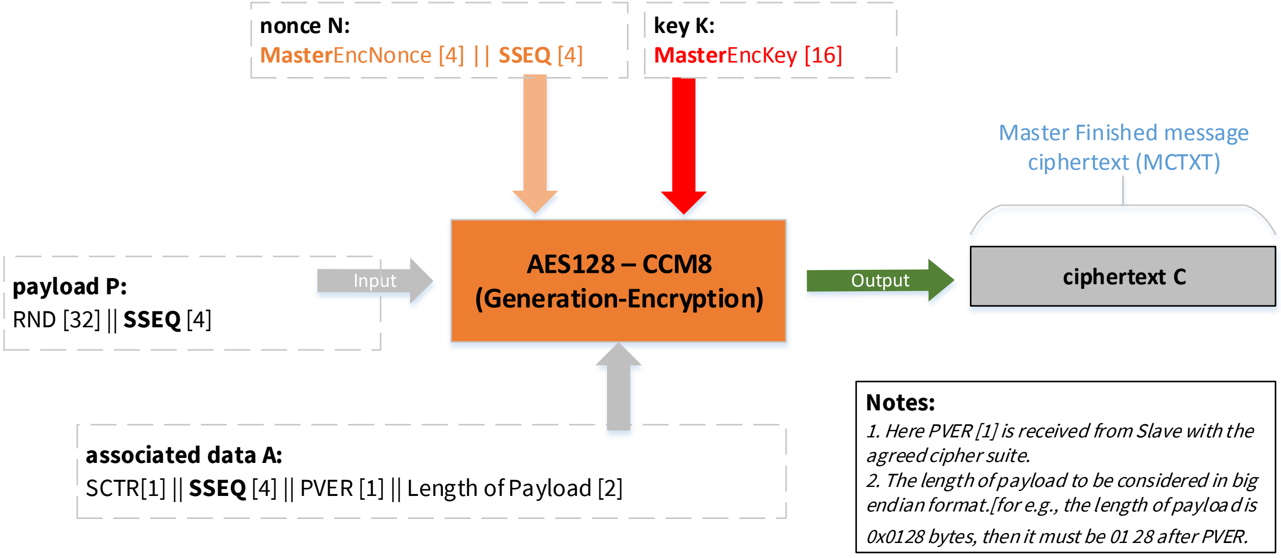

nonce N:
MasterEncNonce [4] || SSEQ [4]
key K:
MasterEncKey [16]
Master Finished message ciphertext (MCTXT)
payload P:
RND [32] || SSEQ [4]
Input
AES128 – CCM8 (Generation-Encryption)
Output
ciphertext C
associated data A:
SCTR[1] || SSEQ [4] || PVER [1] || Length of Payload [2]
Notes:
1. Here PVER [1] is received from Slave with the agreed cipher suite.
2. The length of payload to be considered in big endian format.[for e.g., the length of payload is 0x0128 bytes, then it must be 01 28 after PVER.

Figure 6-2 Finished message ciphertext calculation (acc. AES128-CCM8) details at master

