2.5.2.1 MAC data service requirements

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be included in the MAC frame header.

The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the first hop in the path back to the originator of the corresponding route request command frame. The destination PAN identifier shall be the same as the PAN identifier of the originator.

The source MAC address and PAN identifier shall be set to the address and PAN identifier of the device sending the route reply command, which may or may not be the device from which the command originated.

The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. The transmission options shall be set to require acknowledgment. The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

2.5.2.2 NWK header fields

In order for this route reply to reach its destination and for the route discovery process to complete correctly, the following information must be provided.

The frame type subfield of the NWK frame control field should be set to indicate that this frame is a NWK layer command frame.

The destination address field in the NWK header shall be set to the network address of the first hop in the path back to the originator of the corresponding route request.

The source address in the NWK header shall be set to the NWK 16-bit network address of the device that is transmitting the frame.

2.5.2.3 NWK payload fields

The NWK frame payload contains a command identifier field, a command options field, the route request identifier, originator and responder addresses and an up-to-date summation of the path cost.

The command frame identifier shall contain the value indicating a route reply command frame.

2.5.2.3.1 Command options field

The format of the 8-bit command options field is shown in Figure 43.

Bit: 0—6	7
Reserved	Route repair

Figure 43 Route reply command options field

The route repair sub-field is a single-bit field. It shall have a value of 1 if and only if the route request command frame is being generated as part of a route repair operation for mesh network topology (see sub-clause 2.7.3.5.1).

2.5.2.3.2 Route request identifier

The request identifier field shall be set to the request identifier value from the corresponding route request command frame.

2.5.2.3.3 Originator address

The originator address field shall be 2 octets in length and shall contain the 16-bit network address of the originator of the route request command frame to which this frame is a reply.

2.5.2.3.4 Responder address

The responder address field shall be 2 octets in length and shall contain the 16-bit network address of the device for whom the route is being discovered. The value in this field shall always be the same as the value in the destination address field of the corresponding route request command frame.[111]

2.5.2.3.5 Path cost

The path cost field is used to sum link cost as the route reply command frame transits the network (see sub-clause 2.7.3.4.3).

2.5.3 Route error command

A device uses the route error command when it is unable to forward a data frame. The command notifies the source device of the data frame about the failure in forwarding the frame. The payload of a route error command shall be formatted as illustrated in Figure 44.

Octets: 1	1	2
Command frame identifier (see Table 129)	Error code	Destination address

Figure 44 Route error command frame format

2.5.3.1 MAC data service requirements

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be provided.

The destination MAC address and PAN identifier shall be set to the address and PAN identifier, respectively, of the first hop in the path back to the source of the data frame that encountered a forwarding failure.

The source MAC address and PAN identifier shall be set to the address and PAN identifier of the device sending the route error command.

The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. The implementer shall determine whether an acknowledgment shall be requested. The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

2.5.3.2 NWK header fields

In order to send the route error command frame, the destination address field in the NWK header shall be set to the same value as the source address field of the data frame that encountered a forwarding failure.

[111] CCB Comment #132

The source address in the NWK header shall be set to the address of the device sending the route error command.

2.5.3.3 Error code

The error code shall be set to one of the non-reserved values shown in Table 130.

Table 130 Error codes for route error command frame

Value	Error code
0x00	No route available
0x01	Tree link failure
0x02	Non-tree link failure
0x03	Low battery level
0x04	No routing capacity
0x05 -- 0xff	Reserved

2.5.3.4 Destination address

The destination address is 2 octets in length and shall contain the destination address from the data frame that encountered the forwarding failure.

2.5.4 Leave command

The leave command is used by the NLME to inform the parent and children of a device that it is leaving the network or else to request that a device leave the network. The payload of the leave command shall be formatted as shown in Figure 45

Octets: 1	1
Command frame identifier (see Table 129)	Command options

Figure 45 Leave command frame format

2.5.4.1 MAC data service requirement

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be provided.

The destination MAC address and PAN identifier shall be set to the address and PAN identifier, respectively, of the neighbor device to which the frame is being sent.

The source MAC address and PAN identifier shall be set to the address and PAN identifier of the device sending the leave command.

The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. Acknowledgment shall be requested. The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

2.5.4.2 NWK header fields

In order to send the leave command frame, the destination address field in the NWK header shall be set to the network address of the neighbor to which the frame is being sent. The source address in the NWK header shall be set to the address of the device sending the leave command. The radius field in the NWK header shall be set to 1.

2.5.4.3 Command options

The format of the 8-bit command options field is shown in figure 17.

Bit: 0—5	6	7
Reserved	Request/indication	Remove children

Figure 46 Leave command options field

2.5.4.3.1 Request/indication sub-field

The request/indication sub-field is a single bit field located at bit 6. If the value of this sub-field is 1 then the leave command frame is a request for another device to leave the network. If the value of this sub-field is 0 then the leave command frame is an indication that the sending device plans to leave the network.

2.5.4.3.2 Remove children sub-field

The remove children sub-field is a single bit field located at bit 7. If this sub-field has a value of 1 then the children of the device that is leaving the network will also be removed.[112]

2.6 Constants and NIB attributes

2.6.1 NWK constants

The constants that define the characteristics of the NWK layer are presented in Table 131.

Table 131 NWK layer constants

Constant	Description	Value
nwkcCoordinatorCapable	A Boolean flag indicating whether the device is capable of becoming the ZigBee coordinator. A value of 0x00 indicates that the device is not capable of becoming a coordinator while a value of 0x01 indicates that the device is capable of becoming a coordinator.	Set at build time.
nwkcDefaultSecurityLevel	The default security level to be used (see Chapter 3)	ENC-MIC-64
nwkcDiscoveryRetryLimit	The maximum number of times a route discovery will be retried.	0x03
nwkcMaxDepth	The maximum depth (minimum number of logical hops from the ZigBee coordinator) a device can have.	0x0f[a]

[112] CCB Comment #107

Table 131 NWK layer constants

nwkcMinHeaderOverhead	The minimum number of octets added by the NWK layer to a NSDU.	0x08[b]
nwkcProtocolVersion	The version of the ZigBee NWK protocol in the device.	0x01
nwkcRepairThreshold	Maximum number of allowed communication errors after which the route repair mechanism is initiated.	0x03
nwkcRouteDiscoveryTime	Time duration in milliseconds until a route discovery expires.	0x2710
nwkcMaxBroadcastJitter	The maximum broadcast jitter time measured in milliseconds.	0x40
nwkcInitialRREQRetries	The number of times the first broadcast transmission of a route request command frame is retried.	0x03
nwkcRREQRetries	The number of times the broadcast transmission of a route request command frame is retried on relay by an intermediate ZigBee router or ZigBee coordinator.	0x02
nwkcRREQRetryInterval	The number of milliseconds between retries of a broadcast route request command frame.	0xfe
nwkcMinRREQJitter	The minimum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame.	0x01
nwkcMaxRREQJitter	The maximum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame.	0x40

[a]CCB Comment #229
[b]CCB Comment #366

2.6.2 NWK information base

The NWK information base (NIB) comprises the attributes required to manage the NWK layer of a device. Each of these attributes can be read or written using the NLME-GET.request and NLME-SET.request primitives, respectively. The attributes of the NIB are presented in Table 132.

Table 132 NWK IB attributes[a]

Attribute	Id	Type	Range	Description	Default
nwkSequenceNumber	0x81	Integer	0x00-0xff	A sequence number used to identify outgoing frames (see sub-clause 2.7.2)[b]	Random value from within the range.
nwkPassiveAckTimeout	0x82	Integer	0x00-0x0a	The maximum time duration in seconds allowed for the parent and all child devices to retransmit a broadcast message (passive acknowledgment time-out).	0x03
nwkMaxBroadcastRetries	0x83	Integer	0x00-0x5	The maximum number of retries allowed after a broadcast transmission failure.	0x03
nwkMaxChildren	0x84	Integer	0x00 – 0xff	The number of children a device is allowed to have on its current network.	0x07
nwkMaxDepth	0x85	Integer	0x01- *nwkcMaxDepth*	The depth a device can have.	0x05
nwkMaxRouters	0x86	Integer	0x01-0xff	The number of routers any one device is allowed to have as children. This value is determined by the ZigBee coordinator for all devices in the network.	0x05
nwkNeighborTable	0x87	Set	Variable	The current set of neighbor table entries in the device (see Table 133).	Null set
nwkNetworkBroadcastDeliveryTime	0x88	Integer	(*nwkPassiveAckTimeout* * *nwkBroadcastRetries*) – 0xff	Time duration in seconds that a broadcast message needs to encompass the entire network.	*nwkPassiveAckTimeout* * *nwkBroadcastRetries*

Table 132 NWK IB attributes[a]

nwkReportConstantCost	0x89	Integer	0x00-0x01	If this is set to zero, the NWK layer shall calculate link cost from all neighbor nodes using the LQI values reported by the MAC layer. Otherwise it shall report a constant value	0x00
nwkRouteDiscoveryRetriesPermitted	0[c]x8a	Integer	0x00-x03	The number of retries allowed after an unsuccessful route request.	nwkcDiscoveryRetryLimit
nwkRouteTable	0x8b	Set	Variable	The current set of routing table entries in the device (see Table 135).	Null set
nwkSymLink	0x8e	Boolean	TRUE or FALSE	The current route symmetry setting: TRUE means that routes are considers to be comprised of symmetric links. Backward and forward routes are created during one route discovery and they are identical. FALSE indicates that routes are not consider to be comprised of symmetric links. Only the forward route is stored during route discovery	FALSE
nwkCapabilityInformation	0x8f	Bit vector	See Table 112	This field shall contain the capability device capability information established at network joining time.	0x00
nwkUseTreeAddrAlloc	0x90	Boolean	TRUE or FALSE	A flag that determines whether the NWK layer should use the default distributed address allocation scheme or allow the next higher layer to define a block of addresses for the NWK layer to allocate to its children: TRUE = use distributed address allocation. FALSE = allow the next higher layer to define address allocation.	TRUE

Table 132 NWK IB attributes[a]

nwkUseTreeRouting	0x91	Boolean	TRUE or FALSE	A flag that determines whether the NWK layer should assume the ability to use hierarchical routing: TRUE = assume the ability to use hierarchical routing. FALSE = never use hierarchical routing.	TRUE
nwkNextAddress	0x92	Integer	0x0000 - 0xfffd	The next network address that will be assigned to a device requesting association. This value shall be incremented by *nwkAddressIncrement* every time an address is assigned.	0x0000
nwkAvailableAddresses	0x93	Integer	0x0000 - 0xfffd	The size of remaining block of addresses to be assigned. This value will be decremented by 1 every time an address is assigned. When this attribute has a value of 0, no more associations may be accepted.	0x0000
nwkAddressIncrement	0x94	Integer	0x0000 - 0xfffd	The amount by which *nwkNextAddress* is incremented each time an address is assigned.	0x0001
nwkTransactionPersistenceTime	0x95	Integer	0x0000-0xffff	The maximum time (in superframe periods) that a transaction is stored by a coordinator and indicated in its beacon. This attribute reflects the value of the MAC PIB attribute *macTransactionPersistenceTime* (see [B1]) and any changes made by the higher layer will be reflected in the MAC PIB attribute value as well.	0x01f4[d]

[a]CCB Comment #120
[b]CCB Comment #111
[c]CCB Comment #115
[d]CCB Comment #252

2.7 Functional description

2.7.1 Network and device maintenance

All ZigBee devices shall provide the following functionality:

- Join a network.
- Leave a network.

Both ZigBee coordinators and routers shall provide the following additional functionality:

- Permit devices to join the network using the following:
  - Association indications from the MAC.
  - Explicit join requests from the application.
- Permit devices to leave the network using the following:
  - Disassociation indications from the MAC.
  - Explicit leave requests from the application.
- Participate in assignment of logical network addresses.
- Maintain a list of neighboring devices.

ZigBee coordinators shall provide functionality to establish a new network.

2.7.1.1 Establishing a new network

The procedure to establish a new network is initiated through the use of the NLME-NETWORK-FORMATION.request primitive. Only devices that are ZigBee coordinator capable and not currently joined to a network shall attempt to establish a new network. If this procedure is initiated on any other device, the NLME shall terminate the procedure and notify the next higher layer of the illegal request. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to INVALID_REQUEST.

When this procedure is initiated, the NLME shall first request that the MAC sub-layer perform an energy detection scan over either a specified set of channels or, by default, the complete set of available channels, as dictated by the PHY layer [B1], to search for possible interferers. A channel scan is initiated by issuing the MLME-SCAN.request primitive, with the ScanType parameter set to energy detection scan, to the MAC sub-layer and the results are communicated back via the MLME-SCAN.confirm primitive.

On receipt of the results from a successful energy detection scan, the NLME shall order the channels according to increasing energy measurement and discard those channels whose energy levels are beyond an acceptable level. The choice of an acceptable energy level is left to the implementation. The NLME shall then perform an active scan, by issuing the MLME-SCAN.request primitive with a ScanType parameter set to active scan and ChannelList set to the list of acceptable channels to search for other ZigBee devices. To determine the best channel on which to establish a new network, the NLME shall review the list of returned PAN descriptors and find the first channel with the lowest number of existing networks, favoring a channel with no detected networks.

If no suitable channel is found, the NLME shall terminate the procedure and notify the next higher layer of the startup failure. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to STARTUP_FAILURE.

If a suitable channel is found, the NLME shall select a PAN identifier for the new network. To do this, check if the optional *PANId* parameter was specified in the NLME-NETWORK-FORMATION.request was specified. If present and there is no conflict with existing PANIds this value will become the new network's PANId. Otherwise the device shall choose a random PAN identifier such that it is not the broadcast PAN identifier (0xffff) and it is unique amongst the networks found on the selected channel. Additionally, the PAN identifier shall be less than or equal to 0x3fff as the two most significant bits of the 16-bit PAN identifier are reserved for future use. Once the NLME makes its choice, it shall set the *macPANID* attribute in the MAC sub-layer to this value by issuing the MLME-SET.request primitive.

If no unique PAN identifier can be chosen, the NLME shall terminate the procedure and notify the next higher layer of the startup failure by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to STARTUP_FAILURE.

Once a PAN identifier is selected, the NLME shall select a 16-bit network address equal to 0x0000 and set the *macShortAddress* PIB attribute in the MAC sub-layer so that it is equal to the selected network address.

Once a network address is selected, the NLME shall begin operation of the new PAN by issuing the MLME-START.request primitive to the MAC sub-layer. The parameters of the MLME-START.request primitive shall be set according to those passed in the NLME-NETWORK-FORMATION.request, the results of the channel scan, and the chosen PAN identifier. The status of the PAN startup is communicated back via the MLME-START.confirm primitive.

On receipt of the status of the PAN startup, the NLME shall inform the next higher layer of the status of its request to initialize the ZigBee coordinator. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to that returned in the MLME-START.confirm from the MAC sub-layer.

The procedure to successfully start a new network is illustrated in the message sequence chart (MSC) shown in Figure 47.

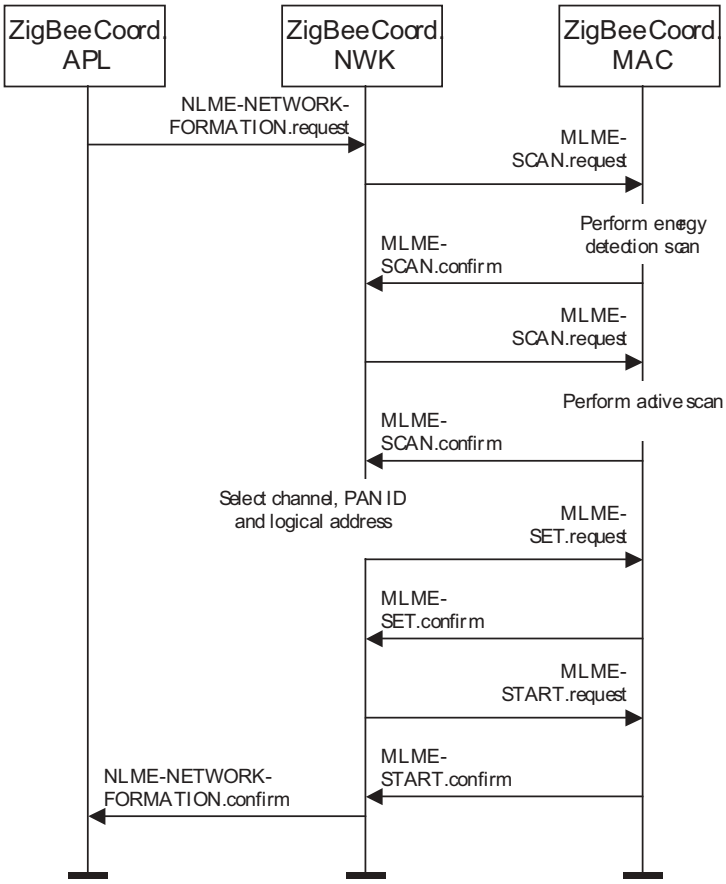


Figure 47 Establishing a new network

2.7.1.2 Permitting devices to join a network

The procedure for permitting devices to join a network is initiated through the NLME-PERMIT-JOINING.request primitive. Only devices that re either the ZigBee coordinator or a ZigBee router shall attempt to permit devices to join the network. If this procedure is initiated on any other device, the NLME shall terminate the procedure.

When this procedure is initiated with the PermitDuration parameter set to 0x00, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE. A MAC sub-layer attribute setting is initiated by issuing the MLME-SET.request primitive.

When this procedure is initiated with the PermitDuration parameter set to a value between 0x01 and 0xfe, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE. The NLME shall then start a timer to expire after the specified duration. On expiry of this timer, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE.

When this procedure is initiated with the PermitDuration parameter set to 0xff, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE for an unlimited amount of time, unless another NLME-PERMIT-JOINING.request primitive is issued.

The procedure for permitting devices to join a network is illustrated in the MSC shown in Figure 48.

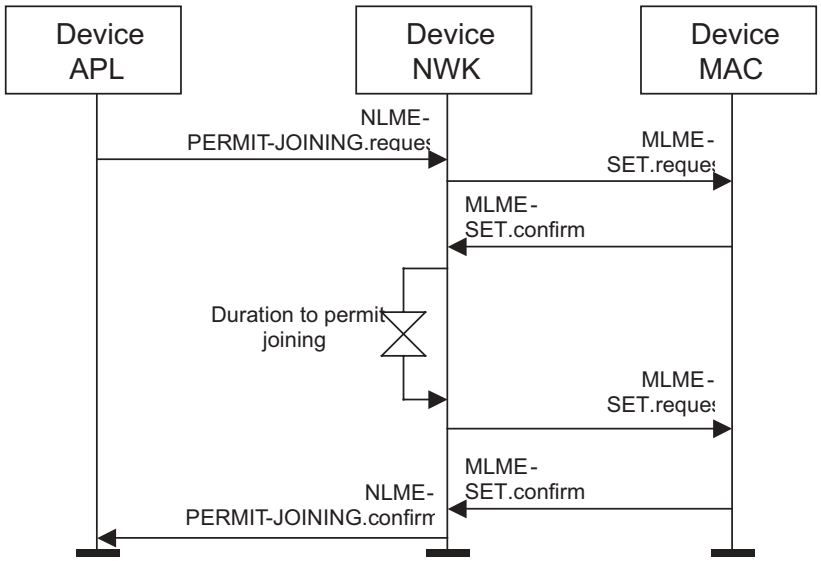


Figure 48 Permitting devices to join a network

2.7.1.3 Joining a network

A parent-child relationship is formed when a device having membership in the network allows a new device to join. The new device becomes the child, while the first device becomes the parent. A child can be added to a network in the following two ways: the child can join the network using the MAC layer association procedure or the child can be added to the network directly by a previously designated parent device.

2.7.1.3.1 Joining a network through association

This sub-clause specifies the procedure a device (child) shall follow to join a network, as well as the procedure a ZigBee coordinator or router (parent) shall follow upon receipt of a join request. Any device may accept a join request from a new device so long as it has the necessary physical capabilities and the available network address space. Only a ZigBee coordinator or a router is physically capable of accepting a join request, while an end device is not.

2.7.1.3.1.1 Child procedure

The procedure for joining a network using the MAC layer association procedure shall be initiated by issuing the NLME-NETWORK-DISCOVERY.request primitive with the ScanChannels parameter set to indicate which channels are to be scanned for networks and the ScanDuration parameter set to indicate the length of time to be spent scanning each channel. Upon receipt of this primitive, the NWK layer shall issue an MLME-SCAN.request primitive asking the MAC sub-layer to perform a passive or[113] active scan.

Every beacon frame received during the scan having a non-zero length payload shall cause the MLME-BEACON-NOTIFY.indication primitive to be issued from the MAC sub-layer of the scanning device to its NLME. This primitive includes information such as the addressing information of the beaconing device, whether it is permitting association and the beacon payload (see [B1] for the complete list of parameters). The NLME of the scanning device shall check the protocol ID field in the beacon payload and verify that it matches the ZigBee protocol identifier. If not, the beacon is ignored. Othewise, the device shall copy the

[113] CCB Comment #117

relevant information from each received beacon (see Figure 63 for the structure of the beacon payload) into its neighbor table (see Table 133 for the contents of a neighbor table entry).

Once the MAC sub-layer signals the completion of the scan by issuing the MLME-SCAN.confirm primitive to the NLME, the NWK layer shall issue the NLME-NETWORK-DISCOVERY.confirm primitive containing a description of each network that was heard. Every network description contains the ZigBee version, stack profile, PAN ID, logical channel[114], and information on whether it is permitting joining (see Table 104).

Upon receipt of the NLME-NETWORK-DISCOVERY.confirm primitive, the next higher layer is informed of the networks present in the neighborhood. The next higher layer may choose to redo the network discovery to discover more networks or for other reasons. If not, it shall choose a network to join from the discovered networks. It shall then issue the NLME-JOIN.request with the PANId parameter set to the PAN identifier of the desired network, the RejoinNetwork parameter set to FALSE and the JoinAsRouter parameter set to indicate whether the device wants to join as a routing device.

Only those devices that are not already joined to a network shall initiate the join procedure. If any other device initiates this procedure, the NLME shall terminate the procedure and notify the next higher layer of the illegal request by issuing the NLME-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST.

For a device that is not already joined to a network, the NLME-JOIN.request primitive shall cause the NWK layer to search its neighbor table for a suitable parent device. A suitable parent device shall have the desired PAN ID and shall be permitting association and shall have a link cost (see sub-clause 2.7.3.1 for details on link cost) of at most 3. It shall also have the potential parent field set to one, if that field is present in the neighbor table entry.

If the neighbor table contains no devices that are suitable parents, the NLME shall respond with an NLME-JOIN-CONFIRM with a status parameter of NOT_PERMITTED. If the neighbor table has more than one device that could be a suitable parent, the device which is at a minimum depth from the ZigBee coordinator shall be chosen. If more than device has a minimum depth, the implementation is free to choose from among them.

Once a suitable parent is identified, the NLME shall issue an MLME-ASSOCIATE.request primitive to the MAC sub-layer. The addressing parameters in the MLME-ASSOCIATE.request primitive (see Chapter 1) shall be set to contain the addressing information for the device chosen from the neighbor table. The status of the association is communicated back to the NLME via the MLME-ASSOCIATE.confirm primitive.

If the attempt to join was unsuccessful, the NWK layer shall receive an MLME-ASSOCIATE.confirm primitive from the MAC sub-layer with the status parameter indicating the error. If the status parameter indicates a refusal to permit joining on the part of the neighboring device (i.e., PAN at capacity or PAN access denied), then the device attempting to join should set the Potential parent bit to zero in the corresponding neighbor table entry to indicate a failed join attempt. Setting the Potential parent bit to zero ensures that the NWK layer shall not issue another request to associate to the same neighboring device. The Potential parent bit should be set to one for every entry in the neighbor table each time an MLME-SCAN.request primitive is issued.

A join request may also be unsuccessful, if the potential parent is not allowing new routers to associate (e.g. the max number of routers, *nwkMaxRouters* may already have associated with the device) and the joining device has set the JoinAsRouter parameter to TRUE. In this case the NLME-JOIN.confirm primitive will indicate a status of NOT_PERMITTED. In this case the child device's application may wish to attempt to join again but as an end device by issuing another NLME-JOIN.request with the JoinAsRouter parameter set to FALSE.

[114]CCB Comment #267

If the attempt to join was unsuccessful, the NLME shall attempt to find another suitable parent from the neighbor table. If no such device could be found, the NLME shall issue the NLME-JOIN.confirm primitive with the Status parameter set to the value returned in the MLME-ASSOCIATE.confirm primitive.

If the attempt to join was unsuccessful and there is a second neighboring device that could be a suitable parent, the NWK layer shall initiate the MAC sub-layer association procedure with the second device. The NWK layer shall repeat this procedure until it either joins the PAN successfully or exhausts its options to join the PAN.

If the device cannot successfully join the PAN specified by the next higher layer, the NLME shall terminate the procedure by issuing the NLME-JOIN.confirm primitive with the Status parameter set to the value returned in the last received MLME-ASSOCIATE.confirm primitive. In this case, the device shall not receive a valid logical address and shall not be permitted to transmit on the network.

If the attempt to join was successful, the MLME-ASSOCIATE.confirm primitive received by the NWK layer shall contain a 16-bit logical address unique to that network that the child can use in future transmissions. The NWK layer shall then set the Relationship field in the corresponding neighbor table entry to indicate that the neighbor is its parent. By this time, the parent shall have each added the new device to its neighbor table.

If the device is attempting to join a secure network and it is a router it will need to wait until its parent has authenticated it before transmitting beacons. The device shall therefore, wait for an NLME-START-ROUTER.request primitive to be issued from the next higher layer. Upon receipt of this primitive the NLME shall issue an MLME-START.request primitive as described below if it is a router. If the NLME-START.ROUTER.request primitive is issued on an end device, the NWK layer shall issue an NLME-START-ROUTER.confirm primitive with the status value set to INVALID_REQUEST.

Once the device has successfully joined the network and the next higher layer has issued a NLME-START-ROUTER.request, if it is a router, the NWK layer shall issue the MLME-START.request primitive to its MAC sub-layer to setup its superframe configuration and begin transmitting beacon frames, if applicable. Beacon frames are only transmitted if the BeaconOrder parameter is not equal to fifteen [B1]. The PANId, LogicalChannel, BeaconOrder and SuperframeOrder parameters shall be set equal to the corresponding values held in the neighbor table entry for its parent. The PANCoordinator and CoordRealignment parameters shall both be set to FALSE. Upon receipt of the MLME-START.confirm primitive, the NWK layer shall issue an NLME-START-ROUTER.confirm primitive with the same status value.

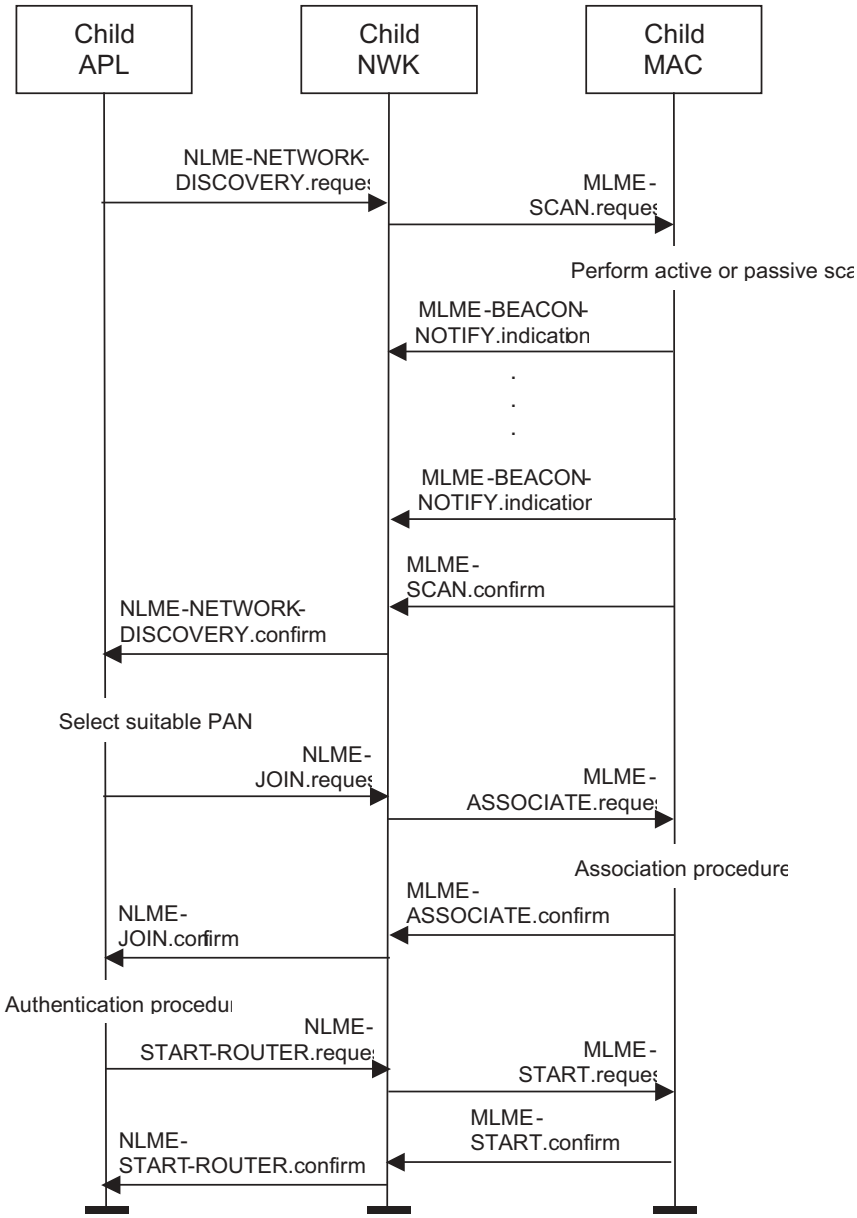


Figure 49 Procedure for joining a network through association

2.7.1.3.1.2 Parent procedure

The procedure for a ZigBee coordinator or router to join a device to its network using the MAC sub-layer association procedure is initiated by the MLME-ASSOCIATE.indication primitive arriving from the MAC sub-layer. Only those devices that are either a ZigBee coordinator or a ZigBee router and that are permitting devices to join the network shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure.

When this procedure is initiated, the NLME of a potential parent shall first determine whether the device wishing to join already exists on its network. To do this, the NLME shall search its neighbor table in order to determine whether a matching 64-bit, extended address can be found. If a match is found, the NLME shall obtain the corresponding 16-bit network address and issue an association response to the MAC sub-layer. If

a match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device that is unique to that network. A finite address space is allocated to every potential parent device, and a device may disallow a join request once this address space is exhausted. The ZigBee coordinator determines the amount of address space given. See sub-clause 2.7.1.4 and sub-clause 2.7.1.5[115] for an explanation of the address assignment mechanism.