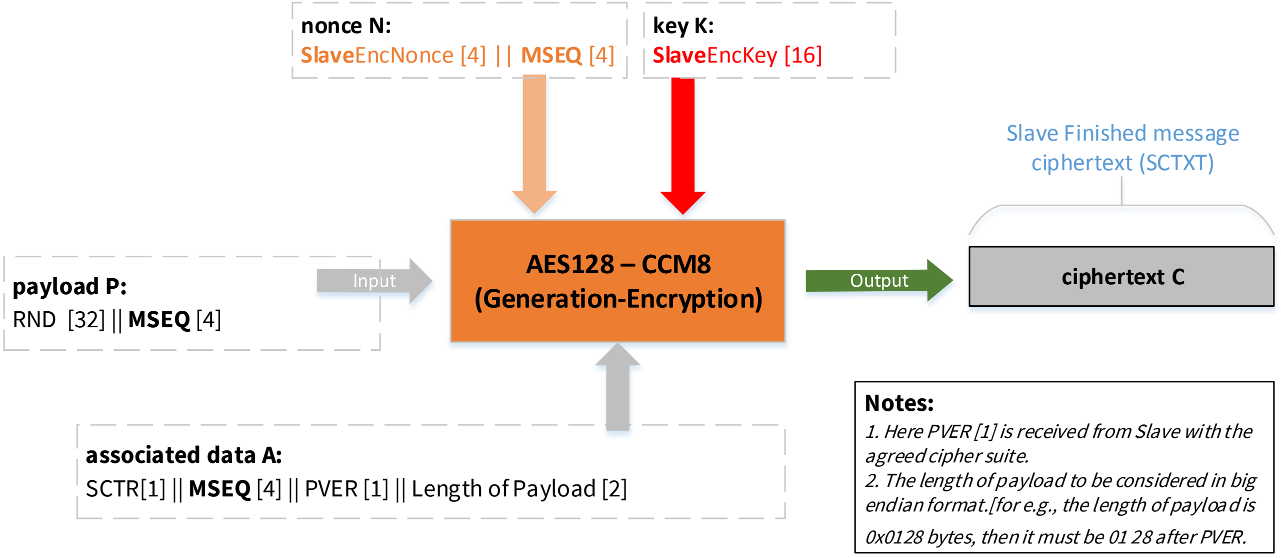

nonce N:
SlaveEncNonce [4] || MSEQ [4]
key K:
SlaveEncKey [16]
Slave Finished message ciphertext (SCTXT)
payload P:
RND [32] || MSEQ [4]
Input
AES128 – CCM8 (Generation-Encryption)
Output
ciphertext C
associated data A:
SCTR[1] || MSEQ [4] || PVER [1] || Length of Payload [2]
Notes:
1. Here PVER [1] is received from Slave with the agreed cipher suite.
2. The length of payload to be considered in big endian format.[for e.g., the length of payload is 0x0128 bytes, then it must be 01 28 after PVER.

Figure 6-3 Finished message ciphertext calculation (acc. AES128-CCM8) details at slave

[1] According to [SP800-38C] Section 6.1

6.3.4 AES128-CCM8 based Record Exchange Payload Encryption

Figure 6-4 and Figure 6-5 illustrating the payload encryption details[1] at master and slave side.

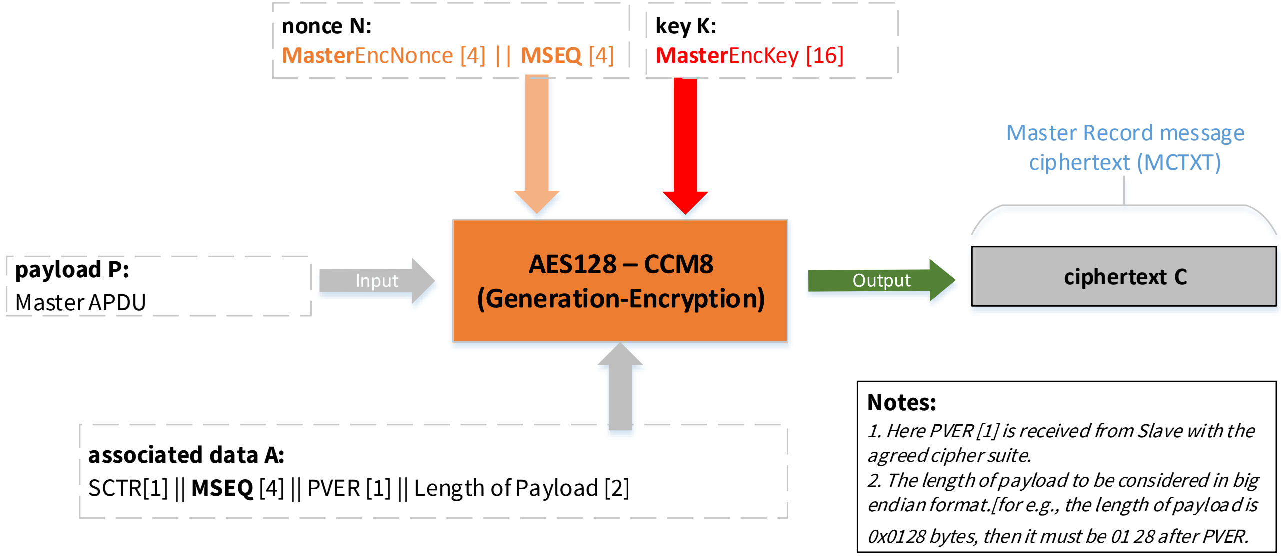

nonce N:
MasterEncNonce [4] || MSEQ [4]
key K:
MasterEncKey [16]
Master Record message ciphertext (MCTXT)
payload P:
Master APDU
Input
AES128 – CCM8 (Generation-Encryption)
Output
ciphertext C
associated data A:
SCTR[1] || MSEQ [4] || PVER [1] || Length of Payload [2]
Notes:
1. Here PVER [1] is received from Slave with the agreed cipher suite.
2. The length of payload to be considered in big endian format.[for e.g., the length of payload is 0x0128 bytes, then it must be 01 28 after PVER.

Figure 6-4 Payload ciphertext calculation (acc. AES128-CCM8) details at master

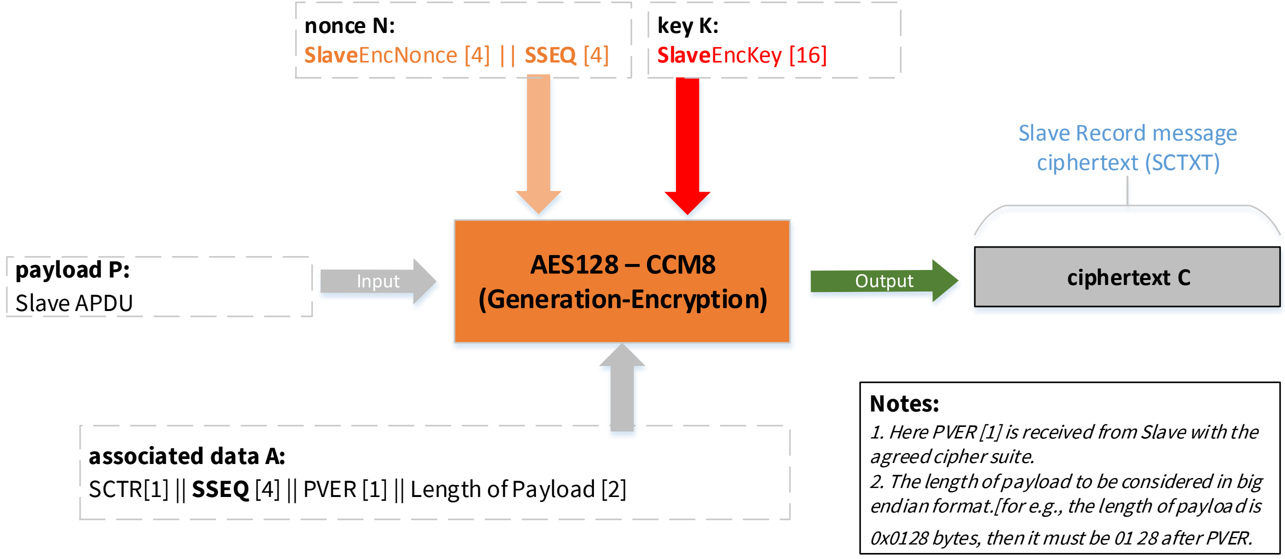

nonce N:
SlaveEncNonce [4] || SSEQ [4]
key K:
SlaveEncKey [16]
Slave Record message ciphertext (SCTXT)
payload P:
Slave APDU
Input
AES128 – CCM8 (Generation-Encryption)
Output
ciphertext C
associated data A:
SCTR[1] || SSEQ [4] || PVER [1] || Length of Payload [2]
Notes:
1. Here PVER [1] is received from Slave with the agreed cipher suite.
2. The length of payload to be considered in big endian format.[for e.g., the length of payload is 0x0128 bytes, then it must be 01 28 after PVER.

Figure 6-5 Payload ciphertext calculation (acc. AES128-CCM8) details at slave

6.3.5 Shared secret based version (xxxx xxx1).

In case of the shared secret based handshake, both the master and the slave sharing the same secret, which was agreed on during pairing of both entities. This pairing usually takes place during PCB production and is a pre-condition for the shared secret based handshake specified in this chapter.

Sequence of functions:

1. The master sends the supported protocol version(s) to the slave with SCTR = 0x00 (Handshake.Hello). The protocol state gets cleared and re-started in case the protocol was already started.
2. The slave generates a random value (RND) and SSEQ and returns both together with the chosen protocol version with SCTR = 0x00 (Handshake.Hello).