If the potential parent has exhausted its allocated address space, the NLME shall terminate the procedure and indicate the fact in the subsequent MLME-ASSOCIATE.response primitive to the MAC sub-layer. The Status parameter of this primitive shall indicate that the PAN is at capacity. This status value uses MAC sub-layer terminology and only indicates that the potential parent does not have the capacity to accept any more children. It is possible in a multihop network that other potential parents still having sufficient address space exist within the same network.

If the request to join is granted, the NLME of the parent shall create a new entry for the child in its neighbor table using the supplied device information and indicate a successful association in the subsequent MLME-ASSOCIATE.response primitive to the MAC sub-layer. The status of the response transmission to the child is communicated back to the network layer via the MLME-COMM-STATUS.indication primitive.

If the transmission was unsuccessful (the MLME-COMM-STATUS.indication primitive contained a Status parameter not equal to SUCCESS), the NLME shall terminate the procedure. If the transmission was successful, the NLME shall notify the next higher layer that a child has just joined the network by issuing the NLME-JOIN.indication primitive.

The procedure for successfully joining a device to the network is illustrated in the MSC shown in Figure 50 – Procedure for handling a join request.

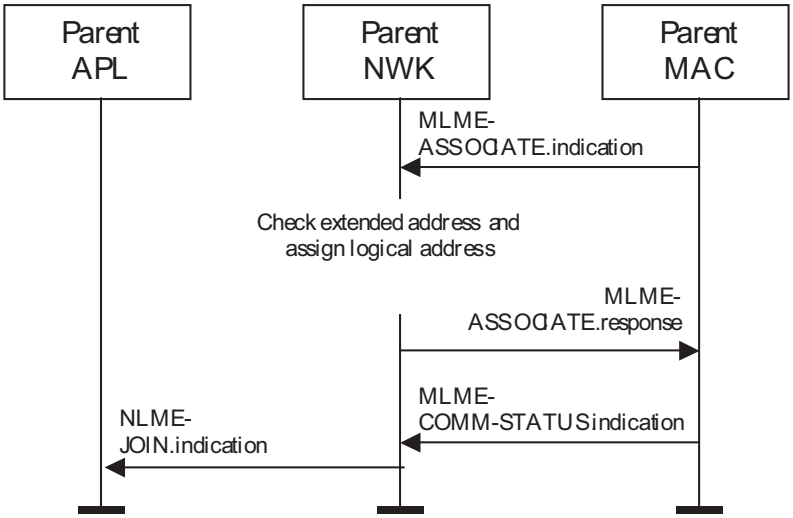


Figure 50 Procedure for handling a join request

2.7.1.3.2 Joining a network directly

This sub-clause specifies how a child can be directly added to a network by a previously designated parent device (ZigBee coordinator or router). In this case, the parent device is preconfigured with the 64-bit address

[115] CCB Comment #122

of the child device. The following text describes how this prior address knowledge shall be used to establish the parent-child relationship.

The procedure for a ZigBee coordinator or router to directly join a device to its network is initiated by issuing the NLME-DIRECT-JOIN.request primitive with the DeviceAddress parameter set to the address of the device to be joined to the network. Only those devices that are either a ZigBee coordinator or a ZigBee router shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure and notify the next higher layer of the illegal request by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST.

When this procedure is initiated, the NLME of the parent shall first determine whether the specified device already exists on its network. To do this, the NLME shall search its neighbor table in order to determine whether a matching 64-bit, extended address can be found. If a match is found, the NLME shall terminate the procedure and notify the next higher layer that the device is already present in the device list by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to ALREADY_PRESENT.

If a match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device, which is unique to that network. A finite address space is allocated to every potential parent device, and the potential parent shall only create a new entry for the device in its neighbor table if it has the capacity to do so. If capacity is not available, the NLME shall terminate the procedure and notify the next higher layer of the unavailable capacity by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to TABLE_FULL. If capacity is available, the NLME shall inform the next higher layer that the device has joined the network by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to SUCCESS.

The ZigBee coordinator determines the amount of address space given to every potential parent device. See sub-clause 2.7.1.4 and sub-clause 2.7.1.5[116] for an explanation of the address assignment mechanism.

Once the parent has added the child to its network, it is still necessary for the child to make contact with the parent to complete the establishment of the parent-child relationship. The child shall fulfill this requirement by initiating the orphaning procedure, which is described in sub-clause 2.7.1.3.

The procedure a parent shall follow to successfully join a device to the network directly is illustrated in the MSC shown in Figure 51. This procedure does not require any over-the-air transmissions.

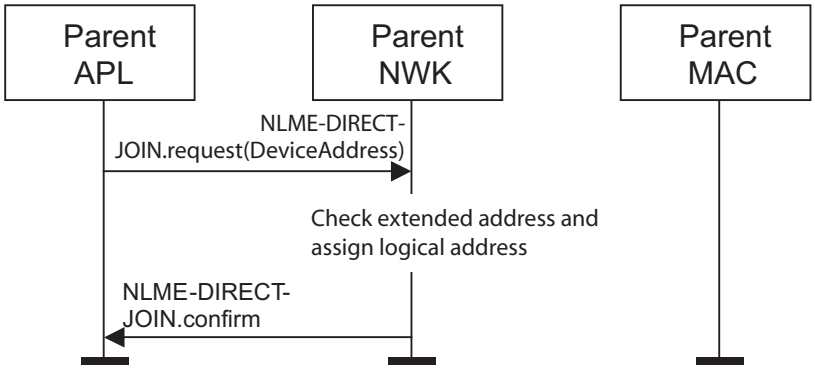


Figure 51 Joining a device to a network directly

[116]CCB Comment #122

2.7.1.3.3 Joining or re-joining a network through orphaning

This sub-clause specifies how the orphaning procedure can be initiated by a device that has been directly joined to a network (joining through orphaning) or by a device that was previously joined to a network but has lost contact with its parent (re-joining through orphaning).

A device that has been added to a network directly shall initiate the orphan procedure in order to complete the establishment of its relationship with its parent. The application on the device will determine whether to initiate this procedure and, if so, will notify the network layer upon power up.

A device that was previously joined to a network has the option of initiating the orphan procedure if its NLME repeatedly receives communications failure notifications from its MAC sub-layer.

2.7.1.3.3.1 Child procedure

The joining through orphaning procedure is initiated by a child device through the NLME-JOIN.request primitive with RejoinNetwork parameter set to TRUE.

When this procedure is initiated, the NLME shall first request that the MAC sub-layer perform an orphan scan over the complete set of available channels, as dictated by the PHY layer [B1]. An orphan scan is initiated by issuing the MLME-SCAN.request primitive to the MAC sub-layer, and the result is communicated back to the NLME via the MLME-SCAN.confirm primitive.

If the orphan scan was successful (the child has found its parent), the NLME shall inform the next higher layer of the success of its request to join or re-join the network by issuing the NLME-JOIN.confirm primitive with the Status parameter set to SUCCESS.

If the orphan scan was unsuccessful (the parent has not been found), the NLME shall terminate the procedure and notify the next higher layer that no networks were found. This is achieved by issuing the NLME-JOIN.confirm primitive with the Status parameter set to NO_NETWORKS.

The procedure for a child to successfully join or re-join a network through orphaning is illustrated in the MSC shown in Figure 52.

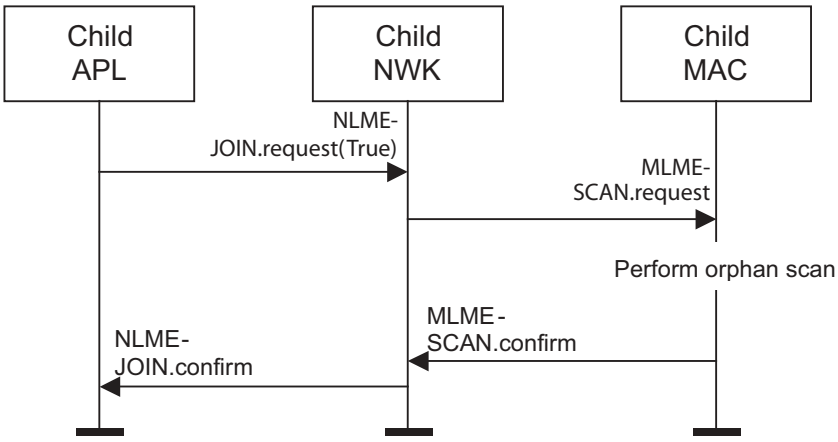


Figure 52 Child procedure for joining or re-joining a network through orphaning

2.7.1.3.3.2 Parent procedure

A device is notified of the presence of an orphaned device when it receives the MLME-ORPHAN.indication primitive from the MAC sub-layer. Only those devices that are either a ZigBee coordinator or a ZigBee

router (a device with parental capabilities) shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure.

When this procedure is initiated, the NLME shall first determine whether the orphaned device is its child. This is accomplished by comparing the extended address of the orphaned device with the addresses of its children, as recorded in its neighbor table. If a match is found (the orphaned device is its child), the NLME shall obtain the corresponding 16-bit network address and include it in its subsequent orphan response to the MAC sub-layer. The orphan response to the MAC sub-layer is initiated by issuing the MLME-ORPHAN.response primitive, and the status of the transmission is communicated back to the NLME via the MLME-COMM-STATUS.indication primitive.

If an address match is not found (the orphaned device is not its child), the NLME shall indicate the fact in its subsequent orphan response to the MAC sub-layer.

The procedure for a parent to join or re-join its orphaned child to the network is illustrated in the MSC shown in Figure 53.

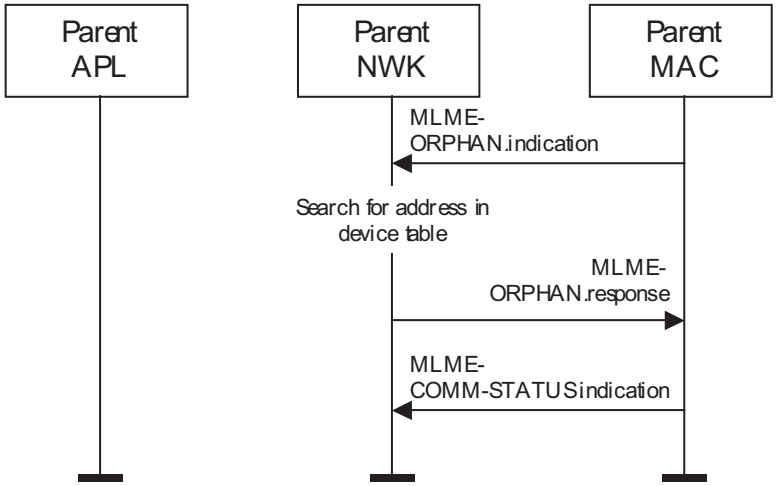


Figure 53 Parent procedure for joining or re-joining a device to its network through orphaning

2.7.1.3.4 Neighbor tables

The neighbor table of a device shall contain information on every device within transmission range up to some implementation-dependent limit. The information in stored in the neighbor table is used for a variety of purposes, however, not all fields described in this subsection are required for the operation of a ZigBee device. Each entry in the table shall contain the following information about a neighboring device:

- PAN identifier
- Extended address if device is parent or child
- Network address
- Device type
- Relationship

The following are not required but implementers may wish to include the following information in each neighbor table. Additionally, it should be noted that implementers may wish to record additional information in the neighbor table and the following list is not intended to exclude this possibility.

- RxOnWhenIdle
- Extended address (any neighbor)
- Beacon Order
- Depth
- Permit joining
- Transmit Failure
- Potential parent
- Average LQI
- Logical Channel
- Incoming beacon frame timestamp
- Beacon transmission time offset

A table entry shall be updated every time a device receives any frame from the corresponding neighbor. The contents of a neighbor table entry are shown in Table 133.

Table 133 Neighbor table entry format

Field name	Field type	Valid range	Description
PAN Id	Integer	0x0000—0x3fff	The 16-bit PAN identifier of the neighboring device. This field shall be present in every neighbor table entry.
Extended address	Integer	An extended 64-bit, IEEE address	64-bit IEEE address that is unique to every device. This field shall be present if the neighbor is a parent or child of the device
Network address	Network address	0x0000—0xffff	The 16-bit network address of the neighboring device. This field shall be present in every neighbor table entry.
Device type	Integer	0x00—0x02	The type of the neighbor device.: 0x00 = ZigBee coordinator 0x01 = ZigBee router 0x02 = ZigBee end device This field shall be present in every neighbor table entry.

Table 133 Neighbor table entry format

RxOnWhenIdle	Boolean	TRUE or FALSE	Indicates if neighbor's receiver is enabled during idle portions of the CAP[a]: TRUE = Receiver is off FALSE = Receiver is on This field should be present for entries that record the parent or children of a ZigBee router or ZigBee coordinator.
Relationship	Integer	0x00—0x03	The relationship between the neighbor and the current device: 0x00=neighbor is the parent 0x01=neighbor is a child 0x02=neighbor is a sibling 0x03=None of the above. This field shall be present in every neighbor table entry.
Depth	Integer	0x00—*nwkcMaxDepth*	The tree depth of the neighbor device. A value of 0x00 indicates that the device is the ZigBee coordinator for the network. This field is optional.
Beacon order	Integer	0x00—0x0f	This specifies how often the beacon is to be transmitted. For a definition and discussion of beacon order see [B1]. This field is optional.
Permit joining	Boolean	TRUE or FALSE	An indication of whether the neighbor device is accepting join requests. TRUE = neighbor is accepting join requests FALSE = neighbor is not accepting join requests This field is optional.
Transmit Failure	Integer	0x00—0xff	A value indicating if previous transmissions to the device were successful or not. Higher values indicate more failures. This field is optional.

Table 133 Neighbor table entry format

Potential parent	Integer	0x00—0x01	An indication of whether the neighbor has been ruled out as a potential parent due to a failed join attempt: 0x00 indicates that the neighbor is not a potential parent 0x01 indicates that the neighbor is a potential parent. This field is optional.
LQI	Integer	0x00—0xff	The estimated link quality for RF transmissions from this device. See sub-clause 2.7.3.1 for discussion of how this is calculated. This field is optional.
Logical channel	Integer	Selected from the available logical channels supported by the PHY	The logical channel on which the neighbor is operating. This field is optional.
Incoming beacon timestamp	Integer	0x000000-0xffffff	The time, in symbols, at which the last beacon frame was received from the neighbor. This value is equal to the timestamp taken when the beacon frame was received, as described in [B1].
Beacon transmission time offset	Integer	0x000000-0xffffff	The transmission time difference, in symbols, between the neighbor's beacon and its parent's beacon. This difference may be subtracted from the corresponding incoming beacon timestamp to calculate the beacon transmission time of the neighbor's parent.

[a]CCB Comment #138

2.7.1.4 Distributed[117] address assignment mechanism

By default, i.e. when the NIB attribute *nwkUseTreeAddrAlloc* has a value of TRUE[118], network addresses are assigned using a distributed addressing scheme that is designed to provide every potential parent with a finite sub-block of network addresses. These addresses are unique within a particular network and are given by a parent to its children. The ZigBee coordinator determines the maximum number of children that any device within its network is allowed. Of these children, a maximum of *nwkMaxRouters* can be router-capable devices while the rest shall be reserved for end devices. Every device has an associated depth, which indicates the minimum number of hops a transmitted frame must travel, using only parent-child links, to

[117]CCB Comment #122
[118]Ibid

reach the ZigBee coordinator. The ZigBee coordinator itself has a depth of zero, while its children have a depth of one. Multihop networks have a maximum depth that is greater than one. The ZigBee coordinator also determines the maximum depth of the network.

Given values for the maximum number of children a parent may have, *nwkMaxChildren* (*Cm*), the maximum depth in the network, *nwkMaxDepth* (*Lm*), and the maximum number of routers a parent may have as children, *nwkMaxRouters* (*Rm*), we may compute the function, $Cskip(d)$, essentially the size of the address sub-block being distributed by each parent at that depth to its router-capable child devices for a given network depth, d , as follows:[119]

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1), \text{if Rm} = 1 \\ \dfrac{1 + Cm - Rm - Cm \cdot Rm^{Lm-d-1}}{1 - Rm}, \text{otherwise} \end{cases}$$

If a device has a $Cskip(d)$ value of zero, then it shall not be capable of accepting children and shall be treated as a ZigBee end device for purposes of this discussion. The NLME of the device shall set the macAssociationPermit PIB attribute in the MAC sub-layer to FALSE by issuing the MLME-SET.request primitive and shall respond to future NLME-PERMIT-JOINING.request primitive with a PermitDuration of equal or greater than 0x01 with a NLME-PERMIT-JOINING.confirm primitive with a Status parameter of INVALID_REQUEST and shall terminate the permit joining procedure.

A parent device that has a $Cskip(d)$ value greater than zero shall accept child devices and shall assign addresses to them differently depending on whether the child device is router-capable or not.

Network addresses shall be assigned to router-capable child devices using the value of $Cskip(d)$ as an offset. A parent assigns an address that is one greater than its own to its first router-capable child device. Subsequently assigned addresses to router-capable child devices are separated from each other by $Cskip(d)$. A maximum of *nwkMaxRouters* of such addresses shall be assigned.

Network addresses shall be assigned to end devices in a sequential manner with the nth address, A_n , given by the following equation:

$$A_n = A_{parent} + Cskip(d) \cdot Rm + n$$

Where $1 \le n \le (Cm - Rm)$ and A_{parent} represents the address of the parent.

The $Cskip(d)$ values for an example network having *nwkMaxChildren*=4, *nwkMaxRouters*=4 and *nwkMaxDepth*=3 are calculated and listed in Table 134. Figure 54 illustrates the example network.

[119]CCB Comment #100 resulted in a change to this formula.The original was:

$$Cskip(d) = \frac{1 + Cm - Rm - Cm \cdot Rm^{Lm-d-1}}{1 - Rm}$$

Table 134 Example addressing offset values for each given depth within the network

Depth in the network, *d*	Offset value, *Cskip(d)*
0	21
1	5
2	1
3	0

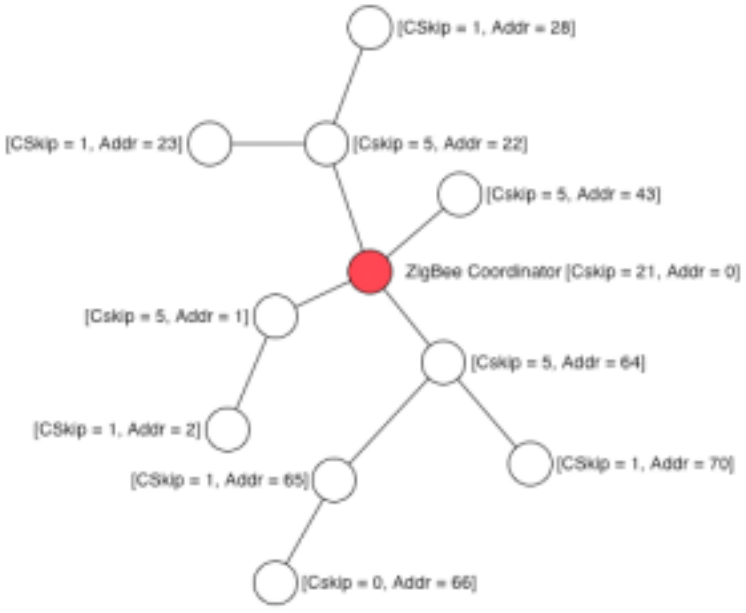


Figure 54 Address assignment in an example network

Because an address sub-block cannot be shared between devices, it is possible that one parent exhausts its list of addresses while a second parent has addresses that go unused. A parent having no available addresses shall not permit a new device to join the network. In this situation, the new device shall find another parent. If no other parent is available within transmission range of the new device, the device shall be unable to join the network unless it is physically moved or there is some other change.

2.7.1.5 Higher-layer address assignment mechanism

When the NIB attribute *nwkUseTreeAddrAlloc* has a value of FALSE, an alternate addressing scheme is used where the block of addresses to be assigned by a device is set by the next higher layer using the NIB attributes *nwkNextAddress*, *nwkAvailableAddresses* and *nwkAddressIncrement*. In this scheme, when a device has *nwkAvailableAddresses* equal to 0 it shall be incapable of accepting association requests. The NLME of such a device shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE by issuing the MLME-SET.request primitive and shall respond to future NLME-PERMIT-JOINING.request primitives with a PermitDuration of equal to or greater than 0x01 with a NLME-PERMIT-JOINING.confirm primitive with a Status parameter of INVALID_REQUEST and shall terminate the

permit joining procedure. If the device has *nwkAvailableAddresses* greater than 0, it shall accept association requests by setting the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE and by issuing the MLME-SET.request primitive and it shall respond to future NLME-PERMIT-JOINING.request primitives with a PermitDuration of equal to or greater than 0x01 with a Status parameter of SUCCESS. While a device is accepting associations it shall use the value of *nwkNextAddress* as the address to be assigned to the next device that successfully associates. After a successful association, the value of *nwkNextAddress* shall be incremented by the value of *nwkAddressIncrement* and the value of *nwkAvailableAddresses* shall be decremented by 1.

2.7.1.6 Installation and addressing

It should be clear that *nwkMaxDepth* roughly determines the “distance” in network terms from the root of the tree to the farthest end device. In principal *nwkMaxDepth* also determines the overall network diameter. In particular, for an ideal network layout where the ZigBee coordinator is located in the center of the network, as in Figure 54, the network diameter should be 2 * *nwkMaxDepth*. In practice, application-driven placement decisions and order of deployment may lead to a smaller diameter. In this case, *nwkMaxDepth* provides a lower bound on the network diameter while the 2* *nwkMaxDepth* provides the upper bound.

Finally, due to the fact that the tree is not dynamically balanced in ZigBee 1.0, the possibility exists that certain installation scenarios, such as long lines of devices, may exhaust the address capacity of the network long before the real capacity is reached.

2.7.1.7 Leaving a network

This sub-clause specifies two methods for removing a child from a network that both use the MAC layer disassociation procedure. The first is initiated by a child as a request to its parent, while the second is initiated by a parent as a request to its child.

2.7.1.7.1 Method for a child to initiate its own removal from a network

This sub-clause describes how a device can initiate its own removal from the network in response to the receipt of an NLME-LEAVE.request primitive from the next higher layer or in response to the receipt of a leave command frame from its parent with the request/indication sub-field of the command options field of the command frame payload set to 1.

When this procedure is initiated, the NLME shall transmit a leave request command frame to each of its children, if any. If the procedure was initiated from the next higher layer and the RemoveChildren parameter of the NLME-LEAVE.request that initiated the procedure is equal to FALSE then the remove children sub-field of the command options field in the command frame payload of each outgoing frame shall be set to 0. If the RemoveChildren parameter has a value of TRUE the remove children sub-field shall be set to 1. If the procedure was initiated by the receipt of a leave command frame then the remove children sub-field of each outgoing command frame's payload should match that of the received frame. If removal of children is called for, and the device has children, the NLME shall attempt to remove each of the device's children in turn using the procedure described in sub-clause 2.7.1.7.2. The NLME shall then transmit a leave command frame to the device's parent, using the MCPS-DATA.request primitive of the MAC sub-layer, with the request/indication sub-field of the command options field of the command frame payload set to 0. If removal of children was not called for then the remove children sub-field of the command options field in the command frame payload shall be set to 0. If removal of children was called for the remove children sub-field of the command options field in the command frame payload shall be set to 1 if the device has no children or else if the device has children and all of the device's children were successfully removed and to 0 otherwise. The NLME may then issue the MLME-DISASSOCIATE.request primitive to the MAC sub-layer with the DeviceAddress parameter equal to the address of the device's parent and the DisassociateReason parameter equal to 0x02. On receipt of the MLME-DISASSOCIATE.confirm primitive, the NLME shall issue the NLME-LEAVE.confirm primitive to the next higher layer with the DeviceAddress parameter equal to 0. The Status parameter of the NLME-LEAVE.confirm primitive shall have a value of SUCCESS if:

1) The status returned by the initial MCPS-DATA.confirm above has a value of SUCCESS, and
2) If the NLME attempts to remove the children of the device in turn, then each of the children is successfully removed, and
3) The status returned by the MLME.DISASSOCIATE.confirm primitive, if any, is also SUCCESS.

Otherwise it shall have a value of LEAVE_UNCONFIRMED.

When the NLME of any device receives one of the leave command frames issued by the leaving device as described above, it must check its relationship to the sender. If the device receiving the leave command frame is the parent of the leaving device then it shall check the value of the remove children sub-field of the command options field in the command frame payload. If this sub-field has a value of 1 then the parent may be able to reuse the 16-bit network address previously in use by the leaving device. If the remove children sub-field has a value of 0 then the parent shall not reuse the 16-bit network address of the leaving device. In either case, it shall set the relationship field of its neighbor table entry corresponding to the leaving device to 0x03 indicating no relationship. If the device receiving a leave command frame is a child of the leaving device then it shall check the value of the request/indication sub-field of the command options field in the command frame header. If this sub-field has a value of 1, the NLME shall execute the procedure outlined in this sub-clause. If the request/indication sub-field has a value of 0, the NLME shall issue a NLME.LEAVE.indication primitive to the next higher layer with the DeviceAddress set to the 64-bit address of the sender of the leave command frame.[120]

2.7.1.7.2 Method for a parent to force a child to leave its network

This sub-clause specifies how a parent can force a child to leave its network employing the leave command frame and MAC sub-layer disassociation.

The procedure for a parent to remove a child from its network is initiated by issuing the NLME-LEAVE.request primitive with the DeviceAddress parameter set to the address of the device to be removed from the network. Only those devices that are either a ZigBee coordinator or a ZigBee router shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure and notify the next higher layer of the illegal request by issuing the NLME-LEAVE.confirm primitive with the Status parameter set to INVALID_REQUEST.

When this procedure is initiated by the next higher layer the NLME shall first determine whether the specified device already exists on its network. To do this, the NLME shall search its neighbor table in order to determine whether a matching extended address can be found. If a match is not found, the NLME shall terminate the procedure and inform the next higher layer of the unknown device by issuing the NLME-LEAVE.confirm primitive with the Status parameter set to UNKNOWN_DEVICE. The NLME shall then transmit a leave command frame to the child device, using the MCPS-DATA.request primitive of the MAC sub-layer, with the request/indication sub-field of the command options field of the command frame payload set to 1. If the recursive removal of children is called for, then the remove children sub-field of the outgoing leave command payload will have a value of 1. Otherwise it will have a value of 0. After issuing the leave command frame the NLME shall wait for a time-out period that is equal to:

$$\begin{cases} nwkTransactionPersistenceTime, & \text{if removal of children is not called for} \\ nwkTransactionPersistenceTime \bullet Cskip(d), & \text{if it is} \end{cases}$$

to receive a leave command frame from the MAC, via the MCPS-DATA.indication, where the source address of the frame is that of the child being asked to leave the network the request/indication subfield of

[120] CCB Comment #107

the command options field of the command frame payload has a value of 0. It may also wait for the receipt of an MLME-DISSSOCIATE.indication from the leaving device. At this point it shall issue the NLME-LEAVE.confirm primitive with the DeviceAddress parameter set to the 64-bit IEEE address of the leaving device. The Status parameter shall have a value of SUCCESS if:

1) The status value returned by the MLME-DATA.confirm resulting from the transmission of the leave command frame was SUCCESS, and
2) The leave command frame issued by the device's was received before the time-out, and
3) The recursive removal of children was not called for, or else recursive removal of children was called for and the remove children subfield of the command options field of the command frame payload of the received leave command frame above had a value of 1.

Otherwise it shall have a value of LEAVE_UNCONFIRMED.