[1] According to [SP800-38C] Section 6.1

3. Master and slave calculating the session secrets, one for master (MasterSsec) and one for slave (SlaveSsec) (MasterSsec and SlaveSsec = **TLS PRF**[1] ("Platform Binding", RND || Shared Secret[2])). For details refer to chapter 6.3.2.
4. The master uses SSEQ and generates the ciphertext (Generation-Encryption Process[3]; refer to Figure 6-2) and sends both to the slave with SCTR = 0x08 (Handshake.Finished).
5. The slave decrypts the ciphertext (Decryption-Verification Process[4]) and verifies the payload whether the expected values were received and upon success saves the received sequence number (SSEQ) for further use.
6. The slave generates the MSEQ and the ciphertext (Generation-Encryption Process; refer to Figure 6-3) and sends both to the master with SCTR = 0x08 (Handshake.Finished).
7. The master decrypts the ciphertext (Decryption-Verification Process) and verifies the payload whether the expected values were received and upon success saves the received sequence number (MSEQ) for further use.
8. In case both ciphertext validations were successful, the cryptographic link is established between master and slave, which is the pre-condition for a subsequent protected record exchange. In all other cases the handshake must be re-started at step 1

Master message (step 1)

1 byte	1 byte
SCTR	PVER
Hello (0x00)	xxxx xxx1

⇔

Slave message (step 2)

1 byte	1 byte	32 bytes	4 bytes
SCTR	PVER	RND	SSEQ
Hello (0x00)	0000 0001	NONCE	0x00...00- 0x0000FFFF

Master message (step 4)

1 byte	4 bytes	44 bytes
SCTR	SSEQ	MCTXT
Finished (0x08)	0x00...00- 0x0000FFFF	0x00-0xFF

⇔

Slave message (step 6)

1 byte	4 bytes	44 bytes
SCTR	MSEQ	SCTXT
Finished (0x08)	0x00...00- 0x0000FFFF	0x00-0xFF

Figure 6-6 Structure of handshake (shared secret based) messages

6.3.6 ECDHE based version (xxxx xx1x)

In case of the ECDHE based handshake, both the master and the slave sharing parts of an ECC 256 key pair, which was generated during pairing of both entities. This pairing usually takes place during PCB production and is a pre-condition for the ECDHE based handshake specified in this chapter. During the pairing process the slave generates an ECC NISTP256 key pair and exports the public key to the master. For later use with the handshake protocol, both entities are storing the key components in non-volatile memory.

Sequence of functions:

1. The master generates a random value (MRND) and sends it together with the supported protocol version(s) to the slave with SCTR = 0x00 (Handshake.Hello). The protocol state gets cleared and re-started in case the protocol was already started.

[1] TLS PRF SHA256 according to [RFC5246].

[2] Which particular pre-shared secret to use MUST be implicitly agreed out-of-band and is outside the scope of this specification.

[3] According to [SP800-38C] Section 6.1

[4] According to [SP800-38C] Section 6.2

2. The slave generates a random value (SRND) and returns it together with the chosen protocol version with SCTR = 0x00 (Handshake.Hello). Both master and slave constructing RND from MRND||SRND.
3. The Master generate an ephemeral key pair and sends the public key to the slave with SCTR = 0x04 (Handshake.KeyAgreement).
4. Both entities calculating a Shard Secret according Elliptic Curve Diffie-Hellman key exchange protocol (ECDH[1]) and the slave returns with SCTR = 0x04 (Handshake.KeyAgreement) indicating success.
5. Master and slave calculating the session secrets, one for master (MasterSsec) and one for slave (SlaveSsec) messages (MasterSsec and SlaveSsec = **TLS PRF**[2] ("Platform Binding", RND || Shared Secret)). For details refer to chapter 6.3.2.
6. The master randomly chose SSEQ and generates the ciphertext (Generation-Encryption Process[3]; refer to Figure 6-2) and sends both to the slave with SCTR = 0x08 (Handshake.Finished).
7. The slave decrypts the ciphertext (Decryption-Verification Process[4]) and verifies the payload whether the expected values were received and upon success saves the received sequence number (SSEQ) for further use.
8. The slave randomly chose the MSEQ and generates the ciphertext (Generation-Encryption Process; refer to Figure 6-3) and sends both to the master with SCTR = 0x08 (Handshake.Finished).
9. The master decrypts the ciphertext (Decryption-Verification Process) and verifies the payload whether the expected values were received and upon success saves the received sequence number (MSEQ) for further use.
10. In case both ciphertext validations were successful, the cryptographic link is established between master and slave, which is the pre-condition for a subsequent protected record exchange. In all other cases the handshake must be re-started at step 1

[1] Which particular pre-shared public key pair to use MUST be implicitly agreed out-of-band and is outside the scope of this specification.