Child devices receiving the leave command frame will execute the procedure described in sub-clause 2.7.1.7.1.[121]

[121]CCB Comment #107

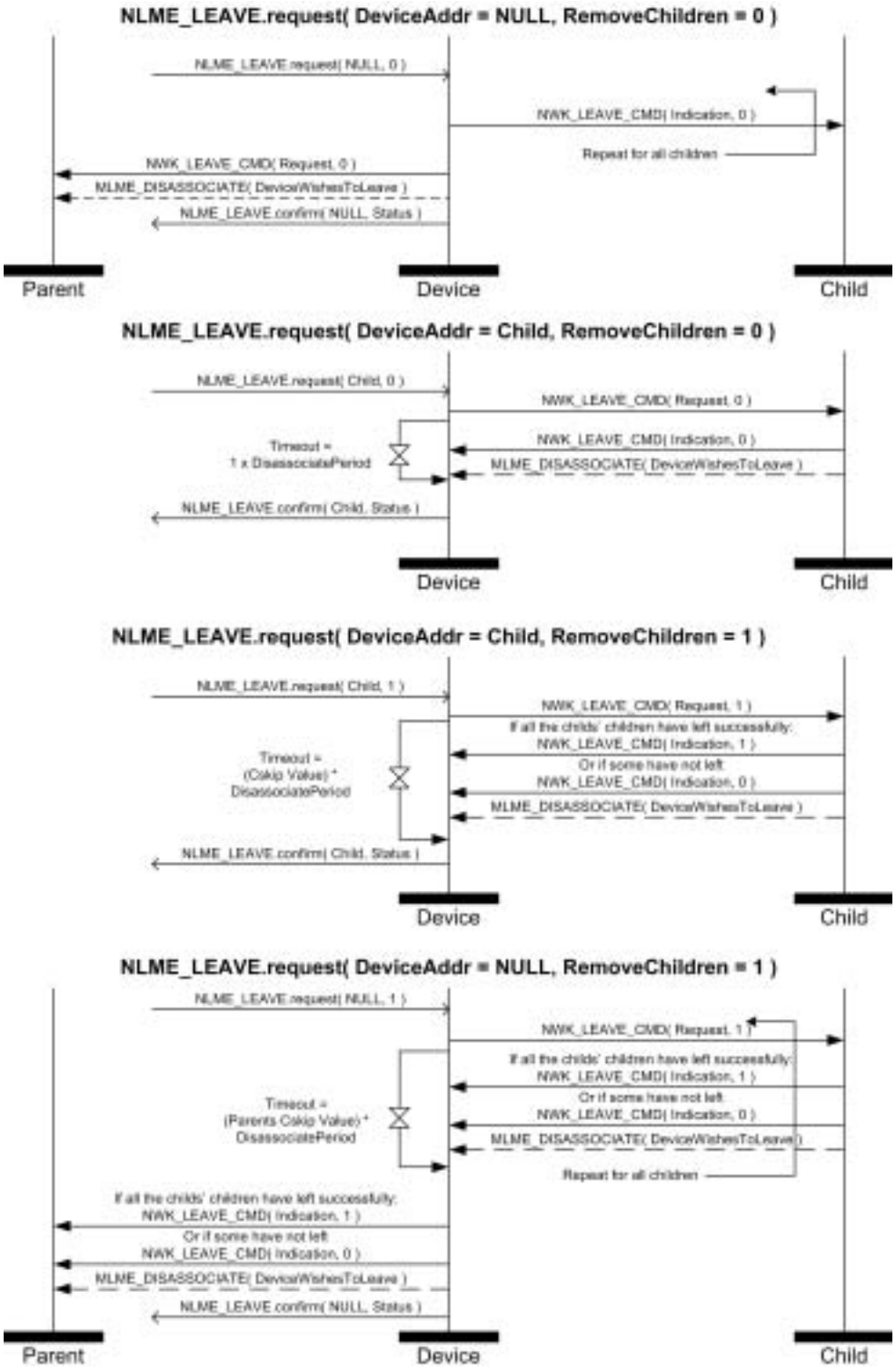


Figure 55 Sequence diagrams for NLME-LEAVE.request, various scenarios[122]

[122] CCB Comment #107

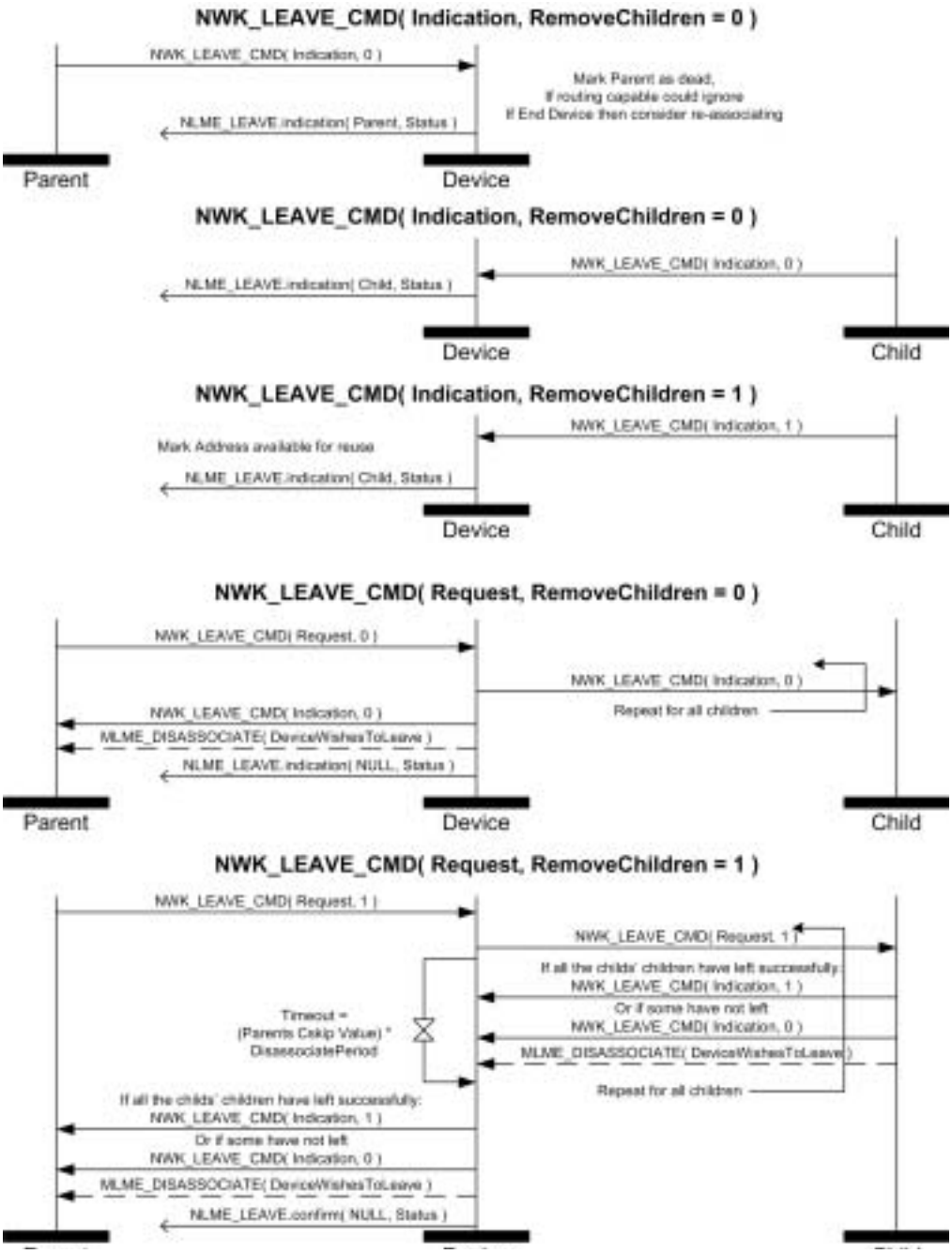


Figure 56 Leave command, various scenarios[123]

2.7.1.8 Changing the ZigBee coordinator configuration

If the next higher layer of a ZigBee coordinator device wishes to change the configuration of the network, it shall request that the MAC sub-layer instigate the changes in its PIB. The ZigBee coordinator configuration is composed of the following items:

— Whether the device wishes to be the ZigBee coordinator.

[123]CCB Comment #107

— The beacon order of the MAC super-frame.

— The super-frame order of the MAC super-frame.

— Whether battery life extension mode is to be used.

A change to the ZigBee coordinator configuration is initiated by issuing the NLME-NETWORK-FORMATION.request primitive to the NLME. The status of the attempt is communicated back via the NLME-NETWORK-FORMATION.confirm primitive.[124]

The impact of such changes imposed on the MAC sub-layer is out of the scope of this specification. For more details on these changes see [B1].

2.7.1.9 Resetting a device

The NWK layer of a device shall be reset immediately following power-up, before a join attempt by association and after a leave attempt by disassociation. A reset is initiated by issuing the NLME-RESET.request primitive to the NLME and the status of the attempt is communicated back via the NLME-RESET.confirm primitive. The reset process shall clear the routing table entries of the device. Some devices may store NWK layer quantities in non-volatile memory and restore them after a reset. However, a device shall discard its network address after the reset. Such a device shall look for an association and get a network address from its coordinator. The new network address may be different from its old network address. In such a case, any device that is communicating with the device that has been reset must rediscover the device using higher-layer protocols and procedures that are out of scope for this specification.

2.7.2 Transmission and reception

2.7.2.1 Transmission

Only those devices that are currently associated shall send data frames from the NWK layer. If a device that is not associated receives a request to transmit a frame, it shall discard the frame and notify the higher layer of the error by issuing an NLDE-DATA.confirm primitive with a status of INVALID_REQUEST.

All frames handled by or generated within the NWK layer shall be constructed according to the general frame format specified in Figure 36 and transmitted using the MAC sub-layer data service.

In addition to source address and destination address fields, all NWK layer transmissions shall include a radius field and a sequence number field. For data frames originating at a higher layer, the value of the radius field may be supplied using the Radius parameter of the NLDE-DATA.request primitive. If a value is not supplied, then the radius field of the NWK header shall be set to twice the value of the *nwkMaxDepth* attribute of the NWK IB (see clause 2.6). The NWK layer on every device shall maintain a sequence number that is initialized with a random value. The sequence number shall be incremented by one, each time the NWK layer constructs a new NWK frame, either as a result of a request from the next higher layer to transmit a new NWK data frame or when it needs to construct a new NWK layer command frame. After being incremented the value of the sequence number shall be inserted into the sequence number field of the frame's NWK header[125]. Once an NPDU is complete, if security is required for the frame, it shall be passed to the security service provider for subsequent processing according to the specified security suite (see sub-clause 3.2.3). Security processing is not required if the SecurityEnable parameter of the NDLE-DATA.request is EQUAL to FALSE or if the NWK security level as specified in *nwkSecurityLevel* is equal to 0. In this case the security sub-field of the frame control field shall always be set to 0. On successful completion of the secure processing, the security suite returns the frame to the NWK layer for transmission. The processed frame will have the correct auxiliary header attached. If security processing of the frame fails

[124] CCB Comment #137

[125] CCB Comment #111, 125

and the frame was a data frame then the higher layer will be informed via the NDLE-DATA.confirm primitive's status. If security processing of the frame fails and the frame is a network command frame then it is discarded and no further processing shall take place

When the frame is constructed and ready for transmission, it shall be passed to the MAC data service. An NPDU transmission is initiated by issuing the MCPS-DATA.request primitive to the MAC sub-layer and the results of the transmission returned via the MCPS-DATA.confirm primitive.

2.7.2.2 Reception and rejection

In order to receive data, a device must enable its receiver. The next higher layer may initiate reception using the NLME-SYNC.request primitive. On a beacon-enabled network, receipt of this primitive by the NWK layer will cause a device to synchronize with its parent's next beacon and, optionally, to track future beacons. The NWK layer shall accomplish this by issuing an MLME-SYNC.request to the MAC sub-layer. On a non-beacon-enabled network the NLME-SYNC.request will cause the NWK layer to poll the device's parent using the MLME-POLL.request primitive.

On a non-beacon-enabled network, the NWK layer on a ZigBee coordinator or ZigBee router shall ensure, to the maximum extent feasible, that the receiver is enabled whenever the device is not transmitting. On a beacon-enabled network, the NWK layer should ensure that the receiver is enabled when the device is not transmitting during the active period of its own superframe and of its parent's superframe. The NWK layer may use the *macRxOnWhenIdle* attribute of the MAC PIB for this purpose.

Once the receiver is enabled, the NWK layer will begin to receive frames via the MAC data service. On receipt of each frame, the Radius field of the NWK header shall be decremented by 1. If, as a result of being decremented, this value falls to 0 then the frame shall not, under any circumstances, be retransmitted, although it may be passed to the next higher layer or otherwise processed by the NWK layer as outlined elsewhere in this specification.[126] Data frames for which the destination address matches the device's network address shall be passed to the next higher layer. Broadcast data frames shall also be passed to the next higher layer. Broadcast data frames shall also be relayed according to the procedure outlined in sub-clause 2.7.5. If the receiving device is a ZigBee coordinator or an operating ZigBee router, i.e. a router that has already invoked the NLME-START-ROUTER.request primitive, then it may relay data frames for which the destination address does not match the device's network address according to the procedures outlined in sub-clause 2.7.3.3. Under all other circumstances, data frames shall be discarded immediately. The procedure for handling route request frames is outlined in sub-clause 2.7.3.4.2. The procedure for handling route reply command frames for which the destination address matches the device's network address is outlined in sub-clause 2.7.3.4.3. Route reply command frames for which the destination address does not match the device's network address shall be discarded immediately. Route error command frames shall be handled in the same manner as data frames.[127]

The NWK layer shall indicate the receipt of a data frame to the next higher layer using the NLDE-DATA.indication primitive.

On receipt of a frame, the NLDE shall check the value of the security sub-field of the frame control field. If this value is non-zero, the NLDE shall pass the frame to the security service provider (see sub-clause 3.2.3) for subsequent processing according to the specified security suite.

2.7.3 Routing

ZigBee coordinators and routers shall provide the following functionality:

— Relay DATA frames on behalf of higher layers.

[126] CCB Comment #125
[127] CCB Comment #134

- Relay DATA frames on behalf of other ZigBee routers.
- Participate in route discovery in order to establish routes for subsequent DATA frames.
- Participate in route discovery on behalf of end devices.
- Participate in end-to-end route repair.
- Participate in local route repair.
- Employ the ZigBee path cost metric as specified in route discovery and route repair.

ZigBee coordinators or routers may provide the following functionality:

- Maintain routing tables in order to remember best available routes.
- Initiate route discovery on behalf of higher layers.
- Initiate route discovery on behalf of other ZigBee routers.
- Initiate end-to-end route repair.
- Initiate local route repair on behalf of other ZigBee routers.

2.7.3.1 Routing cost

The ZigBee routing algorithm uses a path cost metric for route comparison during route discovery and maintenance. In order to compute this metric, a cost, known as the link cost, is associated with each link in the path and link cost values are summed to produce the cost for the path as a whole.

More formally, if we define a path P of length L as an ordered set of devices $[D_1, D_2 ... D_L]$ and a link, $[D_i, D_{i+1}]$, as a sub-path of length 2, then the path cost

$$C\{P\} = \sum_{i=1}^{L-1} C\{[D_i, D_{i+1}]\}$$

where each of the values $C\{[D_i, D_{i+1}]\}$ is referred to as a link cost. The link cost $C\{l\}$ for a link l is a function with values in the interval $[0...7]$ defined as:[128]

$$C\{l\} = \begin{cases} 7, \\ \min\left(7, \text{round}\left(\frac{1}{p_l^4}\right)\right) \end{cases}$$

where p_l is defined as the probability of packet delivery on the link l.

[128] CCB Comment #133 mandates a change in value for this formula. The previous value was:

$$C\{l\} = \begin{cases} 7, \\ \min\left(7, \left\lceil \frac{1}{p_l} \right\rceil\right) \end{cases}$$

Thus, implementers may report a constant value of 7 for link cost or they may report a value that reflects the probability p_l of reception - specifically, the reciprocal of that probability – which should, in turn, reflect[129] the number of expected attempts required to get a packet through on that link each time it is used. A device that offers both options may be forced to report a constant link cost by setting the value of the NIB attribute *nwkReportConstantCost* to TRUE.

The question that remains, however, is how p_l is to be estimated or measured. This is primarily an implementation issue and implementers are free to apply their ingenuity. p_l may be estimated over time by actually counting received beacon and data frames, observing the appropriate sequence numbers to detect lost frames, and this is generally regarded as the most accurate measure of reception probability. However, the most straightforward method, available to all, is to form estimates based on an average over the per-frame LQI value provided by the IEEE 802.15.4-2003 MAC and PHY. Even if some other method is used, the initial cost estimates shall be based on average LQI. A table-driven function may be used to map average LQI values onto C{*l*} values. It is strongly recommended that implementers check their tables against data derived from tests on production hardware, as inaccurate costs will hamper the ability of the ZigBee routing algorithm to operate[130]

2.7.3.2 Routing tables

A ZigBee router or ZigBee coordinator may maintain a routing table. The information that shall be stored in a ZigBee routing table is shown in Table 135. A ZigBee router or ZigBee coordinator may also reserve a number of routing table entries to be used only for route repair and only in case all other routing capacity has been exhausted. The aging and retirement of routing table entries in order to reclaim table space from entries that are no longer in use, while it is a recommended practice, is out of scope of this specification.[131].

Table 135 Routing table

Field Name	Size	Description
Destination address	2 bytes	The 16-bit network address of this route.
Status	3 bits	The status of the route. See Table 136 below for values.
Next-hop address	2 bytes	The 16-bit network address of the next hop on the way to the destination.

Table 136 enumerates the values for the route status field.

Table 136 Route status values

Numeric Value	Status
0x0	ACTIVE
0x1	DISCOVERY_UNDERWAY
0x2	DISCOVERY_FAILED
0x3	INACTIVE
0x4 – 0x7	Reserved

[129]CCB Comment #133
[130]CCB Comment #133
[131]CCB Comment #255

In the text below that describes the routing algorithm the term “routing table capacity” is used to describe the situation in which a device has the ability to use its routing table to establish a route to a particular destination device. A device is said to have routing table capacity if:

— It is a ZigBee coordinator or ZigBee router.

— It maintains a routing table.

— It has a free routing table entry or it already has a routing table entry corresponding to the destination.

— The device is attempting route repair and it has reserved some entries for this purpose as described above.

If a ZigBee router or ZigBee coordinator maintains a routing table it shall also maintain a route discovery table containing the information shown in Table 137. Routing table entries are long-lived and persistent, while route discovery table entries last only as long as the duration of a single route discovery operation and may be reused.

Table 137 Route discovery table

Field Name	Size	Description
Route request ID	1 byte	A sequence number for a route request command frame that is incremented each time a device initiates a route request.
Source address	2 bytes	The 16-bit network address of the route request’s initiator.
Sender address	2 bytes	The 16-bit network address of the device that has sent the most recent lowest cost route request command frame corresponding to this entry’s Route request identifier and Source address. This field is used to determine the path that an eventual route reply command frame should follow.
Forward Cost	1 byte	The accumulated path cost from source of the route request to the current device.
Residual cost	1 byte	The accumulated path cost from the current device to the destination device.
Expiration time	2 bytes	A countdown timer indicating the number of milliseconds until route discovery expires. The initial value is *nwkcRouteDiscoveryTime*.

A device is said to have “route discovery table capacity” if:

— It maintains a route discovery table.

— It has a free entry in its route discovery table.

If a device has both routing table capacity and route discovery table capacity then it may be said the have “routing capacity”.

2.7.3.3 Upon receipt of a data frame

On receipt of a data frame the NWK layer routes it according to the following procedure, which is also outlined in Figure 57.

If a data frame is received by the NWK layer from its next higher layer and the destination address is equal to the broadcast address, the NWK layer shall broadcast the frame according to the procedures described in sub-clause 2.7.5.

If the receiving device is a ZigBee router or ZigBee coordinator and the destination of the frame is a ZigBee end device which is also the child of the receiving device then the frame shall be routed directly to the destination using the MCPS[132]-DATA.request primitive as described in sub-clause 2.7.2.1 and setting the next hop destination address equal to the final destination address.

A device that has routing capacity shall examine the discover route sub-field of the NWK header frame control field, If the discover route sub-field has a value of 0x02 then the device shall initiate route discovery immediately as described below in sub-clause 2.7.3.4.1. If the discover route sub-field does not have a value of 0x02, the device shall check its routing table for an entry corresponding to the destination of the frame. If there is such an entry, and if the value of the route status field for that entry is ACTIVE, then the device shall relay the frame using the MCPS-DATA.request primitive. When relaying a data frame, the SrcAddrMode and DstAddrMode parameters of the MCPS-DATA.request primitive shall both have a value of 0x02, indicating 16-bit addressing. The SrcPANId and DstPANId parameters shall both have the value provided by the macPANId attribute of the MAC PIB for the relaying device. The SrcAddr parameter will be set to the value of macShortAddress from the MAC PIB of the relaying device, and the DstAddr parameter shall be the value provided by the next-hop address field of the routing table entry corresponding to the destination. The TxOptions parameter should always be non-zero when bitwise ANDed with the value 0x01, indicating acknowledged transmission. If the device has a routing table entry corresponding to the destination of the frame but the value of the route status field for that entry is DISCOVERY_UNDERWAY, then the frame shall be treated as though route discovery has been initiated for this frame, as described below in sub-clause 2.7.3.4.1. The frame may optionally be buffered pending route discovery or else routed along the tree using hierarchical routing, provided that the NIB attribute *nwkUseTreeRouting* has a value of TRUE. If the frame is routed along the tree, the discover route sub-field of the NWK header frame control field shall be set to 0x00. If the device has a routing table entry corresponding to the destination of the frame but value of the route status field for that entry has a value of DISCOVERY_FAILED or INACTIVE, the device may route the frame along the tree using hierarchical routing, again provided that the NIB attribute *nwkUseTreeRouting* has a value of TRUE. If the device does not have a routing table entry for the destination, it shall examine the discover route sub-field of the NWK header frame control field. If the discover route sub-field has a value of 0x01 then the device shall initiate route discovery as described below in sub-clause 2.7.3.4.1. If the discover route sub-field has a value of 0 and the NIB attribute *nwkUseTreeRouting* has a value of TRUE then the device shall route along the tree using hierarchical routing. If the discover route sub-field has a value of 0, the NIB attribute *nwkUseTreeRouting* has a value of FALSE and there is no routing table corresponding to the destination of the frame, the frame shall be discarded and the NLDE shall issue the NLDE-DATA.confirm primitive with a Status value of INVALID_REQUEST.[133]

A device without routing capacity shall route along the tree using hierarchical routing, again provided that the value of the NIB attribute *nwkUseTreeRouting* is TRUE.[134]

For hierarchical routing, if the destination is a descendant of the device, the device shall route the frame to the appropriate child. If the destination is a child, and it is also and end device, delivery of the frame can fail due to the *macRxOnWhenIdle* state of the child device. In the case when the child has *macRxOnWhenIdle* set to FALSE, indirect transmission as described in [B1] may be used be used to deliver the frame. If the destination is not a descendant, the device shall route the frame to its parent.

Trivially every other device in the network is a descendant of the ZigBee coordinator and no device in the network is the descendant of any ZigBee end device. For a ZigBee router with address A_n at depth d, if the following logical expression is true, then a destination device with address D is a descendant:

[132]CCB Comment #119
[133]CCB Comment #129, 258, 122
[134]CCB Comment #129

$$A < D < A + Cskip(d-1)$$.

For a definition of $Cskip(d)$, see sub-clause 2.7.1.4.

If it is determined that the destination is a descendant of the receiving device, the address N of the next hop device is given by:

$$N = D$$

for ZigBee end devices, where $D > A + Rm \times Cskip(d)$, and[135]

$$N = A + 1 + \left\lfloor \frac{D - (A+1)}{Cskip(d)} \right\rfloor \times Cskip(d)$$

otherwise.

If a data frame is received by the NWK layer from the MAC sub-layer and the destination address is equal to the broadcast address, the NWK layer shall first re-broadcast the frame and then send it to its next higher layer for processing. If the data frame is received by the MAC and is not to be broadcast, the NWK layer shall determine whether the destination address is equal to its own logical address. If so, the NWK layer shall send the frame to its next higher layer for processing. Otherwise the device is an intermediate device. In this case, the device shall follow the same procedure outlined above for the case of receiving a unicast frame from the next higher layer.

[135]CCB Comment #228 specifies a modified formula here. The original was:

$$A + 1 + \left\lfloor \frac{D - A + 1}{Cskip(d)} \right\rfloor \times Cskip(d)$$

Figure 57 Basic routing algorithm

2.7.3.4 Route discovery

Route discovery is the procedure whereby network devices cooperate to find and establish routes through the NWK and is always performed with regard to a particular source and destination device.

2.7.3.4.1 Initiation of route discovery

The route discovery procedure shall be initiated by the NWK layer on receipt of an NLDE-DATA.request primitive from a higher layer with the DiscoverRoute parameter set to 0x02, or on receipt of an NLDE-DATA.request primitive from a higher layer with the DiscoverRoute parameter set to 0x01for which there is

no routing table entry corresponding to the DstAddr parameter, or on receipt of a frame from the MAC sub-layer for which the value of the destination address field in the NWK header is not that of the current device or the broadcast address and in which either the discover route sub-field of the frame control field has a value of 0x02 or the discover route sub-field of the frame control field has a value of 0x01 and there is no routing table entry corresponding to the value of the destination address field of the NWK header. In either case, if the device does not have routing capacity and the NIB attribute *nwkUseTreeRouting* has a value of TRUE, the data frame in question shall be routed along the tree using hierarchical routing. If the device does not have routing capacity and the NIB attribute *nwkUseTreeRouting* has a value of FALSE, then the frame shall be discarded.[136]

If the device has no existing routing table entry for the destination it shall establish a routing table entry with status equal to DISCOVERY_UNDERWAY. If the device has an existing routing table entry corresponding to the destination address with status equal to ACTIVE, then that entry shall be used and the status field of that entry shall remain ACTIVE. If it has an existing routing table entry with some other status value than ACTIVE then that entry shall be used and the status of that entry shall be set to DISCOVERY_UNDERWAY. The device shall also establish the corresponding route discovery table entry if one does not already exist.[137]

Each device issuing a route request command frame shall maintain a counter used to generate route request identifiers. When a new route request command frame is created the route request counter is incremented and the value is stored in the device's route discovery table in the Route request identifier field. Other fields in the routing table and route discovery table are set as described in sub-clause 2.7.3.2. The route request timer in the route discovery table shall be set to expire in *nwkcRouteDiscoveryTime* milliseconds when the timer expires, the device shall delete the route request entry from the route discovery table. When this happens, the routing table entry corresponding to the destination address shall also be deleted if its Status field still has a value of DISCOVERY_UNDERWAY and there are no other entries with the same destination field value in the route discovery table[138].

The NWK layer may choose to buffer the received frame pending route discovery or else, if the NIB attribute *nwkUseTreeRouting* has a value of TRUE, set[139] the discover route sub-field of the frame control field in the NWK header to 0 and forward the data frame along the tree.

Once the device creates the route discovery table and routing table entries, the route request command frame shall be created with the payload depicted in Figure 41. The individual fields are populated as follows. The command frame identifier field shall be set to indicate the command frame is a route request, see Figure 129. The Route request identifier field shall be set to the value stored in the route discovery table entry. The destination address field shall be set to the 16-bit network address of the device for which the route is to be discovered. The path cost field shall be set to 0. Once created the route request command frame is ready for broadcast and is passed to the MAC sub-layer using the MCPS-DATA.request primitive.

When broadcasting a route request command frame at the initiation of route discovery, the NWK layer shall retry the broadcast *nwkcInitialRREQRetries* times after the initial broadcast, resulting in a maximum of *nwkcInitialRREQRetries* + 1 transmission. The retries will be separated by a time interval of *nwkcRREQRetryInterval* milliseconds.

2.7.3.4.2 Upon receipt of a route request command frame

Upon receipt of a route request command frame the device shall determine if it has routing capacity.

If the device does not have routing capacity, it shall check if the frame was received along a valid path. A path is valid if the frame was received from one of the device's child devices and the source device is a

[136] CCB Comment #122, 256
[137] CCB Comment #136, #260
[138] CCB Comment #260
[139] CCB Comment #122

descendant of that child device or if the frame was received from the device's parent device and the source device is not a descendant of the device. If the route request command frame was not received along a valid path, it shall be discarded. Otherwise the device shall check if it is the intended destination. It shall also check if the destination of the command frame is one of its end-device children by comparing the destination address field of the route request command frame payload with the address of each of its end-device children, if any. If either the device or one of its end-device children is the destination of the route request command frame, it shall reply with a route reply command frame. When replying to a route request with a route reply command frame, the device shall construct a frame with the frame type field set to 0x01. The route reply's source address shall be set to the 16-bit network address of the device creating the route reply and the destination address shall be set to the calculated next hop address, considering the originator of the route request as a destination. The link cost from the next hop device to the current device shall be computed as described in sub-clause 2.7.3.1 and inserted into the path cost field of the route reply command frame. The route reply command frame shall be unicast to the next hop device by issuing an MCPS-DATA.request primitive. If the device is not the destination of the route request command frame, the device shall compute the link cost from the previous device that transmitted the frame, as described in sub-clause 2.7.3.1. This value shall be added to the path cost value stored in the route request command frame. The route request command frame shall then be unicast towards the destination using the MCPS-DATA.request service primitive. The next hop for this unicast transmission is determined in the same manner as if the frame were a data frame addressed to the device identified by the destination address field in the payload.

If the device does have routing capacity (see Figure 58), it shall check if it is the destination of the command frame by comparing the destination address field of the route request command frame payload with its own address. It shall also check if the destination of the command frame is one of its end-device children by comparing the destination address field of the route request command frame payload with the address of each of its end-device children, if any. If either the device or one of its end-device children is the destination of the route request command frame, the device shall determine if a route discovery table (see Table 137) entry exists with the same route request identifier and source address field. If no such entry exits then one shall be created. When creating the route discovery table entry, the fields are set to the corresponding values in the route request command frame. The only exception is the forward cost field, which is determined by using the previous sender of the command frame to compute the link cost as described in sub-clause 2.7.3.1 and adding it to the path cost contained the route request command frame. The result of the above calculation is stored in the forward cost field of the newly created route discovery table entry. If the *nwkSymLink* attribute is set to true, the device shall also create a routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then issue a route reply command frame to the source of the route request command frame. In the case where the device already has a route discovery table entry for the source address and route request identifier pair, the device shall determine if the path cost in the route request command frame is less than the forward cost stored in the route discovery table entry. The comparison is made by first computing the link cost from the previous device that sent this frame as described in sub-clause 2.7.3.1 and adding it to the path cost value in the route request command frame. If this value is greater than the value in the route discovery table entry then the frame shall be dropped and no further processing is required. Otherwise the forward cost and sender address fields in the route discovery table are updated with the new cost and the previous device address from the route request command frame. If the *nwkSymLink* attribute is set to true, the device shall also create a routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then respond with a route reply command frame. In either of the above cases, if the device is responding on behalf of one of its end-device children, the responder address in the route reply command frame payload shall be set equal to the address of the end device child and not of the responding device.

When a device with routing capacity is not the destination of the received route request command frame, it shall determine if a route discovery table entry (see Table 137) exists with the same route request identifier and source address field. If no such entry exits then one shall be created. The route request timer shall be set

to expire in *nwkcRouteDiscoveryTime* milliseconds. If a routing table entry corresponding to the destination exists and its status is not ACTIVE, the status shall be set to DISCOVERY_UNDERWAY.[140] If no such entry exists, it shall be created and its status set to DISCOVERY_UNDERWAY. If the *nwkSymLink* attribute is set to true, the device shall also create a routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. When the route request timer expires, the device deletes the route request entry from the route discovery table. When this happens, the routing table entry corresponding to the destination address shall also be deleted if its status field has a value of DISCOVERY_UNDERWAY and there are no other entries in the route discovery table created as a result of a route discovery for that destination address.[141] If an entry in the route discovery table already exists then the path cost in the route request command frame shall be compared to the forward cost value in the route discovery table entry. The comparison is made by computing the link cost from the previous device, as described in sub-clause 2.7.3.1, and adding it to the path cost value in the route request command frame. If this path cost is greater, the route request command frame is dropped and no further processing is required. Otherwise the forward cost and sender address fields in the route discovery table are updated with the new cost and the previous device address from the route request command frame. Additionally, the path cost field in the route request command frame shall be updated with the cost computed for comparison purposes. If the *nwkSymLink* attribute is set to true, the device shall also update any routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then rebroadcast the route request command frame using the MCPS-DATA.request primitive.

When rebroadcasting a route request command frame, the NWK layer shall delay retransmission by a random jitter amount calculated using the formula:

$$2 \times R[nwkcMinRREQJitter, nwkcMaxRREQJitter]$$

where $R[a,b]$ is a random function on the interval $[a,b]$. The units of this jitter amount are milliseconds. Implementers may adjust the jitter amount so that route request command frames arriving with large path cost are delayed more than frames arriving with lower path cost. The NWK layer shall retry the broadcast *nwkcRREQRetries* times after the original relay resulting in a maximum of *nwkcRREQRetries* + 1 relays per relay attempt. Implementers may choose to discard route request command frames awaiting retransmission in the case that a frame with the same source and route request identifier arrives with a lower path cost than the one awaiting retransmission.

The device shall also set the status field of the routing table entry corresponding to the destination address field in the payload to DISCOVERY_UNDERWAY. If no such entry exists, it shall be created.

When replying to a route request with a route reply command frame, a device that has a route discovery table entry corresponding to the source address and route request identifier of the route request shall construct a command frame with the frame type field set to 0x01. The source address field of the network header shall be set to the 16-bit network address of the current device and the destination address field shall be set to the the value of the sender address field from the corresponding route discovery table entry. The device constructing the route reply shall populate the payload fields in the following manner. The NWK command identifier shall be set to route reply. The route request identifier field shall be set to the same value found in the route request identifier field of the route request command frame. The originator address field shall be set to the source address in the network header of route request command frame. Using the sender address field from the route discovery table entry corresponding to the source address in the network header of route request command frame, the device shall compute the link cost as described in sub-clause 2.7.3.1. This link cost shall be entered in the path cost field. The route reply command frame is then unicast to the

[140] CCB Comment #260
[141] Ibid

destination by using MCPS-DATA.request primitive and the sender address obtained from the route discovery table as the next hop.

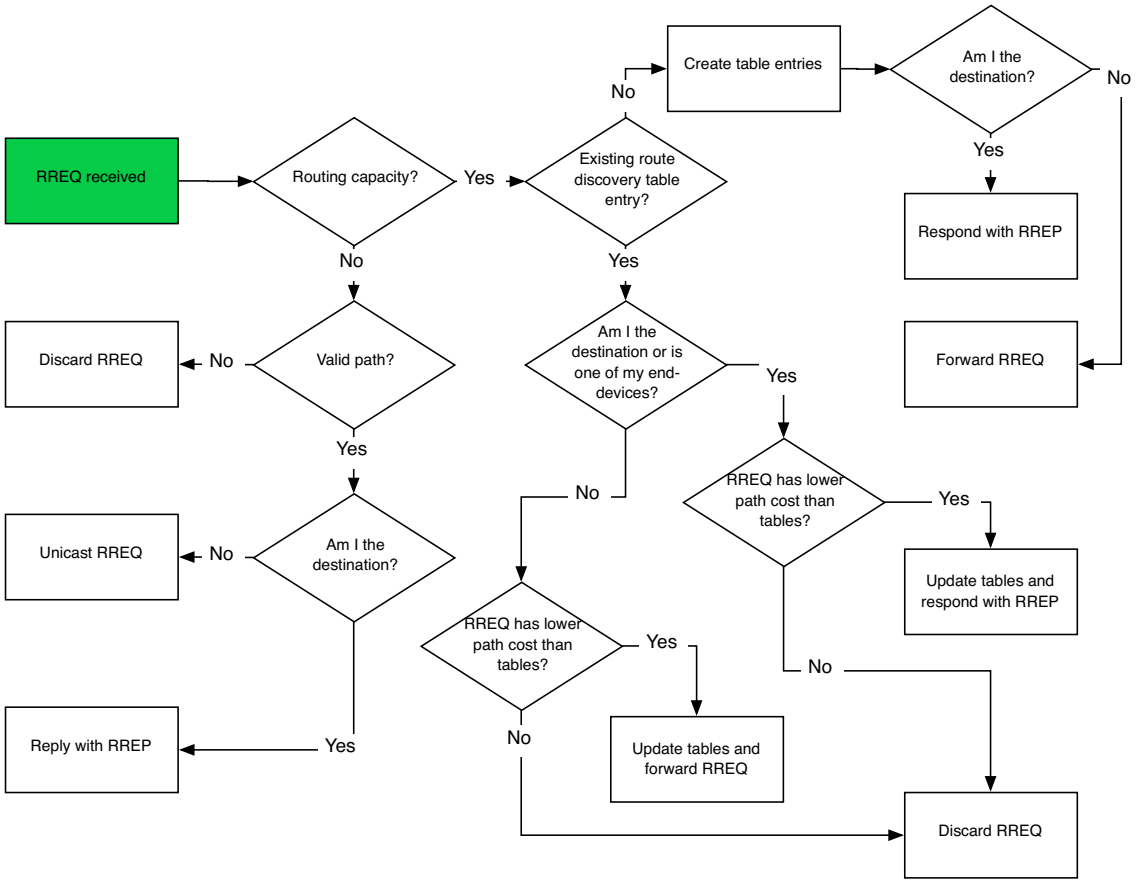


Figure 58 Receipt of route request

2.7.3.4.3 Upon receipt of route reply command frame

On receipt of a route reply command frame, a device performs the procedure outlined in Figure 59.

If the receiving device has no routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of TRUE, it shall forward the route reply using tree routing. If the receiving device has no routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of FALSE, it shall discard the command frame.[142] Before forwarding the route reply command frame the device shall update the path cost field in the payload by computing the link cost from the next hop device to itself as described in sub-clause 2.7.3.1 and adding this to the value in the route reply path cost field.

If the receiving device has routing capacity, it shall check whether it is the destination of the route reply command frame by comparing the contents of the originator address field of the command frame payload with its own address. If so, it shall search its route discovery table for an entry corresponding to the route request identifier in the route reply command frame payload. If there is no such entry, the route reply command frame shall be discarded and route reply processing shall be terminated. If a route discovery table

[142]CCB Comment #122

entry exists, the device shall search its routing table for an entry corresponding to the responder address in the route reply command frame. If there is no such routing table entry the route reply command frame shall be discarded and, if a route discovery table entry corresponding to the route request identifier in the route reply command frame exists, it shall also be removed, and route reply processing shall be terminated. If a routing table entry and a route discovery table entry exist and if the status field of the routing table entry is set to DISCOVERY_UNDERWAY, it shall be changed to active and the next hop field in the routing table shall be set to the previous device that forwarded the route reply command frame. The residual cost field in the route discovery table entry shall be set to the path cost field in the route reply payload.

If the status field was already set to ACTIVE, the device shall compare the path cost in the route reply command frame to the residual cost recorded in the route discovery table entry, and update the residual cost field and next hop field in the routing table entry if the cost in the route reply command frame is smaller. If the path cost in the route reply is not smaller, the route reply shall be discarded and no further processing shall take place.

If the device receiving the route reply is not the destination, the device shall find the route discovery table entry corresponding to the originator address and route request identifier in the route reply command frame payload. If no such route discovery table entry exists, the route reply command frame shall be discarded. If a route discovery table entry exists, the path cost value in the route reply command frame and the residual cost field in the route discovery table entry shall be compared. If the route discovery table entry value is less than the route reply value, the route reply command frame shall be discarded. Otherwise, the device shall find the routing table entry corresponding to the responder address in the route reply command frame. It is an error here if the route discovery table entry exists and there is no corresponding routing table entry, and the route reply command frame should be discarded. The routing table entry shall be updated by replacing the next hop field with the address of the previous device that forwarded the route reply command frame. The route discovery table entry shall also be updated by replacing the residual cost field with the value in the route reply command frame.

After updating its own route entry, the device shall forward the route reply to the destination. Before forwarding the route reply, the path cost value shall be updated. The sender shall find the next hop to the route reply's destination by searching its route discovery table for the entry matching the route request identifier and the source address and extracting the sender address. It shall use this next hop address to compute the link cost as described in sub-clause 2.7.3.1. This cost shall be added to the path cost field in the route reply. The destination address in the command frame network header shall be set to the next hop address and the frame shall be unicast to the next hop device using the MCPS-DATA.request primitive.The DstAddr parameter of the MCPS-DATA.request primitive shall be set to the next-hop address from the route discovery table.[143]

[143]CCB Comment #131

Figure 59 Receipt of route reply

2.7.3.5 Route maintenance

A device NWK layer shall maintain a failure counter for each neighbor to which it has an outgoing link, i.e. to which it has been required to send data frames. If the value of the outgoing link failure counter ever exceeds nwkcRepairThreshold then the device shall initiate route repair as described in the following paragraphs. Implementers may choose a simple failure-counting scheme to generate this failure counter value or they may use a more accurate time-windowed scheme. Note that it is important not to initiate repair too frequently since repair operations may flood the network and cause other traffic disruptions. The procedure for retiring links and ceasing to keep track of their failure counter is out of the scope of this specification.

2.7.3.5.1 Route repair for mesh network topology

When a link or a device fails in mesh network topology, the upstream device shall initiate route repair. If the upstream device is unable to initiate route repair due to a lack of routing capacity or some other limitation, the device shall issue a route error command frame back to the source device with the error code indicating the reason for the failure (see Table 130).

If the upstream device is able to initiate route repair, it shall do so by broadcasting a route request command frame with the source address set to its own address and the destination address set to the destination address of the frame that failed in transmission. The route request command frame shall have the route repair sub-field in the command options field of the command frame payload set to 1 indicating route repair.

While a device is performing route repair for a particular destination, a device shall not forward frames to that destination. Any frames that it has pending for that destination at the time route repair is initiated and any frames for that destination that arrive before the completion of route repair shall either be buffered until the completion of route repair or discarded depending on the capabilities of the device.[144]

On receipt of a route request command frame a routing node shall perform the procedure outlined in sub-clause 2.7.3.4.2. If the routing node is the destination of the route request command frame or the destination is one of its end-device children, it shall reply with a route reply command frame. The route reply command frame shall have the route repair sub-field in the command options field of the command frame payload set to 1 indicating route repair.

If a route reply command frame does not arrive at the upstream device within *nwkcRouteDiscoveryTime* milliseconds, the upstream device shall send a route error command frame to the source device. If the upstream device does receive a route reply within the designated time, it will forward any data that it may have buffered pending the repair to the destination.

If the source device receiving a route error command frame does not have routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of TRUE[145], it shall construct a route request command frame as described in sub-clause 2.7.3.4.1 and unicast the command frame towards its destination along the tree using hierarchical routing. If the source device does have routing capacity, it shall initiate normal route discovery as described in sub-clause 2.7.3.4.1.

If an end device that is also an RFD is unable to transmit messages to its parent, the end device shall initiate the orphaning procedure, as described in [B1]. If the orphaning procedure is successful and the end device re-establishes communications with its parent, the end device shall resume operation on the network as before. If the orphaning procedure fails, the end device shall attempt to re-join the network through a new parent. In this case the new parent shall issue the end device a new 16-bit network address. If the end device is unable to locate a new parent because there is no other device in its neighborhood with the capacity to accept an additional child device, the end device will not be able to re-join the network. In this case, user intervention may be necessary to enable the end device to re-join.

2.7.3.5.2 Route repair for tree network topology

When a downstream device loses synchronization with its parent beacon, indicated by the MAC sub-layer through the MLME-SYNC-LOSS.indication primitive, or is unable to transmit a message to its parent, the device may either initiate the orphaning procedure to search for its parent or the association procedure to find a new parent.[B1] If either the orphaning procedure fails or the device associates with a new upstream device, the downstream device will receive a new 16-bit network address from its new parent and resume operation on the network. This allows the network to continue operating in a true tree configuration.

Before a device attempts to re-join the network and receive a new 16-bit network address, the device shall use the MAC sub-layer disassociation procedure to disassociate all of its children. If the device is unable to reach one or more of its children, it shall consider the child(ren) disassociated from the network and remove the 16-bit addresses of the children from its neighbor table. The device shall then re-join the network and begin operation with its new address.

Similarly if a disassociated child has its own children, it shall disassociate them from the network before attempting re-association. If the child is able to re-associate either through a new parent or through its original parent, the child shall receive a new 16-bit network address and begin operation on the network using the new address.

[144] CCB Comment #124
[145] CCB Comment #122

Optionally, if the implementer chooses not to seek a new 16-bit network address, the network will be partitioned by a link failure. The remaining portions of the partitioned network would then operate separately.

If an upstream device is unable to transmit a message to one of its children it may drop the message and send a route error command frame to the originating device to indicate that the message has not been delivered.

2.7.4 Scheduling beacon transmissions

Beacon scheduling is necessary in a multihop topology to prevent the beacon frames of one device from colliding with either the beacon frames or data transmissions of its neighboring devices. Beacon scheduling is necessary when implementing a tree topology but not a mesh topology, as beaconing is not permitted in ZigBee mesh networks.

2.7.4.1 Scheduling method

The ZigBee coordinator shall determine the beacon order and superframe order for every device in the network (see [B1] for more information on these attributes). Because one purpose of multihop beaconing networks is to allow routing nodes the opportunity to sleep in order to conserve power, the beacon order shall be set much larger than the superframe order. Setting the attributes in this manner makes it possible to schedule the active portion of the superframes of every device in any neighborhood such that they are non-overlapping in time. In other words, time is divided into approximately (*macBeaconInterval*/*macSuperframeDuration*) non-overlapping time slots, and the active portion of the superframe of every device in the network shall occupy one of these non-overlapping time slots. An example of the resulting frame structure for a single beaconing device is shown in Figure 60.

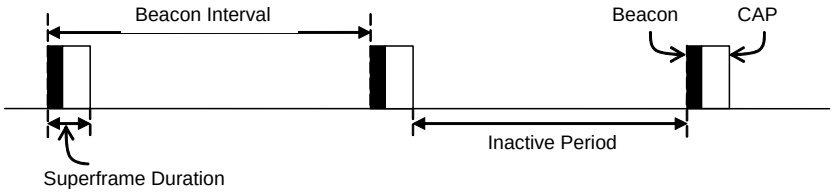


Figure 60 Typical frame structure for a beaconing device

The beacon frame of a device shall be transmitted at the start of its non-overlapping time slot, and the transmit time shall be measured relative to the beacon transmit time of the parent device. This time offset shall be included in the beacon payload of every device in a multihop beaconing network (see sub-clause 2.7.6 for a complete list of beacon payload parameters). Therefore a device receiving a beacon frame shall know the beacon transmission time of both the neighboring device and the parent of the neighboring device, since the transmission time of the parent may be calculated by subtracting the time offset from the timestamp of the beacon frame. The receiving device shall store both the local timestamp of the beacon frame and the offset included in the beacon payload in its neighbor table. The purpose of having a device know when the parent of its neighbor is active is to maintain the integrity of the parent-child communication link by alleviating the hidden node problem. In other words, a device will never transmit at the same time as the parent of its neighbor.

Communication in a tree network shall be accomplished using the parent-child links to route along the tree. Since every child tracks the beacon of its parent, transmissions from a parent to its child shall be completed using the indirect transmission technique. Transmissions from a child to its parent shall be completed during the CAP of the parent. Details for the communication procedures can be found in [B1].

A new device wishing to join the network shall follow the procedure outlined in sub-clause 2.3.6. In the process of joining the network, the new device shall build its neighbor table based on the information collected during the MAC scan procedure. Using this information, the new device shall choose an appropriate time for its beacon transmission and CAP (the active portion of its superframe structure) such that the active portion of its superframe structure does not overlap with that of any neighbor or of the parent of any neighbor. If there is no available non-overlapping time slot in the neighborhood, the device shall not transmit beacons and shall operate on the network as an end device. If a non-overlapping time slot is available, the time offset between the beacon frames of the parent and of the new device shall be chosen and included in the beacon payload of the new device. Any algorithm for selecting the beacon transmission time that avoids beacon transmission during the active portion of the superframes of its neighbors and their parents may be employed, as interoperability will be ensured.

To counteract drift, the new device shall track the beacon of its parent and adjust its own beacon transmission time such that the time offset between the two remains constant. Therefore the beacon frames of every device in the network are essentially synchronized with those of the ZigBee coordinator. Figure 61 illustrates the relationship between the active superframe portions of a parent and its child.

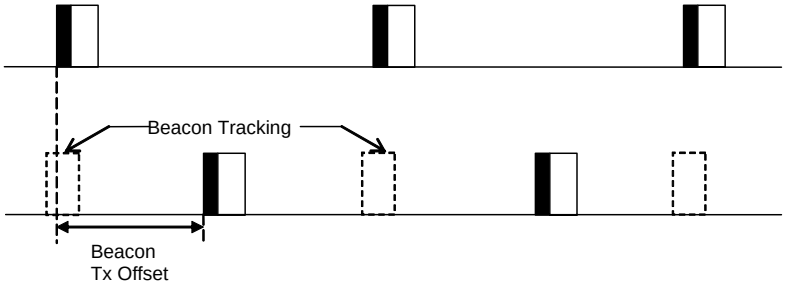


Figure 61 Parent-child superframe positioning relationship

The density of devices that can be supported in the network is inversely proportional to the ratio of the superframe order to the beacon order. The smaller the ratio, the longer the inactive period of each device and the more devices that can transmit beacon frames in the same neighborhood. It is recommended that a tree network utilize a superframe order of 0, which gives a superframe duration of 15.36 ms, and a beacon order of between 6 and 10, which gives a beacon interval between 0.98304s and 15.72864s. Using these superframe and beacon order values, a typical duty cycle for devices in the network will be between ~2% and ~0.1%.

2.7.4.2 MAC enhancement

In order to employ the beacon scheduling algorithm just described, it is necessary to implement the following enhancement to the IEEE Std 802.15.4-2003[146] MAC sub-layer.

[146] CCB Comment #265

A new parameter, StartTime, shall be added to the MLME-START.request primitive to specify the time to begin transmitting beacons. The new format of the primitive is as follows:

MLME-START.request	(PANID, LogicalChannel, BeaconOrder, SuperframeOrder, PANCoordinator, BatteryLifeExtention, CoordRealignment, SecurityEnable, StartTime[a])

[a]CCB Comment #265

The StartTime parameter is fully described in Table 138, and the description of all other parameters can be found in [B1].

Table 138 Start time for beacon transmissions

Name	Type	Valid range	Description
StartTime	Integer	0x000000-0xffffff	The time at which to begin transmitting beacons. If the device issuing the primitive is the PAN coordinator, this parameter is ignored and beacon transmissions will begin immediately. Otherwise, this parameter specifies the time relative to the received beacon of the device with which it is associated. The parameter is specified in symbols and is rounded to a backoff slot boundary. The precision of this value is a minimum of 20 bits, with the lowest 4 bits being the least significant.[a]

[a]CCB Comment #265

2.7.5 Broadcast communication

This sub-clause specifies how a broadcast transmission is accomplished within a ZigBee network. This mechanism is used to broadcast all network layer data frames[147]. Any device within a network may initiate a broadcast transmission intended for all other devices that are part of the same network. A broadcast transmission is initiated by the local APS sub-layer entity through the use of the NLDE-DATA.request primitive by setting the DstAddr parameter to 0xffff.[148]

To transmit a broadcast MSDU, the NWK layer issues an MCPS-DATA.request primitive to the MAC sub-layer with the DstAddrMode parameter set to 0x02 (16-bit network address) and the DstAddr parameter set to 0xffff, which is the broadcast network address. The PANId parameter shall be set to the PANId of the ZigBee network. This specification does not support broadcasting across multiple networks. Broadcast transmissions shall not use the MAC sub-layer acknowledgement; instead a passive acknowledgement mechanism is used in the case of non-beacon-enabled ZigBee networks. Passive acknowledgement means that every device keeps track if its neighboring devices have successfully relayed the broadcast

[147]CCB Comment #201
[148]CCB Comment #125

transmission. The MAC sub-layer acknowledgement is disabled by setting the acknowledged transmission flag of the TxOptions parameter to FALSE. All other flags of the TxOptions parameter shall be set based on the network configuration.

. Each device shall keep a record of any new broadcast transaction that is either initiated locally or received from a neighboring device. This record is called the broadcast transaction record (BTR) and shall contain at least the sequence number and the source address of the broadcast frame. The broadcast transaction records are stored in the broadcast transaction table (BTT).[149]

When a device receives a broadcast frame from a neighboring device**,** it shall compare the Sequence number[150] and the source address of the broadcast frame with the records in its BTT. If the device has a BTR of this particular broadcast frame in its BTT, it shall update the BTR to mark the neighboring device as having relayed the broadcast frame. It shall then drop the frame. If no record is found, it shall create a new BTR in its BTT and shall mark the neighboring device as having relayed the broadcast. The NWK layer shall then indicate to the higher layer that a new broadcast frame has been received. If the radius[151] field value is greater than 0 it shall retransmit the frame. Otherwise it shall drop the frame. Before the retransmission, it shall wait for a random time period called broadcast jitter. This time period shall be bounded by the value of the *nwkcMaxBroadcastJitter* attribute.

If, on receipt of a broadcast frame, the NWK layer finds that the BTT is full and contains no expired entries, then the frame should be ignored. In this situation the frame should not be retransmitted, nor should it be passed up to the next higher layer.[152]

A device, operating in a non-beacon-enabled ZigBee network, shall retransmit a previous broadcast frame if any of its neighboring devices have not relayed the broadcast frame within *nwkPassiveAckTimeout* seconds. In this case a device shall retransmit a broadcast frame for at most *nwkMaxBroadcastRetries* times.

A device should change the status of a BTT entry after *nwkNetworkBroadcastDeliveryTime* seconds have elapsed since its creation. The entry should change status to expired and thus the entry can be overwritten if required when a new broadcast is received.[153]

When a ZigBee router that has the *macRxOnWhenIdle* MAC PIB attribute set to FALSE receives a broadcast transmission, it shall use a different procedure for retransmission than the one outlined above. It shall retransmit the frame without delay to each of its neighbors individually, using a MAC layer unicast, i.e. with the DstAddr parameter of the MCPS-DATA.request primitive set to the address of the receiving device and not to the broadcast address. Similarly, a router with the *macRxOnWhenIdle* MAC PIB attribute set to TRUE, which has one or more neighbors with the *macRxOnWhenIdle* MAC PIB parameter set to FALSE, shall retransmit the broadcast frame to each of these neighbors in turn as a MAC layer unicast in addition to performing the more general broadcast procedure spelled out in the previous paragraphs. Indirect transmission, as described in [B1], may be employed to ensure that these unicasts reach their destination.

Every ZigBee router shall have the ability to buffer at least 1 frame at the NWK layer in order to facilitate retransmission of broadcasts.

Figure 62 shows a broadcast transaction between a device and two neighboring devices**.**

[149]CCB Comment #111
[150]Ibid
[151]Ibid
[152]CCB Comment #108, 110
[153]CCB Comment #110

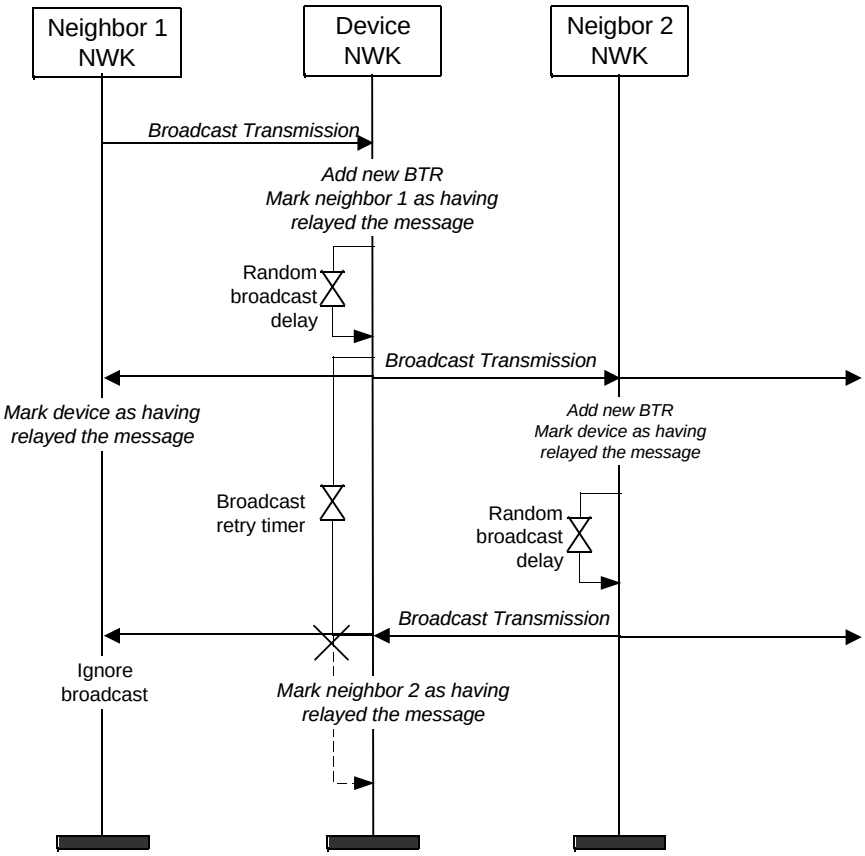


Figure 62 Broadcast transaction message sequence chart

2.7.6 NWK information in the MAC beacons

This sub-clause specifies how the NWK layer uses the beacon payload of a MAC sub-layer beacon frame to convey NWK layer-specific information to neighboring devices.

When the association permit subfield of the superframe specification field of the beacon frame of the device, as defined in [B1], is set to 1 indicating that association is permitted, then the beacon payload shall contain the information shown in Table 139. This enables the NWK layer to provide additional information to new devices that are performing network discovery and allows these new devices to more efficiently select a network and particular neighbor to join. Refer to sub-clause 2.7.1.3.1.1 for a detailed description of the network discovery procedure. This information is not required to be in the beacon payload when the

association permit subfield of the superframe specification field of the beacon frame of the device is set to 0 indicating that association is not permitted.

Table 139 NWK layer information fields[a]

Name	Type	Valid range	Description
Protocol ID	Integer	0x00 – 0xff	This field identifies the network layer protocols in use and, for purposes of this specification shall always be set to 0, indicating the ZigBee protocols. The value 0xff shall also be reserved for future use by the ZigBee alliance.
Stack profile	Integer	0x00 – 0x0f	A ZigBee stack profile identifier.
nwkcProtocolVersion	Integer	0x00 – 0x0f	The version of the ZigBee protocol.
Router capacity[b]	Boolean	TRUE or FALSE	This value is set to TRUE if this device is capable to accept join requests from router-capable devices and is set to FALSE otherwise.
Device depth	Integer	0x00 – *nwkMaxDepth*[c]	The tree depth of this device. A value of 0x00 indicates that this device is the ZigBee coordinator for the network.
End[d] device capacity	Boolean	TRUE or FALSE	This value is set to TRUE if the device is capable of accepting join requests from end devices seeking to join the network and is set to FALSE otherwise.[e]
TxOffset	Integer	0x000000 – 0xffffff	This value indicates the difference in time, measured in symbols, between the beacon transmission time of the device and the beacon transmission time of its parent. This offset may be subtracted from the beacon transmission time of the device to calculate the beacon transmission time of the parent. This parameter is only included when implementing a multihop beaconing network.

[a]CCB Comment #129
[b]Ibid
[c]CCB Comment #229
[d]CCB Comment #129
[e]Ibid

The NWK layer of the ZigBee coordinator shall update the beacon payload immediately following network formation. All other ZigBee devices shall update it immediately after the association is completed and anytime the network configuration (any of the parameters specified in Table 106) changes.

The beacon payload is written into the MAC sub-layer PIB using the MLME-SET.request primitive. The *macBeaconPayloadLength* attribute is set to the length of the beacon payload, and the byte sequence representing the beacon payload is written into the *macBeaconPayload* attribute. The formatting of the byte sequence representing the beacon payload is shown in Figure 63.

Bits: 0-7	8-11	12-15	16-17	18	19[a]-22	23	24-47
Protocol ID	Stack profile	nwkcProtocol-Version	Reserved[b]	Router capacity[c]	Device depth	End[d] device capacity	Tx Offset (optional)

[a]CCB Comment #229
[b]CCB Comment #129
[c]Ibid
[d]Ibid

Figure 63 Format of the MAC sub-layer beacon payload

2.7.7 Persistent data

Devices operating in the field may be reset manually or programmatically by maintenance personnel, or may be reset accidentally for any number of reasons, including localized or network-wide power failures, battery replacement during the course of normal maintenance, impact and so on. At a minimum, the following information must be preserved across resets in order to maintain an operating network:

- The device's PAN ID.
- The device's 16-bit network address.
- The 64-bit IEEE address and 16-bit network address of each associated child.
- The stack profile in use.
- The values of nwkNextAddress and nwkAvailableAddresses NIB attributes, if the alternative addressing is in use.
- The device's tree depth, if the distributed addressing scheme is in use.

The method by which these data are made to persist is beyond the scope of this specification.[154]

[154]CCB Comment #183

Chapter 3 Security Services Specification

3.1 Document Organization

The remaining portions of this document specify in greater detail the various security services available within the ZigBee stack. Basic definitions and references are given in clause 3.2. A general description of the security services is given in sub-clause 3.2.1. In this clause, the overall security architecture is discussed; basic security services provided by each layer of this architecture are introduced. Clauses 3.2.2, 3.2.3, and 3.2.4 give the ZigBee Alliance's security specifications for the Medium Access Control (MAC) layer, the Network (NWK) layer, and the Application Support Sub-layer (APS) layer, respectively. These clauses introduce the security mechanisms, give the primitives, and define any frame formats used for security purposes. Clause 3.6 describes security elements common to the MAC, NWK, and APS layers. Clause 3.7 provides a basic functional description of the available security features. Finally, annexes provide technical details and test vectors needed to implement and test the cryptographic mechanisms and protocols used by the MAC, NWK, and APS layers.

3.2 General Description

Security services provided for ZigBee include methods for key establishment, key transport, frame protection, and device management. These services form the building blocks for implementing security policies within a ZigBee device. Specifications for the security services and a functional description of how these services shall be used are given in this document.

3.2.1 Security Architecture and Design

In this clause, the security architecture is described. Where applicable, this architecture complements and makes use of the security services that are already present in the 802.15.4 security specification.

3.2.1.1 Security Assumptions

The level of security provided by the ZigBee security architecture depends on the safekeeping of the symmetric keys, on the protection mechanisms employed, and on the proper implementation of the cryptographic mechanisms and associated security policies involved. Trust in the security architecture ultimately reduces to trust in the secure initialization and installation of keying material and to trust in the secure processing and storage of keying material. In the case of indirect addressing, it is assumed that the binding manager is trusted.

Implementations of security protocols, such as key establishment, are assumed to properly execute the complete protocol and do not leave out any steps hereof. Random number generators are assumed to operate as expected. Furthermore, it is assumed that secret keys do not become available outside the device in an unsecured way. That is, a device will not intentionally or inadvertently transmit its keying material to other devices, unless the keying material is protected, such as during key-transport. An exception to this assumption occurs when a device that has not been preconfigured joins the network. In this case, a single key may be sent unprotected, thus resulting in a brief moment of vulnerability.

The following caveat in these assumptions applies: due to the low-cost nature of ad hoc network devices, one cannot generally assume the availability of tamper-resistant hardware. Hence, physical access to a

device may yield access to secret keying material and other privileged information and access to the security software and hardware.