[2] TLS PRF SHA256 according to [RFC5246].

[3] According to [SP800-38C] Section 6.1

[4] According to [SP800-38C] Section 6.2

Master message (step 1)

1 byte	1 byte	16 bytes
SCTR	PVER	MRND
Hello (0x00)	xxxx xx1x	NONCE

⇔

Slave message (step 2)

1 byte	1 byte	16 bytes
SCTR	PVER	SRND
Hello (0x00)	0000 0010	NONCE

Master message (step 3)

1 byte	n bytes
SCTR	MPUBKEY
Key Agreement (0x04)	DER format

⇔

Slave message (step 4)

1 byte
SCTR
Key Agreement (0x04)

Master message (step 6)

1 byte	4 bytes	44 bytes
SCTR	SSEQ	MCTXT
Finished (0x08)	0x00...00-0x0000FFFF	0x00-0xFF

⇔

Slave message (step 8)

1 byte	4 bytes	44 bytes
SCTR	MSEQ	SCTXT
Finished (0x08)	0x00...00-0x0000FFFF	0x00-0xFF

Figure 6-7 Structure of handshake (ECDHE based) messages

6.4 Record Exchange messages

6.4.1 Protected Payload

In case the session keys are successfully negotiated the payload conveyed by the protocol stack could be protected as defined by the “Record Exchange” type of the presentation layer. Both entities starting with the message sequence number (xSEQ either MSEQ or SSEQ) as agreed on during handshake. The sequence number gets increase strong monotonic (xSEQ = xSEQ+1) with each protected message. The acceptance window for sequence numbers is of size TRANS_REPEAT (last received xSEQ+TRANS_REPEAT). In case the sequence number is out of window or wraps around the secure channel needs to be re-established (refer to Handshake messages).

The payload protection algorithm is defined for each protocol version in Table 6-3 (Column: Communication protection scheme). Depending on the chosen algorithm the length of the protected payload might vary, due to additional blocks of encrypted data (initial values, padding, etc.) and additional MAC values for data integrity.

Master message
(Payload protected)

1 byte	4 bytes	n bytes
SCTR	MSEQ	MCTXT
Record Exchange (0x23)	0x00...00-0xFF…FF	0x00-0xFF

⇔

Slave message
(Payload protected)

1 byte	4 bytes	n bytes
SCTR	SSEQ	SCTXT
Record Exchange (0x23)	0x00...00-0xFF…FF	0x00-0xFF

Master message
(Payload unprotected)

1 byte	n bytes
SCTR	Payload plain
Record Exchange (0x22)	0x00-0xFF

⇔

Slave message
(Payload protected)

1 byte	4 bytes	n bytes
SCTR	SSEQ	SCTXT
Record Exchange (0x22)	0x00...00-0xFF…FF	0x00-0xFF

Master message
(Payload protected)

1 byte	4 bytes	n bytes
SCTR	MSEQ	MCTXT
Record Exchange (0x21)	0x00...00-0xFF…FF	0x00-0xFF

⇔

Slave message
(Payload unprotected)

1 byte	n bytes
SCTR	Payload plain
Record Exchange (0x21)	0x00-0xFF

Master message
(Payload unprotected)

1 byte	n bytes
SCTR	Payload plain
Record Exchange (0x20)	0x00-0xFF

⇔

Slave message
(Payload unprotected)

1 byte	n bytes
SCTR	Payload plain
Record Exchange (0x20)	0x00-0xFF

Figure 6-8 Structure of a record exchange messages

6.5 Manage Context

In case the slave goes to power down and the session context needs to be preserved, to be used after powering up again, the master requests saving and finally restoring the session context, containing the master and slave credentials and sequence numbers. The session context allows seamlessly continuing the presentation layer protocol after e.g. powering up with reset.