Due to cost constraints, ZigBee has to assume that different applications using the same radio are not logically separated (e.g., by using a firewall). In addition, from the perspective of a given device, it is even not possible (barring certification) to verify whether cryptographic separation between different applications on another device, or even between different layers of the communication stack hereof, is indeed properly implemented. Hence, one has to assume that separate applications using the same radio trust each other (i.e., there is no cryptographic task separation). In addition, lower layers (e.g., APS, NWK, or MAC) are fully accessible by any of the applications. These assumptions lead to an open trust model for a device: different layers of the communication stack and all applications running on a single device trust each other.

In summary: the provided security services cryptographically protect the interfaces between different devices only; separation of the interfaces between different stack layers on the same device is arranged non-cryptographically, via proper design of security service access points.

3.2.1.2 Security Design Choices

The open trust model (as described in sub-clause 3.2.1.1) on a device has far-reaching consequences. It allows re-use of the same keying material among different layers on the same device and it allows end-to-end security to be realized on a device-to-device basis rather than between pairs of particular layers (or even pairs of applications) on two communicating devices.

Another consideration is whether one is concerned with the ability of malevolent network devices to use the network to transport frames across the network without permission.

These observations lead to the following architectural design choices.

First, the principle that "*the layer that originates a frame is responsible for initially securing it*" is established. For example, if a MAC layer disassociate frame needs protection, MAC layer security shall be used. Likewise, if a NWK command frame needs protection, NWK layer security shall be used.

Second, if protection from theft of service is required (i.e., malevolent network devices), NWK layer security shall be used for all frames except those communicated between a router and a newly joined device (until the newly joined device received the Network key). Thus, only a device that has joined the network and successfully received the Network key will be able to have its messages communicated more than one hop across the network.

Third, due to the open trust model, security can be based on the reuse of keys by each layer. For example, the active Network key shall be used to secure APS layer broadcast frames, NWK layer frames, or MAC layer commands. Reuse of keys helps reduce storage costs.

Fourth, end-to-end security is enabled such as to make it possible that only source and destination devices have access to their shared key. This limits the trust requirement to those devices whose information is at stake. Additionally, this ensures that routing of messages between devices can be realized independent of trust considerations (thus, facilitating considerable separation of concern).

Fifth, to simplify interoperability of devices, the security level used by all devices in a given network and by all layers of a device shall be the same. In particular, the security level indicated in the PIB and NIB shall be the same. If an application needs more security than is provided by a given network, it shall form its own separate network with a higher security level.

There are several policy decisions which any real implementation must address correctly. Application profiles should include these policies:

— Handling error conditions arising from securing and unsecuring packets. Some error conditions may indicate loss of synchronization of security material, or may indicate ongoing attacks.

— Detecting and handling loss of counter synchronization and counter overflow.

— Detecting and handling loss of key synchronization.

— Expiration and periodic update of keys, if desired.

3.2.1.3 Security Keys

Security amongst a network of ZigBee devices is based on ‘link’ keys and a “network” key. Unicast communication between APL peer entities is secured by means of a 128-bit link key shared by two devices, while broadcast communications are secured by means of a 128-bit Network key shared amongst all devices in the network. The intended recipient is always aware of the exact security arrangement (i.e., the recipient knows whether a frame is protected with a link or a Network key).

A device shall acquire link keys either via key-transport, key-establishment, or pre-installation (e.g., factory installation). A device shall acquire a Network key via key-transport or pre-installation (e.g., factory installation). The key-establishment technique for acquiring a link key (see sub-clause 3.2.4.1) is based on a 'master' key. A device shall acquire a master key (for purposes of establishing corresponding link keys) via key-transport or pre-installation (e.g., factory installation). Ultimately, security between devices depends on the secure initialization and installation of these keys.

In a secured network there are a variety of security services available. Prudence dictates that one would like to avoid re-use of keys across different security services, which may cause security leaks due to unwanted interactions. As such, these different services use a key derived from a one-way function using the link key (as specified in sub-clause 3.6.3). The use of uncorrelated keys ensures the logical separation of executions of different security protocols. he key-load key is used to protect transported master keys; the key-transport key is used to protect other transported keys.

The Network key may be used by the MAC, NWK, and APL layers of ZigBee. As such, the same Network key and associated outgoing and incoming frame counters shall be available to all of these layers. The link and master keys may be used only by the APS sub-layer. As such, the link and master keys shall be available only to the APL layer.

3.2.1.4 ZigBee Security Architecture

ZigBee applications communicate using the IEEE 802.15.4 wireless standard [B1], which specifies two layers, the Physical (PHY) and Medium Access Control (MAC) layers. ZigBee builds on these layers a Network (NWK) layer and an Application (APL) layer. The PHY layer provides the basic communication capabilities of the physical radio. The MAC layer provides services to enable reliable, single-hop communication links between devices. The ZigBee NWK layer provides routing and multi-hop functions needed for creating different network topologies (e.g., star, tree, and mesh structures). The APL layer includes an Application Support (APS) sublayer, the ZigBee Device Object (ZDO), and applications. The ZDO is responsible for overall device management. The APS layer provides a foundation for servicing the ZDO and ZigBee applications.

The architecture includes security mechanisms at three layers of the protocol stack. The MAC, NWK, and APS layers are responsible for the secure transport of their respective frames. Furthermore, the APS sublayer provides services for the establishment, and maintenance of security relationships. The ZigBee Device Object (ZDO) manages the security policies and the security configuration of a device. Figure 1 shows a complete view of the ZigBee protocol stack. The security mechanisms provided by the APS and NWK layers are described in this version of the specification, as is the processing of secure MAC frames.

3.2.2 MAC Layer Security

When a frame originating at the MAC layer needs to be secured, ZigBee shall use MAC layer security as specified by the 802.15.4 specification [B1] and augmented by clause 3.3. A security corrigendum

proposal [B3] is being developed to augment the MAC layer specification and include the security elements needed by ZigBee. Specifically, at least one of ZigBee's security needs is the ability to protect incoming and outgoing frames using the security levels based on CCM* (see sub-clause 3.6.2.1, and Table 169 for a description of ZigBee security levels). CCM* is a minor modification of CCM specified in Clause 7 and Annex B of the 802.15.4 MAC layer specification [B1]. CCM* includes all of the features of CCM and additionally offers encryption-only and integrity-only capabilities. These extra capabilities simplify security by eliminating the need for CTR and CBC-MAC modes. Also, unlike other MAC layer security modes which require a different key for every security level, the use of CCM* enables the use of a single key for all CCM* security levels. With the use of CCM* throughout the ZigBee stack, the MAC, NWK, and APS layers can reuse the same key.

The MAC layer is responsible for its own security processing, but the upper layers shall determine which security level to use. For ZigBee, MAC layer frames requiring security processing shall be processed using the security material from the *macDefaultSecurityMaterial* or the *macACLEntryDescriptorSet* attributes of the MAC PIB. The upper layer (e.g., APL) shall set *macDefaultSecurityMaterial* to coincide with the active Network key and counters from the NWK layer and shall set *macACLEntryDescriptorSet* to coincide with any link keys from the APS layer that are shared with neighboring devices (e.g., a parent and child). The security suite shall be CCM* and the upper layers shall set the security level to coincide with the *nwkSecurityLevel* attribute in the NIB. For ZigBee, MAC layer link keys shall be preferred, but if not available, the default key (i.e., *macDefaultSecurityMaterial*) shall be used. Figure 64 shows an example of the security fields that may be included in an outgoing frame with security applied at the MAC level.

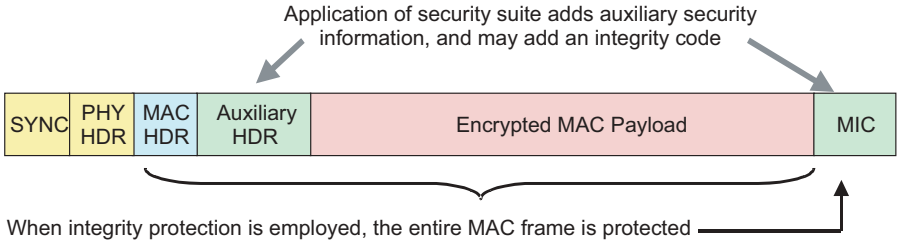


Figure 64 ZigBee frame with security at the MAC level

3.2.3 NWK Layer Security

When a frame originating at the NWK layer needs to be secured or when a frame originates at a higher layer and the *nwkSecureAllFrames* attribute in the NIB is TRUE, ZigBee shall use the frame protection mechanism specified in sub-clause 3.4.1 of this specification, unless the SecurityEnable parameter of the NLDE-DATA.request primitive is FALSE, explicitly prohibiting security[155]. Like the MAC layer, the NWK layer's frame protection mechanism shall make use of the Advanced Encryption Standard (AES) [B8] and use CCM* as specified in Annex A. The security level applied to a NWK frame shall be given by the *nwkSecurityLevel* attribute in the NIB. Upper layers manage NWK layer security by setting up active and alternate Network keys and by determining which security level to use.

One responsibility of the NWK layer is to route messages over multi-hop links. As part of this responsibility, the NWK layer will broadcast route request messages and process received route reply messages. Route request messages are simultaneously broadcast to nearby devices and route reply messages originate from nearby devices. If the appropriate link key is available, the NWK layer shall use the link key to secure outgoing NWK frames. If the appropriate link key is not available, in order to secure messages against outsiders the NWK layer shall use its active Network key to secure outgoing NWK frames and either its active or an alternate Network key to secure incoming NWK frames. In this scenario, the frame format

[155] CCB Comment #148

explicitly indicates the key used to protect the frame, thus intended recipients can deduce which key to use for processing an incoming frame and also determine if the message is readable by all network devices, rather than just by itself.

Figure 65 shows an example of the security fields that may be included in a NWK frame.

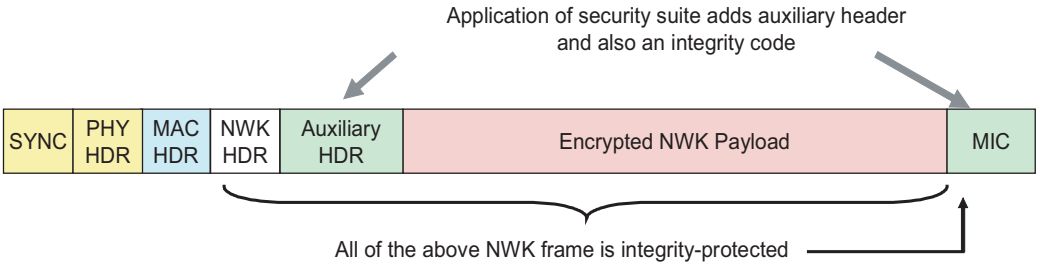


Figure 65 ZigBee frame with security on the NWK level

3.2.4 APL Layer Security

When a frame originating at the APL layer needs to be secured, the APS sublayer shall handle security. The APS layer's frame protection mechanism is specified in sub-clause 3.5.1 of this specification. The APS layer allows frame security to be based on link keys or the Network key. Figure 66 shows an example of the security fields that may be included in an APL frame. Another security responsibility of the APS layer is to provide applications and the ZDO with key establishment, key transport, and device management services.

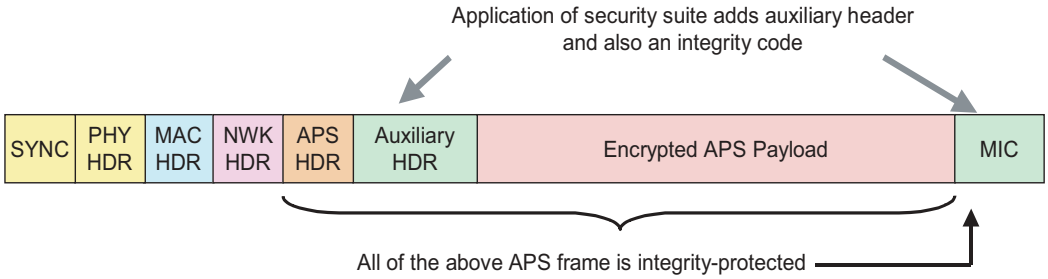


Figure 66 ZigBee frame with security on the APS level

3.2.4.1 Key Establishment

The APS sublayer's key establishment services provide the mechanism by which a ZigBee device may derive a shared secret key, the so-called link key (see sub-clause 3.2.1.3) with another ZigBee device. Key establishment involves two entities, an initiator device and a responder device, and is prefaced by a trust-provisioning step. Trust information (e.g., a master key) provides a starting point for establishing a link key and can be provisioned in-band or out-band. Once trust information is provisioned, a key-establishment protocol involves three conceptual steps: the exchange of ephemeral data, the use of this ephemeral data to derive the link key, and the confirmation that this link key was correctly computed.

In the Symmetric-Key Key Establishment (SKKE) protocol, an initiator device establishes a link key with a responder device using a master key. This master key, for example, may be pre-installed during manufacturing, may be installed by a trust center (e.g., from the initiator, the responder, or a third party device acting as a trust center), or may be based on user-entered data (e.g., PIN, password, or key). The secrecy and authenticity of the master key needs to be upheld in order to maintain a trust foundation.

3.2.4.2 Transport Key

The transport-key service provides secured and unsecured means to transport a key to another device or other devices. The secured transport-key command provides a means to transport a master, link, Network key from a key source (e.g., trust center) to other devices. The unsecured transport-key command provides a means for loading a device with an initial key. This command does not cryptographically protect the key being loaded. In this case, the security of the transported key can be realized by non-cryptographic means, e.g., by communicating the command via an out-of-band channel.

3.2.4.3 Update Device

The update-device service provides a secure means for a device (e.g., a router) to inform a second device (e.g., a trust center) that a third device has had a change of status that must be updated (e.g., the device joined or left the network). This is the mechanism by which the trust center maintains an accurate list of active network devices.

3.2.4.4 Remove Device

The remove device service provides a secure means by which a device (e.g., a trust center) may inform another device (e.g., a router) that one of its children should be removed from the network. This may be employed, for example, to remove a device from the network that has not satisfied the trust center's security requirements for network devices.

3.2.4.5 Request Key

The request-key service provides a secure means for a device to request the current Network key, or an end-to-end application master key, from another device (e.g., its trust center).

3.2.4.6 Switch Key

The switch-key service provides a secure means for a device (e.g., a trust center) to inform another device that it should switch to a different active Network key.

3.2.5 Trust Center Role

For security purposes, ZigBee defines the role of trust center. The trust center is the device trusted by devices within a network to distribute keys for the purpose of network and end-to-end application configuration management. All members of the network shall recognize exactly one trust center, and there shall be exactly one trust center in each secure network.

In high-security, commercial applications (see sub-clause 3.7.2.1) a device can be preloaded with the trust center address and initial master key (e.g., via an unspecified mechanism). Alternatively, if the application can tolerate a moment of vulnerability, the master key can be sent via an in-band unsecured key transport. If not preloaded, a device's trust center defaults to the PAN coordinator or a device designated by the PAN coordinator.

In low-security, residential applications (see sub-clause 3.7.2.2) a device securely communicates with its trust center using the Network key, which can be preconfigured or sent via an in-band unsecured key transport.

The functions performed by the trust center can be subdivided into three sub-roles: trust manager, network manager, and configuration manager. A device trusts its trust manager to identify the device(s) that take on the role of its network and configuration manager. A network manager is responsible for the network and distributes and maintains the Network key to devices it manages. A configuration manager is responsible for binding two applications and enabling end-to-end security between devices it manages (e.g., by distributing master keys or link keys). To simplify trust management, these three sub-roles are contained within a single device – the trust center.

For purposes of trust management, a device shall accept an initial master or Network key originating from its trust center via unsecured key transport. For purposes of network management, a device shall accept an initial Network key and updated Network keys only from its trust center (i.e., network manager). For purpose of configuration, a device shall accept master keys or link keys for the purpose of establishing end-to-end security between two devices only from its trust center (i.e., configuration manager). Aside from the initial master key, additional link, master, and Network keys shall only be accepted if they originate from a device's trust center via secured key transport.

3.3 MAC Layer Security

The MAC layer is responsible for the processing steps needed to securely transmit outgoing MAC frames and securely receive incoming MAC frames. Upper layers control the security processing operations, by setting up the appropriate keys and frame counters and establishing which security level to use.

3.3.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming MAC frames are described in sub-clause 3.3.1.1 and sub-clause 3.3.1.2, respectively.

3.3.1.1 Security Processing of Outgoing Frames

If the MAC layer has a frame, consisting of a header *MacHeader* and payload *Payload*, that needs security protection it shall apply security as follows:

1. Obtain the security material (as specified in sub-clause 3.3.2), including the key, outgoing frame counter *FrameCount*, key sequence count *SeqCount*, and security level identifier (as specified in Table 169) from the MAC PIB using the following procedure. If the outgoing frame counter has as its value the 4-octet representation of the integer 2^{32}-1 or any of this security material cannot be determined, then security processing shall fail and no further security processing shall be done on this frame.
   a) First, an attempt shall be made to retrieve the security material and security level identifier associated with the destination address of the outgoing frame from the *macACLEntryDescriptorSet* attribute in the MAC PIB.
   b) If the first attempt fails, then security material shall be obtained by using the *macDefaultSecurityMaterial* attribute from the MAC PIB and the security level identifier shall be obtained from the *MacDefaultSecuritySuite* attribute from the MAC PIB.
2. The Security Control Field *SecField* is the 1-octet field formatted as in sub-clause 3.6.1.1, with the following settings:
   a) The security level subfield shall be set to the security level obtained in Step 1 above;
   b) The key identifier subfield shall be set to the 2-bit field '00';
   c) The extended nonce subfield shall be set to the 1-bit field '0';
   d) The reserved bits shall be set to the 2-bit field '00'.[156]
3. Execute the CCM* mode encryption and authentication operation, as specified in Annex A, with the following instantiations:
   a) The parameter *M* shall be obtained from Table 169 corresponding to the security level from step 1;
   b) The bit string *Key* shall be the key obtained from step 1;
   c) The nonce *N* shall be the 13-octet string constructed using the local device's 64-bit extended address, *SecField* from Step 1[157], and *FrameCount* from step 1 (see Figure 72 from [B1]);
   d) If the security level requires encryption, the octet string *a* shall be the string *MacHeader* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *MacHeader* || *Payload* and the octet string *m* shall be a string of length zero. Note that ZigBee interprets [B1] to mean that frame counters are authenticated.[158]

[156] CCB Comment #195
[157] Ibid
[158] CCB Comment #180

4. If the CCM* mode invoked in step 3[159] outputs 'invalid', security processing shall fail and no further security processing shall be done on this frame.

5. Let *c* be the output from step 4[160] above. If the security level requires encryption, the secured outgoing frame shall be *MacHeader* || *FrameCount* || *SeqCount* || *c*, otherwise the secured outgoing frame shall be *MacHeader* || *FrameCount* || *SeqCount* || *Payload* || *c*.

6. If the secured outgoing frame size is greater than *aMaxPHYPacketSize* (from [B1]), security processing shall fail and no further security processing shall be done on this frame.

7. The outgoing frame counter from step 1 shall be incremented by one and stored in the location from which the security material was obtained in step 1 (i.e., either the *macDefaultSecurityMaterial* attribute or the *MacDefaultSecuritySuite* attribute).

3.3.1.2 Security Processing of Incoming Frames

If the MAC layer receives a secured frame (consisting of a header *MacHeader*, frame count *ReceivedFrameCount*, sequence count *ReceivedSeqCount*, and payload *SecuredPayload*) it shall perform security processing as follows:

1. If *ReceivedFrameCount* has as value the 4-octet representation of the integer 2^{32}-1, security processing shall fail and no further security processing shall be done on this frame.

2. Obtain the security material (as specified in sub-clause 3.3.2), including the key, optional external frame counter *FrameCount*, optional key sequence count *SeqCount*, and security level identifier (as specified in Table 169) from the MAC PIB using the following procedure. If the security material cannot be obtained or if *SeqCount* exists and does not match *ReceivedSeqCount*, security processing shall fail and no further security processing shall be done on this frame.

   a) First, an attempt shall be made to retrieve the security material and security level identifier associated with the source address of the incoming frame from the *macACLEntryDescriptorSet* attribute in the MAC PIB.

   b) If the first attempt fails, then security material shall be obtained by using the *macDefaultSecurityMaterial* attribute from the MAC PIB and the security level identifier shall be obtained from the *MacDefaultSecuritySuite* attribute from the MAC PIB.

3. If *FrameCount* exists and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall fail and no further security processing shall be done on this frame.

4. The Security Control Field *SecField* is the 1-octet field formatted as in Clause 7.1.1, Figure 18, with the following settings:

   a) The security level subfield shall be set to the security level from the MACPIB (as specified in Table 29);

   b) The key identifier subfield shall be set to the 2-bit field '00';

   c) The extended nonce subfield shall be set to the 1-bit field '0';

   d) The reserved bits shall be set to the 2-bit field '00'.[161]

5. Execute the CCM* mode decryption and authentication checking operation, as specified in Annex A.3, with the following instantiations:

   a) The parameter *M* shall be obtained from Table 169 corresponding to the security level from step 1;

   b) The bit string *Key* shall be the key obtained from step 2;

[159]CCB Comment #195
[160]Ibid
[161]Ibid

c) The nonce *N* shall be the 13-octet string constructed using the 64-bit extended sender address, *SecField* from Step 4[162], and *ReceivedFrameCount* from step 1 (see Figure 72 from [B1]);The nonce N shall be formatted according to the endianness convention used in this specification (the octet containing the lowest numbered bits first to the octet containing the higher numbered bits).[163]

d) Parse the octet string *SecuredPayload* as $Payload_1 \| Payload_2$, where the right-most string $Payload_2$ is an *M*-octet string. If this operation fails, security processing shall fail and no further security processing shall be done on this frame;

e) If the security level requires encryption, the octet string *a* shall be the string *MacHeader* || *ReceivedFrameCount* || *ReceivedSeqCount* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *MacHeader* || *ReceivedFrameCount* || *ReceivedSeqCount* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.

6. Return the results of the CCM* operation:

a) If the CCM* mode invoked in step 5[164] outputs 'invalid', security processing shall fail and no further security processing shall be done on this frame;

b) Let *m* be the output of step 5[165] above. If the security level requires encryption, set the octet string *UnsecuredMacFrame* to the string *a* || *m*. Otherwise, set the octet string *UnsecuredMacFrame* to the string *a*;

7. If the optional *FrameCount* (obtained in step 2) exists, set it to *ReceivedFrameCount* and update MAC PIB. *UnsecuredMacFrame* now represents the unsecured received MAC layer frame.

3.3.2 Security-Related MAC PIB Attributes

The security-related MAC PIB attributes shall be those as defined in Table 72 of [B1]. The security material used for CCM* mode shall the same as given for CCM mode in Figure 70 of [B1].

For the *macDefaultSecurityMaterial* attribute from the MAC PIB, the upper layer shall set the symmetric key, outgoing frame counter, and optional external key sequence counter equal to the corresponding elements of the network security material descriptor in the *nwkSecurityMaterialSet* of the NIB referenced by the *nwkActiveKeySeqNumber* attribute of the NIB. The optional external frame counter shall not be used and the optional external key sequence counter shall correspond to the sequence number of the Network key.

For the *macACLEntryDescriptorSet* attribute from the MAC PIB, the upper layer shall set the symmetric key, and outgoing frame counter equal to the corresponding elements of the Network key-pair descriptor in the *apsDeviceKeyPairSet* of the AIB. The optional external frame counter shall be set to the incoming frame counter, The key sequence counter shall be set to 0x00, and the optional external key sequence counter shall not be used.

3.4 NWK Layer Security

The NWK layer is responsible for the processing steps needed to securely transmit outgoing frames and securely receive incoming frames. Upper layers control the security processing operations, by setting up the appropriate keys and frame counters and establishing which security level to use. The formatting of all frames and fields in this specification are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the leftmost bit is transmitted first in time. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of

[162]CCB Comment #195
[163]CCB Comment #98
[164]CCB Comment #195
[165]Ibid

the field is k bits. Fields that are longer than a single octet are sent to the next layer in the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.[166]

3.4.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming NWK frames are described in sub-clause 3.4.1.1 and sub-clause 3.4.1.2, respectively.

3.4.1.1 Security Processing of Outgoing Frames

If the NWK layer has a frame, consisting of a header *NwkHeader* and payload *Payload*, that needs security protection and *nwkSecurityLevel* > 0, it shall apply security as follows:

1. Obtain the *nwkActiveKeySeqNumbe*r from the NIB and use it to retrieve the active Network key *key*, outgoing frame counter *OutgoingFrameCounter*, and key sequence number *KeySeqNumber* from the *nwkSecurityMaterialSet* attribute in the NIB. Obtain the security level from the *nwkSecurityLevel* attribute from the NIB. If the outgoing frame counter has as its value the 4-octet representation of the integer 2^{32}-1, or if the key cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.
2. Construct auxiliary header *AuxiliaryHeader* (see sub-clause 3.6.1):
   a) The security control field shall be set as follows:
      1) The security level sub-field shall be the security level obtained from step 1.
      2) The key identifier sub-field shall be set to ‘01’ (i.e., the Network key).
      3) The extended nonce sub-field shall be set to 1.
   b) The source address field shall be set to the 64-bit extended address of the local device.
   c) The frame counter field shall be set to the outgoing frame counter from step 1.
   d) The key sequence number field shall be set to the sequence number from step 1.
3. Execute the CCM* mode encryption and authentication operation, as specified in Annex A, with the following instantiations:
   a) The parameter *M* shall be obtained from Table 169 corresponding to the security level from step 1;
   b) The bit string *Key* shall be the key obtained from step 1;
   c) The nonce *N* shall be the 13-octet string constructed using the security control field from step 2a, the frame counter field from step 2c, and the source address field from step 2b (see sub-clause 3.6.2.2);
   d) If the security level requires encryption, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* || *Payload* and the octet string *m* shall be a string of length zero.
4. If the CCM* mode invoked in step 3 outputs ‘invalid’, security processing shall fail and no further security processing shall be done on this frame.
5. Let *c* be the output from step 3 above. If the security level requires encryption, the secured outgoing frame shall be *NwkHeader* || *AuxiliaryHeader* || *c*, otherwise the secured outgoing frame shall be *NwkHeader* || *AuxiliaryHeader* || *Payload* || *c*.
6. If the secured outgoing frame size is greater than *aMaxMACFrameSize* (see [B1]), security processing shall fail and no further security processing shall be done on this frame.
7. The outgoing frame counter from step 1 shall be incremented by one and stored in the *OutgoingFrameCounter* element of the network security material descriptor referenced by the

[166] CCB Comment #98

nwkActiveKeySeqNumber in the NIB (i.e., the outgoing frame counter value associated with the key used to protect the frame is updated).

8. Over-write the security level subfield of the security control field by the 3-bit all-zero string '000'.[167]

3.4.1.2 Security Processing of Incoming Frames

If the NWK layer receives a secured frame (consisting of a header *NwkHeader*, auxiliary header *AuxiliaryHeader*, and payload *SecuredPayload*) as indicated by the security sub-field of the NWK header frame control field it shall perform security processing as follows:

1. Determine the security level from the *nwkSecurityLevel* attribute from the NIB. Over-write the 3-bit security level subfield of the security control field of the AuxillaryHeader with this value. Determine the sequence number SequenceNumber, sender address SenderAddress, and received frame count ReceivedFrameCount from the auxiliary header AuxiliaryHeader (see sub-clause 3.6.1). If ReceivedFrameCounter has as value the 4-octet representation of the integer 232-1, security processing shall fail and no further security processing shall be done on this frame.[168]
2. Obtain the appropriate security material (consisting of the key and other attributes) by matching *SequenceNumber* to the sequence number of any key in the *nwkSecurityMaterialSet* attribute in the NIB. If the security material cannot be obtained, security processing shall fail and no further security processing shall be done on this frame. If the sequence number of the received frame belongs to a newer entry in the *nwkSecurityMaterialSet*, and the source address of the packet is the trust center, then the *nwkActiveKeySeqNumber* shall be set to received sequence number.[169]
3. If there is an incoming frame count *FrameCount* corresponding to *SenderAddress* from the security material obtained in step 2 and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall fail and no further security processing shall be done on this frame.
4. Execute the CCM* mode decryption and authentication checking operation, as specified in Annex A.3, with the following instantiations:
   a) The parameter *M* shall obtained from Table 169 corresponding to the security level from step 1;
   b) The bit string *Key* shall be the key obtained from step 2;
   c) The nonce *N* shall be the 13-octet string constructed using the security control, the frame counter, and the source address fields from *AuxiliaryHeader* (see sub-clause 3.6.2.2). Note that the security level subfield of the security control field has been overwritten in step 1 and now contains the value determined from the *nwkSecurityLevel* attribute from the NIB.[170]
   d) Parse the octet string *SecuredPayload* as $Payload_1 \| Payload_2$, where the right-most string $Payload_2$ is an *M*-octet string. If this operation fails, security processing shall fail and no further security processing shall be done on this frame;
   e) If the security level requires encryption, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.
5. Return the results of the CCM* operation:
   a) If the CCM* mode invoked in step 4 outputs ‘invalid’, security processing shall fail and no further security processing shall be done on this frame;

[167] CCB Comment #245
[168] Ibid
[169] CCB Comment #144
[170] CCB Comment #245

b) Let *m* be the output of step 4 above. If the security level requires encryption, set the octet string *UnsecuredNwkFrame* to the string *a* || *m*. Otherwise, set the octet string *UnsecuredNwkFrame* to the string *a*;

6. Set *FrameCount* to (*ReceivedFrameCount* + 1)[171] and store both *FrameCount* and *SenderAddress* in the NIB. *UnsecuredNwkFrame* now represents the unsecured received network frame and security processing shall succeed. So as to never cause the storage of the frame count and address information to exceed the available memory, the memory allocated for incoming frame counters needed for NWK layer security shall be bounded by *M***N*, where *M* and *N* represent the cardinality of *nwkSecurityMaterialSet* and *nwkNeighborTable* in the NIB, respectively.

3.4.2 Secured NPDU Frame

The NWK layer frame format from [B3] consists of a NWK header and NWK payload field. The NWK header consists of frame control and routing fields. When security is applied to an NPDU frame, the security bit in the NWK frame control field shall be set to 1 to indicate the presence of the auxiliary frame header. The format for the auxiliary frame header is given in sub-clause 3.6.1. The format of a secured NWK layer frame is shown in Figure 67. The auxiliary frame header is situated between the NWK header and payload fields.

Figure 67 Secured NWK layer frame format

<table>
<tr><th>Octets: Variable</th><th>14</th><th colspan="2">Variable</th></tr>
<tr><td rowspan="2">Original NWK Header ([B3], Clause 7.1)</td><td rowspan="2">Auxiliary frame header</td><td>Encrypted Payload</td><td>Encrypted Message Integrity Code (MIC)</td></tr>
<tr><td colspan="2">Secure frame payload = Output of CCM*</td></tr>
<tr><td colspan="2">Full NWK header</td><td colspan="2">Secured NWK payload</td></tr>
</table>

3.4.3 Security-Related NIB Attributes

The NWK PIB contains attributes that are required to manage security for the NWK layer. Each of these attributes can be read and written using the NLME-GET.request and NLME-SET.request primitives, respectively. The security-related attributes contained in the NWK PIB are presented in Table 140 through Table 142.

Table 140 NIB security attributes

Attribute	Identifier	Type	Range	Description	Default
nwkSecurityLevel	0xa0[a]	Octet	0x00-07	The security level for outgoing and incoming NWK frames. The allowable security level identifiers are presented in Table 169.	0x06
nwkSecurity-MaterialSet	0xa1[b]	A set of 0, 1, or 2 network security material descriptors. See Table 141.	Variable	Set of network security material descriptors capable of maintaining an active and alternate Network key.	-

[171] CCB Comment #160

Table 140 NIB security attributes

nwkActiveKey-SeqNumber	0xa2[c]	Octet	0x00-0xFF	The sequence number of the active Network key in *nwkSecurityMaterialSet*.	0x00
nwkAllFresh	0xa3[d]	Boolean	TRUE | FALSE	Indicates whether incoming NWK frames must be all checked for freshness when the memory for incoming frame counts is exceeded.	TRUE
nwkSecureAll-Frames[e]	0xa5	Boolean	TRUE | FALSE	This indicates if security shall be applied to incoming and outgoing NWK frames. If set to 0x01 security processing shall be applied to all incoming and outgoing frames except data frames destined for the current device that have the security sub-field of the frame control field set to 0. If this attribute has a value of 0x01 the NWK layer shall not relay frames that have the security sub-field of the frame control field set to 0. The SecurityEnable parameter of the NLDE-DATA.request primitive shall override the setting of this attribute.	TRUE

[a]CCB Comment #151
[b]Ibid
[c]Ibid
[d]Ibid
[e]Ibid

Table 141 Elements of the network security material descriptor

Name	Type	Range	Description	Default
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys for purposes of key updates, and incoming frame security operations.	00
OutgoingFrame-Counter	Ordered set of 4 octets	0x00000000-0xFFFFFFFF	Outgoing frame counter used for outgoing frames.	0x00000000
IncomingFrame-CounterSet	Set of incoming frame counter descriptor values. See Table 142.	Variable	Set of incoming frame counter values and corresponding device addresses.	Null set
Key	Ordered set of 16 octets	-	The actual value of the key.	-

Table 142 Elements of the incoming frame counter descriptor

Name	Type	Range	Description	Default
SenderAddress	Device address	Any valid 64-bit address	Extended device address.	Device specific
IncomingFrame-Counter	Ordered set of 4 octets	0x00000000-0xFFFFFFFF	Incoming frame counter used for incoming frames.	0x00000000

3.5 APS Layer Security

The APS layer is responsible for the processing steps needed to securely transmit outgoing frames, securely receive incoming frames, and securely establish and manage cryptographic keys. Upper layers control the management of cryptographic keys by issuing primitives to the APS layer. Table 143 lists the primitives available for key management and maintenance. Upper layers also determine which security level to use when protecting outgoing frames. The formatting of all frames and fields in this specification are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the leftmost bit is transmitted first in time. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are sent to the next layer in the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.[172]

Table 143 The APS layer security primitives

APSME Security Primitives	Request	Confirm	Indication	Response	Description
APSME-ESTABLISH-KEY	3.5.2.1	3.5.2.2	3.5.2.3	3.5.2.4	Establish link key with another ZigBee device using the SKKE method.
APSME-TRANS-PORT-KEY	3.5.3.1	-	3.5.3.2	-	Transport security material from one device to another.
APSME-UPDATE-DEVICE	3.5.4.1	-	3.5.4.2	-	Notifies the trust center when a new device joined or an existing device left the network.
APSME-REMOVE-DEVICE	3.5.5.1	-	3.5.5.2	-	Used by the trust center to notify a router that one of the router's child devices should be removed from the network.
APSME-REQUEST-KEY	3.5.6.1	-	3.5.6.2	-	Used by a device to request that the trust center send an application master key or current Network key.
APSME-SWITCH-KEY	3.5.7.1	-	3.5.7.2	-	Used by the trust center to tell a device to switch to a new Network key.

[172] CCB Comment #98

3.5.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming APS frames are described in sub-clause 3.5.1.1 and sub-clause 3.5.1.2, respectively.

3.5.1.1 Security Processing of Outgoing Frames

If the APS layer has a frame, consisting of a header *ApsHeader* and payload *Payload*, that needs security protection and *nwkSecurityLevel* > 0, it shall apply security as follows:

1. Obtain the security material and key identifier *KeyIdentifier* using the following procedure. If security material or key identifier cannot be determined, then security processing shall fail and no further security processing shall be done on this frame.
   a) If the frame is a result of a APSDE-DATA.request primitive:
      i) If the *useNwkKeyFlag* parameter is TRUE, then security material shall be obtained by using the *nwkActiveKeySeqNumbe*r from the NIB to retrieve the active Network key, outgoing frame counter, and sequence number from the *nwkSecurityMaterialSet* attribute in the NIB. *KeyIdentifier* shall be set to '01' (i.e., the Network key).
      ii) Otherwise, the security material associated with the destination address of the outgoing frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB. *KeyIdentifier* shall be set to '00' (i.e., a data key). Note, if the frame is being transmitted using indirect addressing, the destination address shall be the address of the binding manager.
   b) If the frame is a result of an APS command:
      i) First, an attempt shall be made to retrieve the security material associated with the destination address of the outgoing frame from the *apsDeviceKeyPairSet* attribute in the AIB. For all cases, except transport-key commands, *KeyIdentifier* shall be set to '00'(i.e., a data key). For the case of transport-key commands, *KeyIdentifier* shall be set to '02' (i.e., the key-transport key) when transporting a Network key and shall be set to '03' (i.e., the key-load key) when transporting an application link key, application master key, or trust center master key. See sub-clause 3.6.3 for a description of the key-transport and key-load keys.
      ii) If the first attempt fails, then security material shall be obtained by using the *nwkActiveKeySeqNumbe*r from the NIB to retrieve the active Network key, outgoing frame counter, and sequence number from the *nwkSecurityMaterialSet* attribute in the NIB. *KeyIdentifier* shall be set to '01' (i.e., the Network key).
2. If the key identifier is equal to 01 (i.e. network key), the APS layer shall first verify that the NWK layer is not also applying security. If the NWK layer is applying security, then the APS layer shall not apply any security. The APS layer can determine that the NWK layer is applying security by verifying that the value of the *nwkSecureAllFrames* attribute of the NIB has a value of TRUE and the *nwkSecurityLevel* NIB attribute has a non-zero value.[173]
3. Extract the outgoing frame counter (and, if *KeyIdentifier* is 01, the key sequence number) from the security material obtained from step 1. If the outgoing frame counter has as its value the 4-octet representation of the integer 2^{32}-1, or if the key cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.
4. Obtain the security level from the *nwkSecurityLevel* attribute from the NIB. If the frame is a result of an APS command, the security level shall be forced to 7 (ENC-MIC-128).
5. Construct auxiliary header *AuxiliaryHeader* (see sub-clause 3.6.1):
   a) The security control field shall be set as follows:
      i) The security level sub-field shall be the security level obtained from step 3.

[173] CCB Comment #145

ii) The key identifier sub-field shall be set to *KeyIdentifier*.
iii) The extended nonce sub-field shall be set to 0.

b) The frame counter field shall be set to the outgoing frame counter from step 2.

c) If *KeyIdentifier* is 1, the key sequence number field shall be present and set to the key sequence number from step 2. Otherwise, the key sequence number field shall not be present.

6. Execute the CCM* mode encryption and authentication operation, as specified in Annex A.2, with the following instantiations:

a) The parameter *M* shall obtained from Table 169 corresponding to the security level from step 3;

b) The bit string *Key* shall be the key obtained from step 1;

c) The nonce *N* shall be the 13-octet string constructed using the security control and frame counter fields from step 4 and the 64-bit extended address of the local device (see sub-clause 3.6.2.2);

d) If the security level requires encryption, the octet string *a* shall be the string *ApsHeader* || *Auxiliary-Header* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* || *Payload* and the octet string *m* shall be a string of length zero.

7. If the CCM* mode invoked in step 3 outputs ‘invalid’, security processing shall fail and no further security processing shall be done on this frame.

8. Let *c* be the output from step 3 above. If the security level requires encryption, the secured outgoing frame shall be *ApsHeader* || *AuxiliaryHeader* || *c*, otherwise the secured outgoing frame shall be *ApsHeader* || *AuxiliaryHeader* || *Payload* || *c*.

9. If the secured outgoing frame size will result in the MSDU being greater than *aMaxMACFrameSize* octets[174] (see [B1]), security processing shall fail and no further security processing shall be done on this frame.

10. The outgoing frame counter from step 1 shall be incremented and stored in the appropriate location(s) of the NIB, AIB, and MAC PIB corresponding to the key that was used to protect the outgoing frame.

11. Over-write the security level subfield of the security control field by the 3-bit all-zero string '000'.[175]

3.5.1.2 Security Processing of Incoming Frames

If the APS layer receives a secured frame (consisting of a header *ApsHeader*, auxiliary header *AuxiliaryHeader*, and payload *SecuredPayload*) as indicated by the security sub-field of the APS header frame control field it shall perform security processing as follows:

1. Determine the sequence number *SequenceNumber*, key identifier *KeyIdentifier*, and received frame counter value *ReceivedFrameCounter* from the auxiliary header *AuxiliaryHeader*. If *ReceivedFrameCounter* is the 4-octet representation of the integer 232-1, security processing shall fail and no further security processing shall be done on this frame.[176]

2. Determine the source address *SourceAddress* from the address-map table in the AIB, using the source address in the APS frame as the index. If the source address is incomplete or unavailable, security processing shall fail and no further security processing shall be done on this frame. If the delivery-mode sub-field of the frame control field of *ApsHeader* has a value of 1 (i.e., indirect addressing), the source address shall be the address of the binding manager, as described in the APS specification [B7].

3. Obtain the appropriate security material in the following manner. If the security material cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.

[174] CCB Comment #366
[175] CCB Comment #245
[176] Ibid

a) If *KeyIdentifier* is ‘00’ (i.e., data key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB.

b) If *KeyIdentifier* is ‘01’ (i.e., Network key), the security material shall be obtained by matching *SequenceNumber* to the sequence number to the sequence number of any key in the *nwkSecurityMaterialSet* attribute in the NIB. If the sequence number of the received frame belongs to a newer entry in the *nwkSecurityMaterialSet*, then the *nwkActiveKeySeqNumber* may be set to the received sequence number. If the security material associated with the *SourceAddress* of the incoming frame can be obtained from the attribute in the AIB, then security processing shall fail and no further security processing shall be done on this frame.[177]

c) If *KeyIdentifier* is ‘02’ (i.e., key-transport key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB and the key for this operation shall be derived from the security material as specified in sub-clause 3.6.3 for the key-transport key.

d) If *KeyIdentifier* is ‘03’ (i.e., key-load key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB and the key for this operation shall be derived from the security material as specified in sub-clause 3.6.3 for the key-load key.

4. If there is an incoming frame count *FrameCount* corresponding to *SourceAddress* from the security material obtained in step 3 and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall fail and no further security processing shall be done on this frame.

5. Determine the security level *SecLevel* as follows. If the frame type subfield of the frame control field of *ApsHeader* indicates an APS data frame, then *SecLevel* shall be set to the *nwkSecurityLevel* attribute in the NIB. Otherwise *SecLevel* shall be set to 7 (ENC-MIC-128). Overwrite the security level subfield of the security control field in the AuxillaryHeader with the value of *SecLevel*.[178]

6. Execute the CCM* mode decryption and authentication checking operation, as specified in Annex A.3, with the following instantiations:

   a) The parameter *M* shall obtained from Table 169 corresponding to the security level from step 5;

   b) The bit string *Key* shall be the key obtained from step 3;

   c) The nonce *N* shall be the 13-octet string constructed using the security control and frame counter fields from *AuxiliaryHeader*, and *SourceAddress* from step 2 (see sub-clause 3.6.2.2);

   d) Parse the octet string *SecuredPayload* as $Payload_1 \,||\, Payload_2$, where the right-most string $Payload_2$ is an *M*-octet string. If this operation fails, security processing shall fail and no further security processing shall be done on this frame;

   e) If the security level requires encryption, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.

7. Return the results of the CCM* operation:

   a) If the CCM* mode invoked in step 4 outputs ‘invalid’, security processing shall fail and no further security processing shall be done on this frame;

   b) Let *m* be the output of step 4 above. If the security level requires encryption, set the octet string *UnsecuredApsFrame* to the string *a* || *m*. Otherwise, set the octet string *UnsecuredApsFrame* to the string *a*;

[177] CCB Comment #146
[178] CCB Comment #245

8. Set *FrameCount* to (*ReceivedFrameCount* + 1)[179] and store both *FrameCount* and *SourceAddress* in the appropriate security material as obtained in step 3. If storing this frame count and address information will cause the memory allocation for this type of information to be exceeded and the *nwkAllFresh* attribute in the NIB is TRUE, then security processing shall fail and no further security processing shall be done on this frame; otherwise security processing shall succeed.

3.5.2 Key-Establishment Services

The APSME provides services that allow two devices to mutually establish a link key. Initial trust information (e.g., a master key) must be installed in each device prior to running the key establishment protocol (see sub-clause 3.5.3 for mechanisms to provision initial trust information).

3.5.2.1 APSME-ESTABLISH-KEY.request

The APSME-ESTABLISH-KEY.request primitive is used for initiating a key-establishment protocol. This primitive can be used when there is a need to securely communicate with another device. One device will act as an initiator device and another device will act as the responder. The initiator shall start the key-establishment protocol by issuing the APSME-ESTABLISH-KEY.request with parameters indicating the address of the responder device and which key-establishment protocol to use (i.e., SKKE direct or indirect).

3.5.2.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-ESTABLISH-KEY.request	{
	ResponderAddress,
	UseParent,
	ResponderParentAddress,
	KeyEstablishmentMethod
	}

Table 144 specifies the parameters for the APSME-ESTABLISH-KEY.request primitive.

Table 144 APSME-ESTABLISH-KEY.request parameters

Parameter Name	Type	Valid Range	Description
Responder-Address	Device Address	Any valid 64-bit address	The extended 64-bit address of the responder device.

[179] CCB Comment #160

Table 144 APSME-ESTABLISH-KEY.request parameters

UseParent	Boolean	TRUE | FALSE	This parameter indicates if the responder's parent shall be used to forward messages between the initiator and responder devices: TRUE: Use parent. FALSE: Do not use parent.
Responder-ParentAddress	Device Address	Any valid 64-bit address	If the *UseParent* is TRUE, then *Responder-ParentAddress* parameter shall contain the extended 64-bit address of the responder's parent device. Otherwise, this parameter is not used and need not be set.
KeyEstablishment-Method	Integer	0x00 - 0x03	The requested key-establishment method shall be one of the following: 0x00 = SKKE method. 0x01-0x03: reserved.

3.5.2.1.2 When generated

A higher layer on an initiator device shall generate this primitive when it requires a link key to be established with a responder device. If the initiator device wishes to use the responder's parent as a liaison (for NWK security purposes), it shall set the *UseParent* parameter to TRUE and shall set the *ResponderParentAddress* parameter to the 64-bit extended address of the responder's parent.

3.5.2.1.3 Effect on receipt

The receipt of an APSME-ESTABLISH-KEY.request primitive, with the *KeyEstablishmentMethod* parameter equal to SKKE, shall cause the APSME to execute the SKKE protocol, described in sub-clause 3.5.2.6. The local APSME shall act as the initiator of this protocol, the APSME indicated by the *ResponderAddress* parameter shall act as the responder of this protocol, and the *UseParent* parameter will control whether the messages are sent indirectly via the responder's parent device given by the *ResponderParentAddress* parameter.

3.5.2.2 APSME-ESTABLISH-KEY.confirm

This primitive is issued to the ZDO upon completion or failure of a key-establishment protocol.

3.5.2.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-ESTABLISH-KEY.confirm	{
	Address,
	Status
	}

Table 145 specifies the parameters of the APSME-ESTABLISH-KEY.confirm primitive. Table 149 gives a description of some codes that can be returned in the *Status* parameter of this primitive. In addition to these codes, if when sending one of the protocol messages, an NLDE-DATA.confirm primitive with a *Status*

parameter set to a value other than SUCCESS is issued, the *Status* parameter of the APSME-ESTABLISH-KEY.confirm primitive shall be set to that received from the NWK layer.

Table 145 APSME-ESTABLISH-KEY.confirm parameters

Name	Type	Valid Range	Description
Address	Device Address	Any valid 64-bit address	The extended 64-bit address of the device with which the key-establishment protocol was executed.
Status	Enumeration	Value given by Table 149 or any status value returned from the NLDE-DATA.confirm primitive.	This parameter indicates the final status of the key-establishment protocol.

3.5.2.2.2 When generated

The APSME in both the responder and initiator devices shall issue this primitive to the ZDO upon completion of a key-establishment protocol.

3.5.2.2.3 Effect on receipt

If key establishment is successful, the AIB of the initiator and responder shall be updated with the new link key and the initiator shall be able to securely communicate with the responder. If the key establishment was not successful, then the AIB shall not be changed.

3.5.2.3 APSME-ESTABLISH-KEY.indication

The APSME in the responder shall issue this primitive to its ZDO when it receives an initial key-establishment message (e.g., an SKKE-1 frame) from an initiator.

3.5.2.3.1 Semantics of the service primitive

This primitive shall provide the following interface:

```
APSME-ESTABLISH-KEY.indication      {
                                    InitiatorAddress,
                                    KeyEstablishmentMethod
                                    }
```

Table 146 specifies the parameters of the APSME-ESTABLISH-KEY.indication primitive.

Table 146 APSME-ESTABLISH-KEY.indication parameters

Name	Type	Valid Range	Description
InitiatorAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the initiator device.
KeyEstablishmentMethod	Integer	0x00 - 0x03	The requested key-establishment method shall be one of the following: 0x00 = SKKE method. 0x01-0x03: reserved.

3.5.2.3.2 When generated

The APSME in the responder device shall issue this primitive to the ZDO when a request to start a key-establishment protocol (e.g., an SKKE-1 frame) is received from an initiator and a master key associated with the initiator device is present in the AIB.

3.5.2.3.3 Effect on receipt

Upon receiving the APSME-ESTABLISH-KEY.indication primitive, the ZDO may use the *KeyEstablishmentMethod* and *InitiatorAddress* parameters to determine whether to establish a key with the initiator. The ZDO shall respond using the APSME-ESTABLISH-KEY.response primitive.

3.5.2.4 APSME-ESTABLISH-KEY.response

The ZDO of the responder device shall use the APSME-ESTABLISH-KEY.response primitive to respond to an APSME-ESTABLISH-KEY.indication primitive. The ZDO determines whether to continue with the key establishment or halt it. This decision is indicated in the Accept parameter of the APSME-ESTABLISH-KEY.response primitive.

3.5.2.4.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-ESTABLISH-KEY.response	{
	InitiatorAddress,
	Accept
	}

Table 147 specifies the parameters of the APSME-ESTABLISH-KEY.response primitive

Table 147 APSME-ESTABLISH-KEY.response parameters .

Name	Type	Valid Range	Description
InitiatorAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that initiated key establishment.
Accept	Boolean	TRUE | FALSE	This parameter indicates the response to an initiator's request to execute a key-establishment protocol. The response shall be either: TRUE = Accept. FALSE = Reject.

3.5.2.4.2 When generated

The APSME-ESTABLISH-KEY.response primitive shall be generated by the ZDO and provided to the APSME following a request from an initiator device to start a key-establishment protocol (i.e., after receipt of an APSME-ESTABLISH-KEY.indication). This primitive provides the responder's ZDO with an opportunity to determine whether to accept or reject a request to establish a key with a given initiator.

3.5.2.4.3 Effect on receipt

If the *Accept* parameter is TRUE, then the APSME of the *responder* will attempt to execute the key establishment protocol indicated by the *KeyEstablishmentMethod* parameter. If *KeyEstablishmentMethod* is equal to SKKE, the APSME shall execute the SKKE protocol, described in sub-clause 3.5.2.6. The local

APSME shall act as the responder of this protocol and the APSME indicated by the *InitiatorAddress* parameter shall act as the initiator of this protocol.

If the *Accept* parameter is FALSE, the local APSME shall halt and erase all intermediate data pertaining to the pending key-establishment protocol.

3.5.2.5 Data Service Message Sequence Chart

Figure 68 illustrates the sequence of primitives necessary for a successful key establishment between two devices.

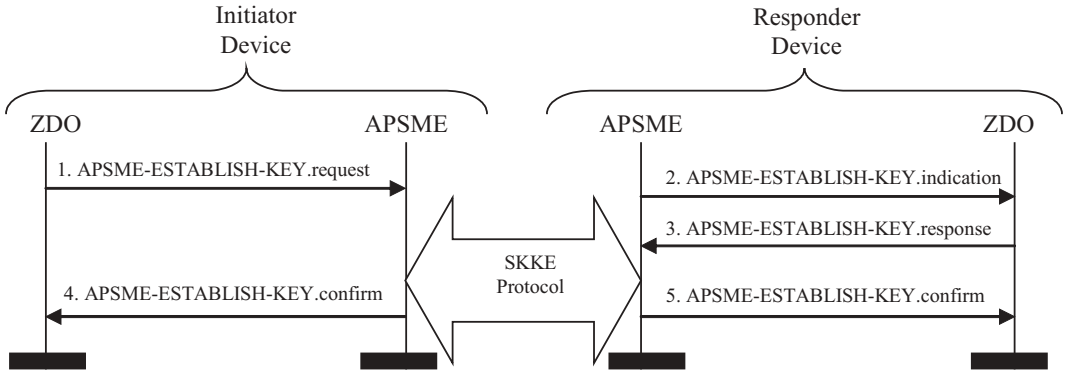


Figure 68 Sequence chart for successful APSME-ESTABLISH-KEY primitives

3.5.2.6 The SKKE Protocol

The APSME on the initiator and responder execute the symmetric-key key-agreement scheme instantiated in B.2.1 and specified in B.7. The shared key, as specified in B.7 prerequisite step 2, shall be the master key shared between the initiator and responder devices as obtained from the appropriate master key element in the *DeviceKeyPairSet* attribute in the AIB. The messages sent during the scheme specified in B.7 shall be assigned to the frame names given in Table 148. The formats for these SKKE frames are given in sub-clause 3.5.9.1. The initiator device is responsible for sending the SKKE-1 and SKKE-3 frames and the responder device is responsible for sending the SKKE-2 and SKKE-4 frames. Additionally, if the *UseParent* parameter to the APSME-ESTABLISH-KEY.request primitive is TRUE, the responder device's parent (as indicated by the *ResponderParentAddress* parameter to the APSME-ESTABLISH-KEY.request primitive) shall act as a liaison and forward messages between the initiator and responder devices.

During the key-establishment scheme, if the responder or initiator device detects any error condition listed in Table 149, the scheme shall be aborted and the local APSME shall issue the APSME-ESTABLISH-KEY.confirm primitive with the *Status* parameter set as indicated in Table 149. If no error conditions occur (i.e., the key-agreement scheme outputs 'valid'), then the initiator and responder shall consider the derived key (i.e., *KeyData*) as their newly shared link key. Both the initiator and responder shall update or add this link key to their AIB, set the corresponding incoming and outgoing frame counts to zero, and issue the APSME-ESTABLISH-KEY.confirm primitive with the *Status* parameter set to SUCCESS.

Table 148 Mapping of frame names to symmetric-key key agreement scheme messages

Frame Name	Description	Reference
SKKE-1	Sent by initiator during action step 1. (B.7.1)	3.5.2.6.2

Table 148 Mapping of frame names to symmetric-key key agreement scheme messages

SKKE-2	Sent by responder during action step 2. (B.7.2)	3.5.2.6.3
SKKE-3	Sent by initiator during action step 11. (B.7.1)	3.5.2.6.4
SKKE-4	Sent by responder during action step 8. (B.7.2)	3.5.2.6.5

Table 149 Mapping of symmetric-key key agreement error conditions to status codes

Status Description	Status Code	Value
No errors occur	SUCCESS	0x00
An invalid parameter was input to one of the key establishment primitives.	INVALID_PARAMETER	0x01
No master key is available	NO_MASTER_KEY	0x02
Challenge is invalid: Initiator during action step 4. (B.7.1) Responder during action step 1. (B.7.2)	INVALID_CHALLENGE	0x03
SKG outputs invalid: Initiator during action step 5. (B.7.1) Responder during action step 3. (B.7.2)	INVALID_SKG	0x04
MAC transformation outputs invalid: Initiator during action step 11. (B.7.1) Responder during action step 7. (B.7.2)	INVALID_MAC	0x05
Tag checking transformation outputs invalid: Initiator during action step 9. (B.7.1) Responder during action step 10. (B.7.2)	INVALID_KEY	0x06
Either the initiator or responder waits for an expected incoming message for time greater than the *apsSecurityTimeOutPeriod* attribute of the AIB.	TIMEOUT	0x07
Either the initiator or responder receives an SKKE frame out of order.	BAD_FRAME	0x08

3.5.2.6.1 Generating and sending the initial SKKE-1 frame

The SKKE protocol begins with the initiator device sending an SKKE-1 frame. The SKKE-1 command frame shall be constructed as specified in sub-clause 3.5.9.1.

If the *UseParent* parameter to the APSME-ESTABLISH-KEY.request primitive is FALSE, the initiator device shall begin the protocol by sending this SKKE-1 frame directly to the responder device (as indicated by the *ResponderAddress* parameter to the APSME-ESTABLISH-KEY.request primitive). Otherwise, the initiator device shall begin the protocol by sending this SKKE-1 frame to the responder device's parent (as indicated by the *ResponderParentAddress* parameter to the APSME-ESTABLISH-KEY.request primitive). The SKKE-1 frame shall be sent using the NLDE-DATA.request primitive with NWK layer security set to the default NWK layer security level.

3.5.2.6.2 On receipt of the SKKE-1 frame

If the responder address field of the SKKE-1 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-1 frame shall be discarded.
2. Otherwise, the APSME of the local device shall send the SKKE-1 frame to the responder device using the NLDE-DATA.request primitive, with the *DestAddr* parameter set to the 16-bit address corresponding to the 64-bit address in the responder address field of the SKKE-1 frame, the *DiscoverRoute* parameter set to 0x01[180], and the *SecurityEnable* parameter set to FALSE.
3. Otherwise, the APSME shall perform the following steps:
4. If the device does not have a master key corresponding to the initiator address field, the SKKE-1 frame shall be discarded and the APSME-ESTABLISH-KEY.confirm primitive shall be issued with the *Status* parameter set to NO_MASTER_KEY (see Table 149). The APSME should halt processing for this SKKE protocol.
5. Otherwise, the APSME shall issue an APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the initiator address field of the SKKE-1 frame and the *KeyEstablishmentMethod* parameter set to 0 (i.e., the SKKE protocol).
6. After issuing the APSME-ESTABLISH-KEY.indication primitive, and upon receipt of the corresponding APSME-ESTABLISH-KEY.response primitive, the APSME shall evaluate the *InitiatorAddress* and *Accept* parameters of the received APSME-ESTABLISH-KEY.response primitive. If the *InitiatorAddress* parameter is set to the initiator address of the SKKE-1 frame and the *Accept* parameter set to FALSE, the APSME shall halt the SKKE protocol and discard the SKKE-1 frame.
7. Otherwise, it shall construct an SKKE-2 frame as specified in sub-clause 3.5.9.1. If the source of the SKKE-1 frame indicates the same device as the initiator address field of the SKKE-1 frame, the device shall send this SKKE-2 frame directly to the initiator device using the NLDE-DATA.request primitive, with the *DestAddr* parameter set to the source of the SKKE-1 frame, the *DiscoverRoute* parameter set to 0x01[181], and the *SecurityEnable* parameter set to TRUE. Otherwise, the device shall send the SKKE-2 frame to its parent using the NLDE-DATA.request primitive, with the *DiscoverRoute* parameter set to [182], and the *SecurityEnable* parameter set to FALSE.

3.5.2.6.3 On receipt of the SKKE-2 frame

If the initiator address field of the SKKE-2 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-2 frame shall be discarded.
2. Otherwise, the device shall send the SKKE-2 to the initiator device using the NLDE-DATA.request primitive with NWK layer set to the default level.

Otherwise, the device shall construct an SKKE-3 frame as specified in sub-clause 3.5.9.1. If the source of the SKKE-2 frame is the same as the responder address field of the SKKE-2 frame, the device shall send this SKKE-3 frame directly to the responder device. Otherwise, the device shall send the SKKE-3 frame to the responder's parent. The SKKE-3 frame shall be sent using the NLDE-DATA.request primitive with NWK layer security set to the default NWK layer security level.

[180]CCB Comment #256
[181]Ibid
[182]Ibid

3.5.2.6.4 On receipt of the SKKE-3 frame

If the responder address field of the SKKE-3 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-3 frame shall be discarded.
2. Otherwise, the device shall send the SKKE-3 to the responder device using the NLDE-DATA.request primitive with NWK layer security disabled.

Otherwise, the device shall process the SKKE-3 data field and if the protocol was not a success it shall issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the initiator's address and the *Status* parameter set appropriately.

If, from the device's perspective, the protocol was a success, the device shall construct an SKKE-4 frame as specified in sub-clause 3.5.9.1. If the source of the SKKE-3 frame is the same as the initiator address field of the SKKE-3 frame, the device shall send this SKKE-4 frame directly to the initiator device using the NLDE-DATA.request primitive with NWK layer security set to the default level. Otherwise, the device shall send the SKKE-4 frame to its parent using the NLDE-DATA.request primitive with NWK layer security disabled. Finally, the device shall issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set the initiator's address and the *Status* parameter set to success.

3.5.2.6.5 On receipt of the SKKE-4 frame

If the initiator address field of the SKKE-4 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-4 frame shall be discarded.
2. Otherwise, the APSME of the local device shall send the SKKE-4 to the initiator device using the NLDE-DATA.request primitive with NWK layer set to the default level.

Otherwise, the APSME shall process the SKKE-4 frame and issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set the responder's address and the *Status* parameter set appropriately.

3.5.3 Transport-Key Services

The APSME provides services that allow an initiator to transport keying material to a responder. The different types of keying material that can be transported are shown in Table 151.