In case the handshake is started but not finalized the context gets flushed and the establishment of the session context must be re-started with the handshake master Hello message.

As soon as the restore is executed or a handshake hello message is sent across, the saved context gets flushed.

Master message
(Save context)

1 byte
SCTR
Save context (0x60)

⇔

Slave message
(Context saved)

1 byte
SCTR
Context saved (0x64)

Master message
(Restore context)

1 byte	4 bytes
SCTR	SSEQ
Restore context (0x68)	Saved Slave Sequence Number

⇔

Slave message
(Context restored)

1 byte	4 bytes
SCTR	MSEQ
Context restored (0x6C)	Saved Master Sequence Number

Figure 6-9 Structure of manage context messages

7 Application Layer

The Application Layer is defined by each particular Application and consists of the Application Data Unit (APDU; e.g. ISO7816 APDU).

One other example is the standard IFX APDU for embedded security applications. Table 7-1 to Table 7-3 are showing the APDU structure and generic field description of it:

Field	Description
Cmd	Command Code
Sta	Response status code
Param	Parameter to control major variations of a command.
UnDef	Undefined value (contains any value 0x00-0xFF)
Len	Length of the data section of the APDU
Data	Data section of the APDU

Table 7-1 APDU Fields

Field	Cmd (1 Byte)	Param (1 Byte)	Len (2 Bytes)	Data (Len Bytes)
Offset	0	1	2	4

Table 7-2 Command APDU

Field	Sta (1 Byte)	UnDef (1 Byte)	Len (2 Bytes)	Data (Len Bytes)
Offset	0	1	2	4

Table 7-3 Response APDU

8 Appendixes

8.1 CRC implementations

8.1.1 Reference implementation

```
unsigned short calcCRC(unsigned short seed[1], unsigned char c)
{
        for(int i = 0;i<8;i++)
        {
                if((seed ^ c) & 0x01)
                        seed = (seed >> 1) ^ 0x8408;
                else
                        seed >>= 1;
                c >>= 1;
        }
        return seed;
}
```

8.1.2 More efficient implementation

The algorithm below is faster than the reference implementation and more memory efficient than a table lookup algorithm.

```
unsigned short calcFast(unsigned short seed[1], unsigned char c)
{
        unsigned int h1,h2,h3,h4;
        h1 = (seed ^ c) & 0xFF;
        h2 = h1 & 0x0F;
        h3 = (h2 << 4) ^ h1;
        h4 = h3 >> 4;
        return ((((h3 << 1) ^ h4) << 4) ^ h2) << 3) ^ h4 ^ (seed >> 8);
}
```

8.1.3 Table lookup implementation

For a table lookup implementation, see [RFC1662].

Note: The table has a size of at least 512 bytes.

[1] The initial seed is 0.

8.2 Common encoding tables

8.2.1 Frame control

FCTR	Description
0x00	Data frame 0, ACK for frame 0
0x01	Data frame 0, ACK for frame 1
0x02	Data frame 0, ACK for frame 2
0x03	Data frame 0, ACK for frame 3
0x04	Data frame 1, ACK for frame 0
0x05	Data frame 1, ACK for frame 1
0x06	Data frame 1, ACK for frame 2
0x07	Data frame 1, ACK for frame 3
0x08	Data frame 2, ACK for frame 0
0x09	Data frame 2, ACK for frame 1
0x0a	Data frame 2, ACK for frame 2
0x0b	Data frame 2, ACK for frame 3
0x0c	Data frame 3, ACK for frame 0
0x0d	Data frame 3, ACK for frame 1
0x0e	Data frame 3, ACK for frame 2
0x0f	Data frame 3, ACK for frame 3
0x10-0x1f	Not used
0x20	Data frame 0, NAK for frame 0
0x21	Data frame 0, NAK for frame 1
0x22	Data frame 0, NAK for frame 2
0x23	Data frame 0, NAK for frame 3
0x24	Data frame 1, NAK for frame 0
0x25	Data frame 1, NAK for frame 1
0x26	Data frame 1, NAK for frame 2
0x27	Data frame 1, NAK for frame 3
0x28	Data frame 2, NAK for frame 0
0x29	Data frame 2, NAK for frame 1
0x2a	Data frame 2, NAK for frame 2