3.5.3.1 APSME-TRANSPORT-KEY.request

The APSME-TRANSPORT-KEY.request primitive is used for transporting a key to another device.

3.5.3.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-TRANSPORT-KEY.request	{
	DestAddress,
	KeyType,
	TransportKeyData
	}

Table 150 specifies the parameters for the APSME-TRANSPORT-KEY.request primitive.

Table 150 APSME-TRANSPORT-KEY.request parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the destination device.
KeyType	Integer	0x00 – 0x03	Identifies the type of key material that should be transported. See Table 151.
TransportKeyData	Variable	Variable	The key being transported along with identification and usage parameters. The type of this parameter depends on the *KeyType* parameter as follows: *KeyType* = 0x00 see Table 152 *KeyType* = 0x01 see Table 153 *KeyType* = 0x02 see Table 154 *KeyType* = 0x03 see Table 154

Table 151 *KeyType* parameter of the transport-key primitive

Enumeration	Value	Description
Trust-center master key	0x00	Indicates the key is a master key which is used to set up link keys between the trust center and another device.
Network key	0x01	Indicates the key is a Network key.
Application master key	0x02	Indicates the key is a master key which is used to set up link keys between two devices.
Application link key	0x03	Indicates the key is a link key which is used as a basis of security between two devices.

Table 152 *TransportKeyData* parameter for a trust-center master key

Parameter Name	Type	Valid Range	Description
ParentAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the parent of the destination device given by the *DestAddress* parameter.
TrustCenter-Master-Key	Set of 16 octets	Variable	The trust center master key.

Table 153 *TransportKeyData* parameter for a Network key

Parameter Name	Type	Valid Range	Description
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys for purposes of key updates, and incoming frame security operations.
NetworkKey	Set of 16 octets	Variable	The Network key.
UseParent	Boolean	TRUE | FALSE	This parameter indicates if the destination device's parent shall be used to forward the key to the destination device: TRUE: Use parent FALSE: Do not use parent
ParentAddress	Device address	Any valid 64-bit address	If the *UseParent* is TRUE, then *ParentAddress* parameter shall contain the extended 64-bit address of the destination device's parent device. Otherwise, this parameter is not used and need not be set.

Table 154 *TransportKeyData* parameter for an application master or link key

Parameter Name	Type	Valid Range	Description
PartnerAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the device that was also sent this master key.
Initiator	Boolean	TRUE | FALSE	This parameter indicates if the destination device of this master key requested it: TRUE: If the destination requested the key. FALSE: otherwise.
Key	Set of 16 octets	Variable	The master or link key (as indicated by the *KeyType* parameter).

3.5.3.1.2 When generated

The ZDO on an initiator device shall generate this primitive when it requires a key to be transported to a responder device.

3.5.3.1.3 Effect on receipt

The receipt of an APSME-TRANSPORT-KEY.request primitive shall cause the APSME to create a transport-key command packet (see sub-clause 3.5.9.2)

If the *KeyType* parameter is 0x00 (i.e., trust center master key), the key descriptor field of the transport-key command shall be set as follows. The key sub-field shall be set to the *Key* sub-parameter of the *TransportKeyData* parameter, the destination address sub-field shall be set to the *DestinationAddress* parameter, and the source address sub-field shall be set to the local device address. This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to

the device specified by the *ParentAddress* sub-parameter of the *TransportKeyData* parameter by issuing a NLDE-DATA.request primitive.

If the *KeyType* parameter is 0x01 (i.e., Network key), the key descriptor field of the transport-key command shall be set as follows. The key sub-field shall be set to the *Key* sub-parameter of the *TransportKeyData* parameter, the sequence number sub-field shall be set to the *KeySeqNumber* sub-parameter of the *TransportKeyData* parameter, the destination address sub-field shall be set to the *DestinationAddress* parameter, and the source address sub-field shall be set to the local device address. This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *ParentAddress* sub-parameter of the *TransportKeyData* parameter (if the *UseParent* sub-parameter of the *TransportKeyData* parameter is TRUE[183]) or the *DestinationAddress* parameter (if the *UseParent* sub-parameter of the *TransportKeyData* parameter is FALSE[184]) by issuing a NLDE-DATA.request primitive.

If the *KeyType* parameter is 0x02 or 0x03 (i.e., an application master or link key), the key descriptor field of the transport-key command shall be set as follows. The key sub-field shall be set to the *Key* sub-parameter of the *TransportKeyData* parameter, the partner address sub-field shall be set to the *PartnerAddress* sub-parameter of the *TransportKeyData* parameter, and the initiator sub-field shall be set 1 (if the *Initiator* sub-parameter of the *TransportKeyData* parameter is TRUE) or 0 (if the *Initiator* sub-parameter of the *TransportKeyData* parameter is FALSE). This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestinationAddress* parameter by issuing a NLDE-DATA.request primitive.

3.5.3.2 APSME-TRANSPORT-KEY.indication

The APSME-TRANSPORT-KEY.indication primitive is used to inform the ZDO of the receipt of keying material.

3.5.3.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

```
APSME-TRANSPORT-KEY.indication      {
                                    SrcAddress,
                                    KeyType,
                                    TransportKeyData
                                    }
```

Table 155 specifies the parameters of the APSME-TRANSPORT-KEY.indication primitive.

Table 155 APSME-TRANSPORT-KEY.indication parameters

Name	Type	Valid Range	Description

[183]CCB Comment #141
[184]Ibid

Table 155 APSME-TRANSPORT-KEY.indication parameters

SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that is the original source of the transported key.
KeyType	Octet	0x00 – 0x03	Identifies the type of key material that was be transported. See Table 151.
TransportKeyData	Variable	Variable	The key that was transported along with identification and usage parameters. The type of this parameter depends on the *KeyType* parameter as follows: *KeyType* = 0x00 see Table 156. *KeyType* = 0x01 see Table 157. *KeyType* = 0x02 see Table 154. *KeyType* = 0x03 see Table 154.

Table 156 TransportKeyData parameter for a trust-center master key

Parameter Name	Type	Valid Range	Description
TrustCenter-Master-Key	Set of 16 octets	Variable	The trust center master key.

Table 157 *TransportKeyData* parameter for a Network key

Parameter Name	Type	Valid Range	Description
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys for purposes of key updates, and incoming frame security operations.
NetworkKey	Set of 16 octets	Variable	The Network key.

3.5.3.2.2 When generated

The APSME shall generate this primitive when it receives a transport-key command that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2, that has the key type field set to 2 or 3 (i.e., application link or master key).

Alternatively, the APSME shall generate this primitive when it receives a transport-key command that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2, that has the key type field set to 0 or 1 (i.e., a trust center master key or Network key) and the destination address sub-field of the key descriptor field is equal to the local address.

3.5.3.2.3 Effect on receipt

Upon receipt of this primitive, the ZDO is informed of the receipt of the keying material.

3.5.3.3 Upon Receipt of a Transport-Key Command

Upon receipt of a transport-key command, the APSME shall execute security processing as specified in sub-clause 3.5.1.2 and then check the key type sub-field.

If the key type field is set to 2 or 3 (i.e., application link or master key), the APSME shall issue the APSME-TRANSPORT-KEY.indication[185] primitive with the *SrcAddress* parameter set to the source of the key-transport command (as indicated by the NLDE-DATA.indication *SrcAddress* parameter), the *KeyType* parameter set to the key type field. The *TransportKeyData* parameter shall be set as follows: the *Key* sub-parameter shall be set to the key field, *PartnerAddress* sub-parameter shall be set to the partner address field, the *Initiator* parameter shall be set to TRUE if the initiator field is 1, otherwise 0.

If the key type field is set to 0 or 1 (i.e., trust center master key or NWK key[186]) and the destination address field is equal to the local address, the APSME shall issue the APSME-TRANSPORT-KEY.indication[187] primitive. The *SrcAddress* parameter set to the source address field of the key-transport command, the *KeyType* parameter set to the key type field. The *TransportKeyData* parameter shall be set as follows: the *Key* sub-parameter shall be set to the key field and, in the case of a Network key (i.e., the key type field is set to 1), the *KeySeqNumber* sub-parameter shall be set to the sequence number field.

If the key type field is set to 0 or 1 (i.e., trust center master key or NWK key[188]) and the destination address field is not equal to the local address, the APSME shall send the command to the address indicated by the destination address field by issuing the NLDE-DATA.request primitive with security disabled.

Upon receipt of an unsecured transport-key command, the APSME shall check the key type sub-field. If the key type field is set to 0 (i.e., a trust center master key), the destination address field is equal to the local address, and the device does not have a trust center master key and address (i.e., the *apsTrustCenterAddress* in the AIB), then the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive. Also, if the key type field is set to 1 (i.e., Network key), the destination address field is equal to the local address, and the device does not have a Network key, then the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive. If an APSME-TRANSPORT-KEY.indication primitive is issued, the SrcAddress parameter shall be set to the source address field of the key-transport command, and the KeyType parameter shall be set to the key type field. The TransportKeyData parameter shall be set as follows: the Key sub-parameter shall be set to the key field and, in the case of a Network key (i.e., the key type field is set to 1), the KeySeqNumber sub-parameter shall be set to the sequence number field.[189]

3.5.4 Update-Device Services

The APSME provides services that allow a device (e.g., a router) to inform another device (e.g., a trust center) that a third device has changed its status (e.g., joined or left the network).

3.5.4.1 APSME-UPDATE-DEVICE.request

The ZDO shall issue this primitive when it wants to inform a device (e.g., a trust center) that another device has a status that needs to be updated (e.g., the device joined or left the network).

[185] CCB Comment #163
[186] CCB Comment #162
[187] CCB Comment #163
[188] CCB Comment #162
[189] CCB Comment #161

3.5.4.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-UPDATE-DEVICE.request	{
	DestAddress,
	DeviceAddress,
	Status,
	DeviceShortAddress
	}

Table 158 specifies the parameters for the APSME- UPDATE-DEVICE.request primitive.

Table 158 APSME-UPDATE-DEVICE.request parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that shall be sent the update information.
DeviceAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device whose status is being updated.
Status	Integer	0x00 – 0x08	Indicates the updated status of the device given by the *DeviceAddress* parameter. 0x00: device secured join. 0x01: device unsecured join. 0x02: device left. 0x03-0x08 reserved.
DeviceShortAddress	Network address	0x0000 - 0xffff	The 16-bit network address of the device whose status is being updated.[a]

[a]CCB Comment

3.5.4.1.2 When generated

The ZDO (e.g., on a router or coordinator) shall initiate the APSME-UPDATE-DEVICE.request primitive when it wants to send updated device information to another device (e.g., the trust center).

3.5.4.1.3 Effect on receipt

Upon receipt of the APSME-UPDATE-DEVICE.request primitive the device shall first create an update-device command frame (see sub-clause 3.5.9.4). The device address field of this command frame shall be set to the *DeviceAddress* parameter and the status field shall be set according to the *Status* parameter and the device short address field shall be set to the DeviceShortAddress parameter[190]. This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

3.5.4.2 APSME-UPDATE-DEVICE.indication

The APSME shall issue this primitive to inform the ZDO that it received an update-device command frame.

[190]CCB Comment #142

3.5.4.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-UPDATE-DEVICE.indication	{
	SrcAddress,
	DeviceAddress,
	Status,
	DeviceShortAddress
	}

Table 159 specifies the parameters for the APSME-UPDATE-DEVICE.indication primitive.

Table 159 APSME-UPDATE-DEVICE.indication parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device originating the update-device command.
DeviceAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device whose status is being updated.
Status	Octet	0x00 – 0xFF	Indicates the updated status of the device given by the *DeviceAddress* parameter. 0x00: device secured join. 0x01: device unsecured join. 0x02: device left. 0x03-0xFF reserved.
DeviceShortAddress	Network address	0x0000 - 0xffff	The 16-bit network address of the device whose status is being updated.[a]

[a]CCB Comment #142

3.5.4.2.2 When generated

The APSME shall generate this primitive when it receives an update-device command frame that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2.

3.5.4.2.3 Effect on receipt

Upon receipt of the APSME-UPDATE-DEVICE.indication primitive the ZDO will be informed that the device referenced by the *DeviceAddress* parameter has undergone a status update according to the *Status* parameter.

3.5.5 Remove Device Services

The APSME provides services that allow a device (e.g., a trust center) to inform another device (e.g., a router) that one of its children should be removed from the network.

3.5.5.1 APSME-REMOVE-DEVICE.request

The ZDO of a device (e.g., a trust center) shall issue this primitive when it wants to request that a parent device (e.g., a router) remove one of its children from the network. For example, a trust center can use this primitive to remove a child device that fails to authenticate properly.

3.5.5.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-REMOVE-DEVICE.request	{
	ParentAddress,
	ChildAddress
	}

Table 160 specifies the parameters for the APSME-REMOVE-DEVICE.request primitive.

Table 160 APSME- REMOVE-DEVICE.request parameters

Parameter Name	Type	Valid Range	Description
ParentAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that is the parent of the child device that is requested to be removed.
ChildAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the child device that is requested to be removed.

3.5.5.1.2 When generated

The ZDO (e.g., on a trust) shall initiate the APSME-REMOVE-DEVICE.request primitive when it wants to request that a parent device (specified by the *ParentAddress* parameter) remove one of its child devices (as specified by the *ChildAddress* parameter).

3.5.5.1.3 Effect on receipt

Upon receipt of the APSME-REMOVE-DEVICE.request primitive the device shall first create a remove-device command frame (see sub-clause 3.5.9.4). The child address field of this command frame shall be set to the *ChildAddress* parameter. This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *ParentAddress* parameter by issuing a NLDE-DATA.request primitive.

3.5.5.2 APSME-REMOVE-DEVICE.indication

The APSME shall issue this primitive to inform the ZDO that it received a remove-device command frame.

3.5.5.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-REMOVE-DEVICE.indication	{
	SrcAddress,
	ChildAddress
	}

Table 161 specifies the parameters for the APSME-REMOVE-DEVICE.indication primitive.

Table 161 APSME-REMOVE-DEVICE.indication parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device requesting that a child device be removed.
ChildAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the child device that is requested to be removed.

3.5.5.2.2 When generated

The APSME shall generate this primitive when it receives a remove-device command frame that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2.

3.5.5.2.3 Effect on receipt

Upon receipt of the APSME-REMOVE-DEVICE.indication primitive the ZDO shall be informed that the device referenced by the *SrcAddress* parameter is requesting that the child device referenced by the *ChildAddress* parameter be removed from the network.

3.5.6 Request Key Services

The APSME provides services that allow a device to request the current Network key or a master key from another device (e.g., its trust center).

3.5.6.1 APSME-REQUEST-KEY.request

This primitive allows the ZDO to request either the current Network key or a new end-to-end application master key.

3.5.6.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

```
APSME-REQUEST-KEY.request       {
                                DestAddress,
                                KeyType,
                                PartnerAddress
                                }
```

Table 162 specifies the parameters for the APSME-REQUEST-KEY.request primitive.

Table 162 APSME-REQUEST-KEY.request parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device to which the request-key command should be sent.

Table 162 APSME-REQUEST-KEY.request parameters

KeyType	Octet	0x00-0xFF	The type of key being requested: 0x01 = Network key 0x02 = Application key 0x00 and 0x03-0xFF = Reserved
PartnerAddress	Device Address	Any valid 64-bit address	In the case that *KeyType* parameter indicates an application key, this parameter shall indicate an extended 64-bit address of a device that shall receive the same key as the device requesting the key.

3.5.6.1.2 When generated

The ZDO of a device shall generate the APSME-REQUEST-KEY.request primitive when it requires either the current Network key or a new end-to-end application master key.

3.5.6.1.3 Effect on receipt

Upon receipt of the APSME-REQUEST-KEY.request primitive the device shall first create a request-key command frame (see sub-clause 3.5.9.6). The key type field of this command frame shall be set to the same value as the *KeyType* parameter. If the *KeyType* parameter is 0x02 (i.e., an application key), then the partner address field of this command frame shall be the *PartnerAddress* parameter. Otherwise, the partner address field of this command frame shall not be present.

This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

3.5.6.2 APSME-REQUEST-KEY.indication

The APSME shall issue this primitive to inform the ZDO that it received a request-key command frame.

3.5.6.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

```
APSME-REQUEST-KEY.indication        {
                                    SrcAddress,
                                    KeyType,
                                    PartnerAddress
                                    }
```

Table 163 specifies the parameters for the APSME-REQUEST-KEY.indication primitive.

Table 163 APSME-REQUEST-KEY.indication parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the request-key command.

Table 163 APSME-REQUEST-KEY.indication parameters

KeyType	Octet	0x00-0xFF	The type of key being requested: 0x01 = Network key 0x02 = Application key 0x00 and 0x03-0xFF = Reserved
PartnerAddress	Device Address	Any valid 64-bit address	In the case that *KeyType* parameter indicates an application key, this parameter shall indicate an extended 64-bit address of a device that shall receive the same key as the device requesting the key.

3.5.6.2.2 When generated

The APSME shall generate this primitive when it receives a request-key command frame that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2.

3.5.6.2.3 Effect on receipt

Upon receipt of the APSME-REQUEST-KEY.indication primitive the ZDO shall be informed that the device referenced by the *SrcAddress* parameter is requesting a key. The type of key being requested shall be indicated by the *KeyType* parameter and if the *KeyType* parameter is 0x02 (i.e., an application key), the *PartnerAddress* parameter shall indicate a partner device that shall receive the same key as the device requesting the key (i.e., the device indicated by the *SrcAddress* parameter).

3.5.7 Switch Key Services

The APSME provides services that allow a device (e.g., a trust center) to inform another device that it should switch to a new active Network key.

3.5.7.1 APSME-SWITCH-KEY.request

This primitive allows a device (e.g., the trust center) to request that another device switch to a new active Network key.

3.5.7.1.1 Semantics of the service primitive

This primitive shall provide the following interface:

```
APSME-SWITCH-KEY.request        {
                                DestAddress,
                                KeySeqNumber
                                }
```

Table 164 specifies the parameters for the APSME-SWITCH-KEY.request primitive.

Table 164 APSME-SWITCH-KEY.request parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device to which the switch-key command is sent.
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys.

3.5.7.1.2 When generated

The ZDO of a device (e.g., the trust center) shall generate the APSME-SWITCH-KEY.request primitive when it wants to inform a device to switch to a new active Network key.

3.5.7.1.3 Effect on receipt

Upon receipt of the APSME-SWITCH-KEY.request primitive the device shall first create a switch-key command frame (see sub-clause 3.5.9.7). The sequence number field of this command frame shall be set to the same value as the *KeySeqNumber* parameter.

This command frame shall be security protected as specified in sub-clause 3.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

3.5.7.2 APSME-SWITCH-KEY.indication

The APSME shall issue this primitive to inform the ZDO that it received a switch-key command frame.

3.5.7.2.1 Semantics of the service primitive

This primitive shall provide the following interface:

APSME-SWITCH-KEY.indication	{
	SrcAddress,
	KeySeqNumber
	}

ATable 165 specifies the parameters for the APSME-SWITCH-KEY.indication primitive.

Table 165 APSME-SWITCH-KEY.indication parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the switch-key command.
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys.

3.5.7.2.2 When generated

The APSME shall generate this primitive when it receives a switch-key command frame that is successfully decrypted and authenticated, as specified in sub-clause 3.5.1.2.

3.5.7.2.3 Effect on receipt

Upon receipt of the APSME-SWITCH-KEY.indication primitive the ZDO shall be informed that the device referenced by the *SrcAddress* parameter is requesting that the Network key referenced by the *KeySeqNumber* parameter become the new active Network key.

3.5.8 Secured APDU Frame

The APS layer frame format from [B7] consists of APS header and APS payload fields. The APS header consists of frame control and addressing fields. When security is applied to an APDU frame, the security bit in the APS frame control field shall be set to 1 to indicate the presence of the auxiliary frame header. The format for the auxiliary frame header is given in sub-clause 3.6.1. The format of a secured APS layer frame is shown in Table 69. The auxiliary frame header is situated between the APS header and payload fields.

Figure 69 Secured APS layer frame format

<table>
<tr><th>Octets: variable</th><th>5 or 6</th><th colspan="2">Variable</th></tr>
<tr><td rowspan="2">Original APS Header
([B7], Clause 7.1)</td><td rowspan="2">Auxiliary frame header</td><td>Encrypted Payload</td><td>Encrypted Message Integrity Code (MIC)</td></tr>
<tr><td colspan="2">Secure frame payload = Output of CCM*</td></tr>
<tr><td colspan="2">Full APS header</td><td colspan="2">Secured APS payload</td></tr>
</table>

3.5.9 Command Frames

The APS layer command frame formats are given in this clause.

Command identifier values are shown in Table 166)[191].

Table 166 Command identifier values

Command identifier	Value
APS_CMD_SKKE_1	0X01
APS_CMD_SKKE_2	0X02
APS_CMD_SKKE_3	0X02
APS_CMD_SKKE_4	0X04
APS_CMD_TRANSPORT_KEY	0X05
APS_CMD_UPDATE_DEVICE	0X06
APS_CMD_REMOVE_DEVICE	0X07
APS_CMD_REQUEST_KEY	0X08
APS_CMD_SWITCH_KEY	0X09

3.5.9.1 Key-Establishment Commands

The APS command frames used during key establishment is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

[191]CCB Comment #205

The generic SKKE command frame shall be formatted as illustrated in Table 70.

Figure 70 Generic SKKE frame command format

Octets: 1	1	8	8	16
Frame control	Command identifier	Initiator Address	Responder Address	Data
APS Header	Payload			

3.5.9.1.1 Command identifier field

The command identifier field shall indicate the APS command type. For SKKE frames, the command identifier shall indicate either an SKKE-1, SKKE-2, SKKE-3, or SKKE-4 frame, depending on the frame type (see Table 166)[192].

3.5.9.1.2 Initiator address field

The initiator address field shall be the 64-bit extended address of the device that acts as the initiator in the key-establishment protocol.

3.5.9.1.3 Responder address field

The responder address field shall be the 64-bit extended address of the device that acts as the responder in the key-establishment protocol.

3.5.9.1.4 Data field

The content of the data field depends on the command identifier field (i.e., SKKE-1, SKKE-2, SKKE-3, or SKKE-4). The following clauses describe the content of the data field for each command type.

3.5.9.1.4.1 SKKE-1 frame

The data field shall be the octet representation of the challenge *QEU* generated by the initiator during action step 1. of clause B.7.1.

3.5.9.1.4.2 SKKE-2 frame

The data field shall be the octet representation of the challenge *QEV* generated by the responder during action step 2. of clause B.7.2.

3.5.9.1.4.3 SKKE-3 frame

The data field shall be the octet representation of the string $MacTag_2$ generated by the initiator during action step 11. of clause B.7.1.

3.5.9.1.4.4 SKKE-4 frame

The data field shall be the octet representation of the string $MacTag_1$ generated by the responder during action step 7. of clause B.7.2.

[192] Ibid

3.5.9.2 Transport-Key Commands

The transport-key command frame shall be formatted as illustrated in Table 71. The optional fields of the APS header portion of the general APS frame format shall not be present.

Figure 71 Transport-key command frame

Octets: 1	1	1	variable
Frame control	APS command identifier	Key Type	Key descriptor
APS Header	Payload		

3.5.9.3 Command identifier field

This field is 8-bits in length shall be set to indicate that this is a transport-key command frame (see Table 166)[193].

3.5.9.3.1 Key type field

This field is 8-bits in length and describes the type of key being transported. The different types of keys are enumerated in Table 151.

3.5.9.3.2 Key descriptor field

This field is variable in length and shall contain the actual (unprotected) value of the transported key along with any relevant identification and usage parameters. The information in this field depends on the type of key being transported (as indicated by the key type field – see sub-clause 3.5.9.3.1) and shall be set to one of the formats described in the following subsections.

3.5.9.3.2.1 Trust center master key descriptor field

If the key type field is set to 0, the key descriptor field shall be formatted as shown in Table 72.

Figure 72 Trust center master key descriptor field in transport-key command

Octets: 16	8	8
Key	Destination address	Source address

The key sub-field shall contain the master key that should be used to set up link keys with the trust center.

The destination address sub-field shall contain the address of the device which should use this master key.

The source address sub-field shall contain the address of the device (e.g., the trust center) which originally sent this master key.

[193] CCB Comment #205

3.5.9.3.2.2 Network key descriptor field

If the key type field is set to 1, this field shall be formatted as shown in Table 73.

Figure 73 Network key descriptor field in transport-key command

Octets: 16	1	8	8
Key	Sequence number	Destination address	Source address

The key sub-field shall contain a Network key.

The sequence number sub-field shall contain the sequence number associated with this Network key.

The destination address sub-field shall contain the address of the device which should use this Network key.

The source address field sub-shall contain the address of the device (e.g., the trust center) which originally sent this Network key.

3.5.9.3.2.3 Application master and link key descriptor field

If the key type field is set to 2 or 3, this field shall be formatted as shown in Table 74.

Figure 74 Application master key descriptor in transport-key command

Octets: 16	8	1
Key	Partner address	Initiator flag

The key sub-field shall contain a master or link key that is shared with the device identified in the partner address field.

The partner address sub-field shall contain the address of the other device that was sent this link or master key.

The initiator flag sub-field shall be set to 1 if the device receiving this packet requested this key. Otherwise, this sub-field shall be set to 0.

3.5.9.4 Update-Device Commands

The APS command frame used for device updates is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The update-device command frame shall be formatted as illustrated in Table 75.

Figure 75 Update-device command frame format

Octets: 1	1	8	2	1
Frame control	Command identifier	Device Address	Device short address[a]	Status
APS Header	Payload			

[a]CCB Comment #142

3.5.9.4.1 Command identifier field

The command identifier field shall indicate the APS command type update-device (see Table 166)[194].

3.5.9.4.2 Device address field

The device address field shall be the 64-bit extended address of the device whose status is being updated.

3.5.9.4.3 Device short address field

The device short address field shall be the 16-bit network address of the device whose status is being updated.[195]

3.5.9.4.4 Status field

The status field shall be assigned a value as described for the Status parameter in Table 149.[196]

3.5.9.5 Remove Device Commands

The APS command frame used for removing a device is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The remove-device command frame shall be formatted as illustrated in Table 76.

Figure 76 Remove-device command frame format

Octets: 1	1	8
Frame control	Command identifier	Child address
APS Header	Payload	

3.5.9.5.1 Command identifier field

The command identifier field shall indicate the APS command type remove-device (see Table 166)[197].

3.5.9.5.2 Child address field

The child address field shall be the 64-bit extended address of the device that is requested to be removed from the network.

3.5.9.6 Request-Key Commands

The APS command frame used by a device for requesting a key is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The request-key command frame shall be formatted as illustrated in Figure 77

Figure 77 Request-key command frame format

Octets: 1	1	1	0/8[a]
Frame control	Command identifier	Key type	Partner address
APS Header	Payload		

[a]CCB Comment #143

[194]CCB Comment #205
[195]CCB Comment #142
[196]CCB Comment #140
[197]CCB Comment #205

3.5.9.6.1 Command identifier field

The command identifier field shall indicate the APS command type request-key (see Table 166)[198].

3.5.9.6.2 Key type field

The key type field shall be set to 1 when the Network key is being requested and shall be set to 2 when an application key is being requested.

3.5.9.6.3 Partner address field

When the key type field is 2 (i.e., an application key), the partner address field shall contain the extended 64-bit address of the partner device that shall be sent the key. Both the partner device and the device originating the request-key command will be sent the key.

When the key-type field is 1 (i.e., Network key), the partner address field will not be present.

3.5.9.7 Switch-Key Commands

The APS command frame used by a device for requesting a key is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The switch-key command frame shall be formatted as illustrated in Table 78.

Figure 78 Switch-key command frame format

Octets: 1	1	1
Frame control	Command identifier	Sequence number
APS Header	Payload	

3.5.9.7.1 Command identifier field

The command identifier field shall indicate the APS command type switch-key (see Table 166)[199].

3.5.9.7.2 Sequence number field

The sequence number field shall contain the sequence number identifying the Network key to make active.

3.5.10 Security-Related AIB Attributes

The AIB contains attributes that are required to manage security for the APS layer. Each of these attributes can be read or written using the APSME-GET.request and APSME-SET.request primitives, respectively. The security-related attributes contained in the APS PIB are presented in Table 167 and Table 168.

Table 167 AIB security attributes

Attribute	Identifier	Type	Range	Description	Default
apsDeviceKeyPairSet	0xaa[a]	Set of Key-Pair Descriptor entries. See Table 168	Variable	A set of key-pair descriptors containing master and link key pairs shared with other devices.	-

[198] Ibid

[199] CCB Comment #205

Table 167 AIB security attributes

apsTrustCenterAddress	0xab[b]	Device address	Any valid 64-bit address	Identifies the address of the device's trust center	-
apsSecurityTime-OutPeriod	0xac[c]	Integer	0x0000-0xFFFF	The period of time a device will wait for an expected security protocol frame (in milliseconds).	1000

[a]CCB Comment #150
[b]Ibid
[c]Ibid

Table 168 Elements of the key-pair descriptor

Name	Type	Range	Description	Default
DeviceAddress	Device address	Any valid 64-bit address	Identifies the address of the entity with which this key-pair is shared.	-
MasterKey	Set of 16 octets	-	The actual value of the master key.	-
LinkKey	Set of 16 octets	-	The actual value of the link key.	-
OutgoingFrame-Counter	Set of 4 octets	0x00000000-0xFFFFFFFF	Unique identifier of the key originating with the device indicated by *KeySrcAddress*.	0x00000000
IncomingFrame-Counter	Set of 4 octets	0x00000000-0xFFFFFFFF	Incoming frame counter value corresponding to *DeviceAddress*.	0x00000000

3.6 Common Security Elements

This clause describes security-related features that are used in more than one ZigBee layer. The NWK and APS layers shall use the auxiliary header as specified in sub-clause 3.6.1. The MAC, NWK, and APS layers shall use the security parameters specified in sub-clause 3.6.2. The formatting of all frames and fields in this specification are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the leftmost bit is transmitted first in time. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are sent to the next layer in the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.[200]

[200]CCB Comment #98

3.6.1 Auxiliary Frame Header Format

The auxiliary frame header, as illustrated by Table 79, shall include a security control field and a frame counter field, and may include a sender address field and key sequence number field.

Figure 79 Auxiliary frame header format

Octets: 1	4	0/8	0/1
Security control	Frame Counter	Source Address	Key Sequence Number

3.6.1.1 Security Control Field

The security control field shall consist of a security level, a key identifier, and an extended nonce sub-field and shall be formatted as shown in Table 80.