FCTR	Description
0x2b	Data frame 2, NAK for frame 3
0x2c	Data frame 3, NAK for frame 0
0x2d	Data frame 3, NAK for frame 1
0x2e	Data frame 3, NAK for frame 2
0x2f	Data frame 3, NAK for frame 3
0x30-0x7f	Not used
0x80	Control frame 0, ACK for frame 0
0x81	Control frame 0, ACK for frame 1
0x82	Control frame 0, ACK for frame 2
0x83	Control frame 0, ACK for frame 3
0x84-0x9f	Not used
0xa0	Control frame 0, NAK for frame 0
0xa1	Control frame 0, NAK for frame 1
0xa2	Control frame 0, NAK for frame 2
0xa3	Control frame 0, NAK for frame 3
0xa4-0xbf	Not used
0xc0	Reset frame counters
0xc1-0xff	Not used

Table 8-1 Encoding of FCTR

8.3 Protocol variations

To fit best to application specific requirements the protocol might be tailored by specifying a couple of parameters. For each implementation of the protocol those parameters must be defined within the products interface specification, which refers to this document.

Parameter	Description
MAX_PACKET_SIZE	Maximum packet size accepted by the receiver. The protocol limits this value to 0xFFFF, but there might be project specific requirements to reduce the transport buffers size for the sake of less RAM footprint in the communication stack. If shortened, it could be statically defined or negotiated at the physical layer.
WIN_SIZE	Window size of the sliding windows algorithm see Section 3.2. The value

Parameter	Description
	could be 1 up to 2.
MAX_NET_CHAN	Maximum number of network channels. The value could be 1 up to 16. One indicates the OSI Layer 3 is not used and the CHAN field of the PCTR must be set to 0000.
CHAINING	Chaining on the transport layer is supported (TRUE) or not (FALSE)
TRANS_TIMEOUT	(Re)transmission timeout specifies the number of milliseconds to be elapsed until the transmitter considers a frame transmission is lost and retransmits the non-acknowledged frame. The Timer gets started as soon as the complete frame is transmitted. The value could be 1 up to 1000. However, as higher the number as longer does it take to recover from a frame transmission error. *Note: The acknowledge timeout on the receiver side must be shorter than the retransmission timeout to avoid unnecessary frame repetitions.*
TRANS_REPEAT	Number of transmissions to be repeated until the transmitter considers the connection is lost and starts a re-synchronization with the receiver. The value could be 1 up to 4.
BASE_ADDR	I2C (base) address. This address could be statically defined or dynamically negotiated by the physical layer. If not different specified the default value is 0x20.
MAX_SCL_FREQU	Maximum SCL clock frequency in KHz.
GUARD_TIME	Minimum time to be elapsed at the I2C master measured from read data (STOP condition) until the next write data (Start condition) is allowed to happen. If not different specified the default value is 500µs. *Note 1: For two consecutive accesses on the same device GUARD_TIME re-specifies the value of t_{BUF} as specified by [I2Cbus].* *Note 2: Even if another I2C address is accessed in between GUARD_TIME has to be respected for two consecutive accesses on the same device.*
SOFT_RESET	Any write attempt to the SOFT_RESET register will trigger a warm reset (reset w/o power cycle). This register is optional and its presence is indicated by the I2C_STATE register's "SOFT_RESET" flag.
PRESENT_LAYER	This flag at the I2C_STATE register indicates the optional availability of the presentation layer, which is providing confidentiality and integrity protection of payloads (APDUs) transferred across the I2C interface.

Table 8-2 List of protocol variations

8.4 Change History

Version	Date	Description
1.65	27-Jun-2017	Migrated to new document template
2.00	29-Jun-2018	• Chapter "Presentation Layer" revised. • I2C_State definition extended by presentation layer flags.
2.02	03-Oct-2018	Chapter Presentation Layer sentence "Once the presentation layer is enabled (PRESENCE bit is set in PCTR), the presentation layer becomes mandatory for both directions “master to slave” and “slave to master” for all network channel until the next reset of the device." added.
2.03	22-Sep-2020	Template adaptations (last page) and editorial changes

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-09-22

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: CSSCustomerService@infineon.com

Document reference

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.