Figure 80 Security control field format

Bit: 0-2	3-4	5	6-7
Security level	Key identifier	Extended Nonce	Reserved

3.6.1.1.1 Security level sub-field

The security level identifier indicates how an outgoing frame is to be secured, respectively, how an incoming frame purportedly has been secured: it indicates whether or not the payload is encrypted and to what extent data authenticity over the frame is provided, as reflected by the length of the message integrity code (MIC). The bit-length of the MIC may take the values 0, 32, 64 or 128 and determines the probability that a random guess of the MIC would be correct. The security properties of the security levels are listed in Table 169.

Table 169 Security levels available to the MAC, NWK, and APS layers

Security level identifier	Security Level Sub-Field (Table 80)	Security Attributes	Data Encryption	Frame Integrity (length M of MIC, in number of octets)
0x00	‘000’	None	OFF	NO (M = 0)
0x01	‘001’	MIC-32	OFF	YES (M=4)
0x02	‘010’	MIC-64	OFF	YES (M=8)
0x03	‘011’	MIC-128	OFF	YES (M=16)
0x04	‘100’	ENC	ON	NO (M = 0)
0x05	‘101’	ENC-MIC-32	ON	YES (M=4)
0x06	‘110’	ENC-MIC-64	ON	YES (M=8)
0x07	‘111’	ENC-MIC-128	ON	YES (M=16)

3.6.1.1.2 Key identifier sub-field

The key identifier sub-field consists of two bits that are used to identify the key used to protect the frame. The encoding for the key identifier sub-field shall be as listed in Table 170.

Table 170 Encoding for the key identifier sub-field

Key Identifier	Key Identifier Sub-Field (Table 80)	Description
0x00	‘00’	A link key.
0x01	‘01’	A Network key.
0x02	‘10’	A key-transport key.
0x03	‘11’	A key-load key.

3.6.1.1.3 Extended nonce sub-field

The extended nonce sub-field shall be set to 1 if the sender address field of the auxiliary header is present. Otherwise, it shall be set to 0.

3.6.1.2 Source Address Field

The source address field shall only be present when the extended nonce sub-field of the security control field is 1. When present, the source address field shall indicate the extended 64-bit address of the device responsible for securing the frame.

3.6.1.3 Counter Field

The counter field is used to provide for frame freshness and to prevent processing of duplicate frames.

3.6.1.4 Key Sequence Number Field

The key sequence number field shall only be present when the key identifier sub-field of the security control field is 1 (i.e., the Network key). When present, the key sequence number field shall indicate the key sequence number of the Network key used to secure the frame.

3.6.2 Security Parameters

This clause specifies the parameters used for the CCM* security operations.

3.6.2.1 CCM* Mode of Operation and Parameters

Applying security to a MAC, NWK, or APS frame on a particular security level corresponds to a particular instantiation of the AES-CCM* mode of operation as specified in section B.1.2. The AES-CCM* mode of operation is an extension of the AES-CCM mode that is used in the 802.15.4-2003 MAC specification and provides capabilities for authentication, encryption, or both.

The nonce shall be formatted as specified in sub-clause 3.6.2.2.

Table 169 gives the relationship between the security level subfield of the security control field (Table 80), the security level identifier, and the CCM* encryption/authentication properties used for these operations.

3.6.2.2 CCM* Nonce

The nonce input used for the CCM* encryption and authentication transformation and for the CCM* decryption and authentication checking transformation consists of data explicitly included in the frame and data that both devices can independently obtain. Figure 81 specifies the order and length of the subfields of the CCM* nonce. The nonce's security control and frame counter fields shall be the same as the auxiliary header's security control and frame counter fields (as defined in sub-clause 3.6.1) of the frame being processed. The nonce's source address field shall be set to the extended 64-bit MAC address of the device originating security protection of the frame. When the extended nonce sub-field of the auxiliary header's security control field is 1, the extended 64-bit MAC address of the device originating security protection of the frame shall correspond to the auxiliary header's source address field (as defined in sub-clause 3.6.1) of the frame being processed.

Figure 81 CCM* nonce

Octets: 8	4	1
Source address	Frame counter	Security control

3.6.3 Cryptographic Key Hierarchy

The link key established between two (or more) devices via one of the key-establishment schemes specified in sub-clause 3.5.2 (or transport-key commands specified in sub-clause 3.5.3) is used to determine related secret keys, including data keys, key-transport keys, and key-load keys. These keys are determined as follows:

1. *Key-Transport Key.* This key is the outcome of executing the specialized keyed hash function specified in clause B.1.5 under the link key with as input string the 1-octet string '0x00'.
2. *Key-Load Key.* This key is the outcome of executing the specialized keyed hash function specified in clause B.1.5 under the link key with as input string the 1-octet string '0x02'.
3. *Data Key.* This key is equal to the link key.

All keys derived from the link key shall share the associated frame counters. Also, all layers of ZigBee shall share the Network key and associated outgoing and incoming frame counters.

3.6.4 Implementation Guidelines (Informative)

This clause provides general guidelines that should be followed to ensure a secure implementation.

3.6.4.1 Random Number Generator

A ZigBee device implementing the key-establishment (i.e., see sub-clause 3.2.4.1) security service may need a strong method of random number generation. For example, when link keys are pre-installed (e.g., in the factory), a random number may not be needed.

In all cases that require random numbers, it is critical that the random numbers are not predictable or have enough entropy, so an attacker will not be able determine them by exhaustive search. The general recommendation is that the random number generation shall meet the random number tests specified in FIPS140-2 [B12]. There are methods for generation of random numbers:

1. Base the random number on random clocks and counters within the ZigBee hardware
2. Base the random number on random external events
3. Seed each ZigBee device with a good random number from an external source during production. This random number can then used as a seed to generate additional random numbers.

A combination of these methods can be used. Since the random number generation is likely integrated into the ZigBee IC, its design and hence the ultimate viability of any encryption/security scheme is left up to the IC manufacturers.

3.6.4.2 Security Implementation

It is very important security be well implemented and tested so that no “bugs” exist that an attacker can use to his advantage. It is also desirable that the security implementation does not need to be re-certified for every application. Security services should be implemented and tested by security experts and should not be re-implemented or modified for different applications.

3.6.4.3 Conformance

Conformance shall be defined by the profile inheriting from this specification. Correct implementation of selected cryptographic protocols should be verified as part of the ZigBee certification process. This verification shall include known value tests: an implementation must show that given particular parameters, it will correctly compute the corresponding results.

3.7 Functional Description

This subclause provides detailed descriptions of how the security services shall be used in a ZigBee network. A description of the ZigBee coordinator’s security initialization responsibilities is given in sub-clause 3.7.1. A brief description of the trust center application is given in sub-clause 3.7.2. Detailed security procedures are given in sub-clause 3.7.3.

3.7.1 ZigBee Coordinator

The coordinator shall configure the security level of the network by setting the *NwkSecurityLevel* attribute in the NWK layer PIB table. If the *NwkSecurityLevel* attribute is set to zero, the network will be unsecured, otherwise it will be secured.

The coordinator shall configure the address of the trust center by setting the AIB attribute *apsTrustCenterAddress*. The default value of this address is the coordinator’s address itself, otherwise, the coordinator may designate an alternate trust center.

3.7.2 Trust Center Application

The trust center application runs on a device trusted by devices within a ZigBee network to distribute keys for the purpose of network and end-to-end application configuration management. The trust center shall be configured to operate in either commercial or residential mode and may be used to help establish end-to-end application keys either by sending out link keys directly (i.e., key-escrow capability) or by sending out master keys.

3.7.2.1 Commercial Mode

The commercial mode of the trust center is designed for high-security commercial applications. In this mode, the trust center shall maintain a list of devices, master keys, link keys, and Network keys that it needs to control and enforce the policies of Network key updates and network admittance. In this mode, the memory required for the trust center grows with the number of devices in the network and the *nwkAllFresh* attribute in the NIB shall be set to TRUE.

3.7.2.2 Residential Mode

The residential mode of the trust center is designed for low-security residential applications. In this mode, the trust center may maintain a list of devices, master keys, or link keys with all the devices in the network; however, it shall maintain the Network key and controls policies of network admittance. In this mode, the memory required for the trust center does not grow with the number of devices in the network and the *nwkAllFresh* attribute in the NIB shall be set to FALSE.

3.7.3 Security Procedures

This subclause gives message sequence charts for joining a secured network, authenticating a newly joined device, updating the Network key, recovering the Network key, establishing end-to-end application keys, and leaving a secured network.

3.7.3.1 Joining a Secured Network

Figure 82 shows an example message sequence chart ensuing from when a joiner device communicates with a router device to join a secured network.

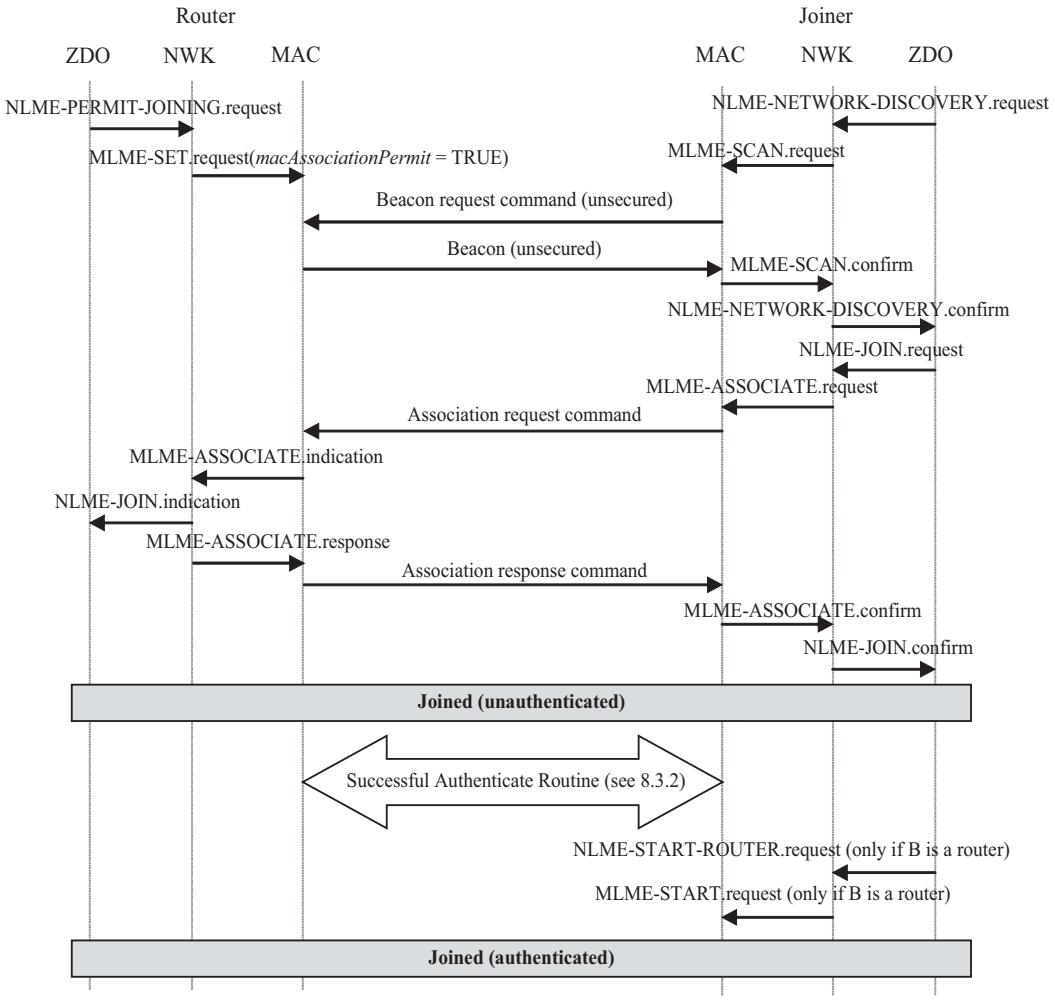


Figure 82 Example of joining a secured network

The joiner device may begin the join procedure by issuing an NLME-NETWORK-DISCOVERY.request primitive. This primitive will invoke an MLME-SCAN.request primitive which may cause the transmission of an unsecured beacon request frame (depending on whether the scan is an active or passive scan).

The joiner device receives beacons from nearby routers and the NWK layer will issue an NLME-NETWORK-DISCOVERY.confirm primitive. The *NetworkList* parameter of this primitive will indicate all of the nearby PANs along with their *nwkSecurityLevel* and *nwkSecureAllFrames nwkSecureAllFrames* attributes. In Figure 82, the shown router device has already been placed in a state such that its beacons have the "association permit" sub-field set to "1" (permit association).

The joiner device shall decide which PAN to join (e.g., based on the security attributes received in NLME-NETWORK-DISCOVERY.confirm primitive) and shall issue the NLME-JOIN.request primitive to join that PAN. If the joiner already has a Network key for this PAN, the *SecurityEnable* parameter for the NLME-JOIN.request primitive shall be set to TRUE; otherwise it shall be set to FALSE. As shown in Figure 82, the NLME-JOIN.request primitive causes an association request command to be sent to the router.

Upon receipt of the association request command, the router shall issue an MLME-ASSOCIATE.indication primitive with the *SecurityUse* parameter set to TRUE or FALSE, according to whether the association request command was secured or not. Next, the NWK layer will issue an NLME-JOIN.indication primitive to the router's ZDO. The router shall now know the joiner device's address and whether the Network key was used to secure the association request command. The router will also issue an MLME-ASSOCIATE.response primitive with the *SecurityEnable* parameter set to TRUE or FALSE, according to whether the association request command was secured or not, respectively. This primitive will cause an association response command to be sent to the joiner.

Upon receipt of the association response command, the joiner shall issue the NLME-JOIN.confirm primitive The joiner is now declared "joined, but unauthenticated" to the network. The authentication routine (see sub-clause 3.7.3.2) shall follow.

If the joiner is not a router, it is declared "joined and authenticated" immediately following the successful completion of the authentication routine.

If the joiner is a router, it is declared "joined and authenticated" only after the successful completion of the authentication routine followed by the initiation of routing operations. Routing operations shall be initiated by the joiner's ZDO issuing the NLME-START.request[201] primitive to cause the MLME-START.request primitive to be sent to the MAC layer of the joiner.

If the router refuses the joiner, its association response frame shall contain the association status field set to a value other than "0x00", and, after this parameter reaches the ZDO of the joiner in the NLME-JOIN.confirm primitive, the joiner shall not begin the authentication routine.

3.7.3.2 Authentication

Once a device joins a secured network and is declared "joined but unauthenticated", it must be authenticated as specified in this sub-clause.

3.7.3.2.1 Router operation

If the router is not the trust center, it shall begin the authentication procedure immediately after receipt of the NLME-JOIN.indication[202] primitive by issuing an APSME-UPDATE-DEVICE.request primitive with the *DestAddress* parameter set to the *apsTrustCenterAddress* in the AIB and the *DeviceAddress* parameter set to the address of the newly joined device. The *Status* parameter of this primitive shall be set to 0x00 (i.e.,

[201] CCB Comment #216
[202] CCB Comment #217

secured join) if the newly joined device secured the associate request command. Otherwise, the *Status* parameter shall be set to 0x01 (i.e., unsecured join).

If the router is the trust center, it shall begin the authentication procedure by simply operating as a trust center.

3.7.3.2.2 Trust center operation

The trust center role in the authentication procedure shall be activated upon receipt of an incoming update-device command or immediately after receipt of the NLME-JOIN.indication[203] primitive (in the case where the router is the trust center). The trust center behaves differently depending on at least five factors:

- Whether the trust center decides to allow the new device to join the network (e.g., the trust center is in a mode that allows new devices to join).
- Whether the trust center is operating in residential or commercial mode (see sub-clause 3.7.2.1 and sub-clause 3.7.2.2, respectively).
- If in residential mode, whether the device is joining unsecured or secured (i.e., as indicated by the *Status* sub-field of the update-device command).
- If in commercial mode, whether the trust center has a master key corresponding to the newly joined device.
- The *nwkSecureAllFrames* parameter of the NIB.

If, at any time during the authentication procedure, the trust center decides not to allow the new device to join the network (e.g., a policy decision or a failed key-establishment protocol), it shall take actions to remove the device from the network. If the trust center is not the router of the newly joined device, it shall remove the device from the network by issuing the APSME-REMOVE-DEVICE.request primitive with the *ParentAddress* parameter set to the address of the router originating the update-device command and the *ChildAddress* parameter set to the address of the joined (but unauthenticated) device. If the trust center is the router of the newly joined device, it shall remove the device from the network by issuing the NLME-LEAVE.request primitive with the *DeviceAddress* parameter set to the address of the joined (but unauthenticated) device.

3.7.3.2.2.1 Residential mode

After being activated for the authentication procedure the trust center shall send the device the active Network key by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the newly joined device, and the *KeyType* parameter to 0x01 (i.e., Network key).

If the joining device already has the Network key (i.e., the *Status* sub-field of the update-device command is 0x00), the *TransportKeyData* sub-parameters shall be set as follows: the *KeySeqNumber* sub-parameter shall be set to 0, the *NetworkKey* sub-parameter shall be set to all zeros, and the *UseParent* sub-parameter shall be set to FALSE.

Otherwise, the *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to Network key. The *UseParent* sub-parameter shall be set to FALSE if the trust center is the router; otherwise, the *UseParent* sub-parameter shall be set to TRUE and the *ParentAddress* sub-parameter shall be set to the address of the router originating the update-device command.

In the case of a joining device that is not preconfigured with a Network key, the issuance of this transport-key primitive will cause the Network key to be sent unsecured from the router to the newly joined device—

[203] CCB Comment #218

security is assumed to be present here via non-cryptographic means, such as only sending this key once, at low power, immediately after external input to both router and joiner, etc.

3.7.3.2.2.2 Commercial mode

After being activated for the authentication procedure, the trust center operation in commercial mode depends on if the device joining the network is preconfigured with a trust center master key.

If the trust center does not already share a master key with the newly joined device, it shall send the device a master key by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the newly joined device, and the *KeyType* parameter to 0x00 (i.e., trust center master key). The *TransportKeyData* sub-parameters shall be set as follows: the *TrustCenterMasterKey* sub-parameter shall be set to trust center master key, and the *ParentAddress* sub-parameter shall set to the address of the local device if the trust center is the router; otherwise, the *ParentAddress* sub-parameter shall set to the address of the router originating the update-device command. The issuance of this primitive will cause the master key to be sent unsecured from the router to the newly joined device—security is assumed to be present here via non-cryptographic means, such as only sending this key once, at low power, immediately after external input to both router and joiner, etc.

The trust center shall initiate the establishment of a link key by issuing the APSME-ESTABLISH-KEY.request primitive with the *ResponderAddress* parameter set to the address of the newly joined device and the *KeyEstablishmentMethod* set to 0x00 (i.e., SKKE). Additionally, if the *nwkSecureAllFrames* parameter of the NIB is FALSE or the trust center is the router, the *UseParent* parameter shall be set to FALSE; otherwise, the *UseParent* parameter shall be set to TRUE and the *ResponderParentAddress* parameter shall be set to the address of the router originating the update-device command.

Upon receipt of the corresponding APSME-ESTABLISH-KEY.confirm primitive with *Status* equal to 0x00 (i.e., success), the trust center shall send the new device the Network key by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the newly joined device, and the *KeyType* parameter to 0x01 (i.e., Network key). The *TransportKeyData* sub-parameters shall be set as follows. The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to Network key, and the *UseParent* sub-parameter shall be set to FALSE.

3.7.3.2.3 Joining device operation

After successfully associating to a secured network, the joining device shall participate in the authentication procedure described in this sub-clause. Following a successful authentication procedure, the joining device shall set the *nwkSecurityLevel* and *nwkSecureAllFrames* attributes in the NIB to the values indicated in the beacon from the router.

A joined and authenticated device in a secured network with *nwkSecureAllFrames* equal to TRUE shall always apply NWK layer security to outgoing (incoming) frames unless the frame is destined for (originated from) a newly joined but unauthenticated child. No such restrictions exist if *nwkSecureAllFrames* is equal to FALSE.

The joining device's participation in the authentication procedure depends on the state of the device. There are three possible initial states to consider:

- Preconfigured with a Network key (i.e., residential mode)
- Preconfigured with a trust center master key and address (i.e., commercial mode)
- Not preconfigured (i.e., undetermined mode – either residential or commercial mode)

In a secured network, if the device does not become authenticated within a preconfigured amount of time, it shall leave the network.

3.7.3.2.3.1 Preconfigured Network key

If the joining device was preconfigured with just a Network key (and the association was successful), it shall set the outgoing frame counter for this key to zero, and empty the incoming frame counter set for this key, and wait to receive a dummy (all zero) Network key from the trust center. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (i.e., the Network key), the joining device shall set the *apsTrustCenterAddress* attribute in its AIB to the *SrcAddress* parameter of the APSME-TRANSPORT-KEY.indication primitive. The joining device is now considered authenticated and shall enter the normal operating state for residential mode.

3.7.3.2.3.2 Preconfigured trust center key

If the joining device is preconfigured with a trust center master key and address (i.e., the *apsTrustCenterAddress* attribute in the AIB) it shall wait to establish a link key and receive a Network key from the trust center. Therefore, upon receipt of the APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the trust center's address and the *KeyEstablishmentMethod* parameter set to SKKE, the joining device shall respond with the APSME-ESTABLISH-KEY.response primitive with the *InitiatorAddress* parameter set to the trust center's address and the *Accept* parameter set to TRUE. After receipt of the APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the trust center's address and the *Status* parameter set to 0x00 (i.e., success), the joining device shall expect to receive the Network key. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with *SourceAddress* parameter set to the trust center's address, the *KeyType* parameter set to 0x01 (i.e., the Network key), the joining device shall use the data in the *TransportKeyData* parameter for configuring the Network key. The joining device is now considered authenticated and shall enter the normal operating state for commercial mode.

3.7.3.2.3.3 Not preconfigured

If the joining device is not preconfigured with a Network key nor a trust center master key and address (i.e., the *apsTrustCenterAddress* attribute in the AIB) it shall wait to receive either an unsecured trust center master key or a Network key. Implementers should note that transmission of an unsecured key represents a security risk and that if security is a concern, keys should be preconfigured – preferable via an out-of-band mechanism.

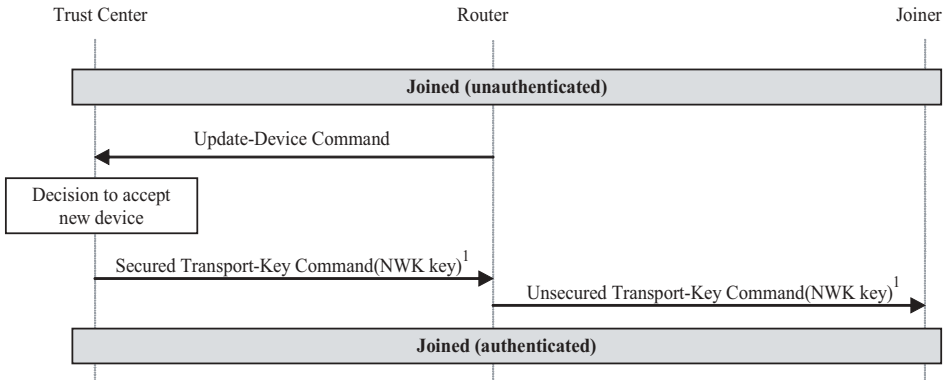
Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (i.e., the Network key), the joining device shall make the data in the *TransportKeyData* parameter its active Network key and shall set the *apsTrustCenterAddress* attribute in its AIB to the *SrcAddress* parameter of the APSME-TRANSPORT-KEY.indication primitive. The joining device is now considered authenticated and shall enter the normal operating state for residential mode.

Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x00 (i.e., the trust center master key), the joining device shall make its trust center master key the data in the *TransportKeyData* parameter and the *apsTrustCenterAddress* attribute in its AIB the *SrcAddress* parameter. Next, upon receipt of the APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the trust center's address and the *KeyEstablishmentMethod* parameter set to SKKE, the joining device shall respond with the APSME-ESTABLISH-KEY.response primitive with the *InitiatorAddress* parameter set to the trust center's address and the *Accept* parameter set to TRUE. After receipt of the APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the trust center's address and the *Status* parameter set to 0x00 (i.e., success), the joining device shall expect to receive the Network key. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with *SourceAddress* parameter set to the trust center's address, the *KeyType* parameter set to 0x01 (i.e., the Network key), the joining device shall use the data in the *TransportKeyData* parameter for configuring the Network key. The joining device is now considered authenticated and shall enter the normal operating state for commercial mode.

3.7.3.2.4 Message sequence charts

Figure 83 and Figure 84 give example message sequence charts for the authentication procedure when the router and trust center are separate devices operating in residential or commercial mode, respectively.

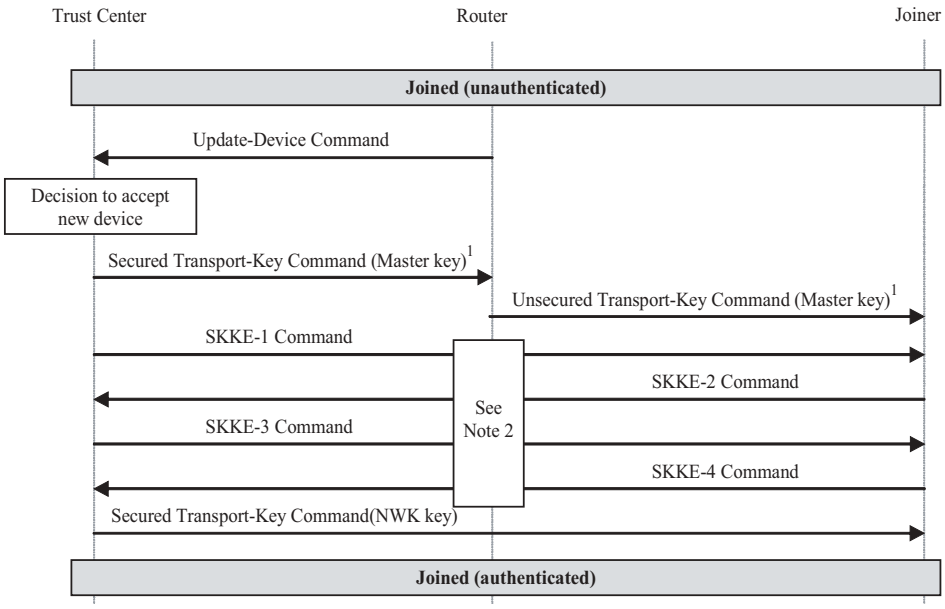
In Figure 83, the update-device and transport-key commands communicated between the trust center and the router shall be secured at the APS layer based on the Network key and, if the *nwkSecureAllFrames* NIB attribute is TRUE, also secured at the NWK layer with the Network key. The transport-key command sent from the router to joiner shall not be secured.



Note:
1. The trust center sends a dummy all-zero NWK key if the joiner securely joined using a preconfigured network key.

Figure 83 Example residential-mode authentication procedure

In Figure 84, the update-device and transport-key commands communicated between the trust center and the router shall be secured at the APS layer based on the trust center link key and, if the *nwkSecureAllFrames* NIB attribute is TRUE, also secured at the NWK layer with the Network key. The transport-key command sent from the router to joiner shall not be secured. The SKKE commands shall be sent using the router as a liaison when the *nwkSecureAllFrames* NIB attribute is TRUE, such that SKKE commands between the trust center and router shall be secured at the NWK layer with the Network key and commands between the router and joiner shall not be secured. Otherwise, the SKKE commands shall be unsecured between the trust center and joiner. The final transport-key communicated between the trust center and the joiner shall be secured at the APS layer based on the trust center link key and, if the *nwkSecureAllFrames* NIB attribute is TRUE, also secured at the NWK layer with the Network key.



Notes:
1. The trust center does not send a master key if it already shares one with the joiner device (i.e., the pre-configured situation)
2. SKKE commands shall be sent using the router as a liaison when the *nwkSecureAllFrame* NIB attribute is TRUE (i.e., these commands will be secured between the trust center and router at the NWK layer, but not between the router and joiner).

Figure 84 Example commercial-mode authentication procedure

3.7.3.3 Network Key Update

The trust center and network device shall follow the procedures described in this sub-clause when updating the Network key.

3.7.3.3.1 Trust center operation

When operating in residential mode, the trust center shall never update the network. This is a tradeoff to limit implementation complexity, at the cost of reduced security.

When operating in commercial mode, the trust center shall maintain a list of all devices in the network. To update the Network key, the trust center shall first send the new Network key to each device on this list and then ask each device to switch to this new key. The new Network key shall be sent to a device on the list by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the device on the list and the *KeyType* parameter set to 0x01 (i.e., Network key). The *TransportKeyData* sub-parameters shall be set as follows. The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to the Network key, and the *UseParent* sub-parameter shall be set to FALSE. If the sequence count for the previously distributed Network key is represented as *N*, then the sequence count for this new Network key shall be (*N*+1) mod 256. The trust center shall ask a device to switch to this new key by issuing the APSME-SWITCH-KEY.request primitive with the *DestAddress* parameter set to the address of the device on the list and the *KeySeqNumber* parameter set to the sequence count value for the updated Network key.

3.7.3.3.2 Network device operation

When in normal operating state for residential mode (i.e., a trust center master key is not present), a device shall not accept an updated Network key. Thus, in this mode, transport-key or a switch-key commands with the *KeyType* parameter set to 0x01 (i.e., Network key) shall be ignored.

When in the normal operating state for commercial mode (i.e., a trust center master key is present) and upon receipt of a APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (i.e., Network key), a device shall accept the *TransportKeyData* parameters as a Network key only if the *SrcAddress* parameter is the same as the trust center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB). If accepted and if the device is capable of storing an alternate Network key, the key and sequence number data contained in the *TransportKeyData* parameter shall replace the alternate Network key. Otherwise, the key and sequence number data contained in the *TransportKeyData* parameter shall replace the active Network key.

When in the normal operating state for commercial mode (i.e., a trust center master key is present) and upon receipt of a APSME-SWITCH-KEY.indication primitive, a device shall switch its active Network key to the one designated by the *KeySeqNumber* parameter only if the *SrcAddress* parameter is the same as the trust center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB).

3.7.3.3.3 Message sequence chart

An example of a successful Network key-update procedure for two devices is shown in Figure 85. In this example, the trust center sends the Network key with sequence number *N* to devices 1 and 2. In this example, device 1 is an FD capable of storing two Network keys, an active and alternate, and device 2 is an RFD that can store only a single Network key. Upon receipt of the transport-key command, device 1 replaces its alternate Network key with the new Network key; however device 2 must replace its active Network key with the new key. Next, upon receipt of the switch-key command, device 1 makes the new Network key the active Network key; however device 2 has just one active Network key, so it ignores this command.

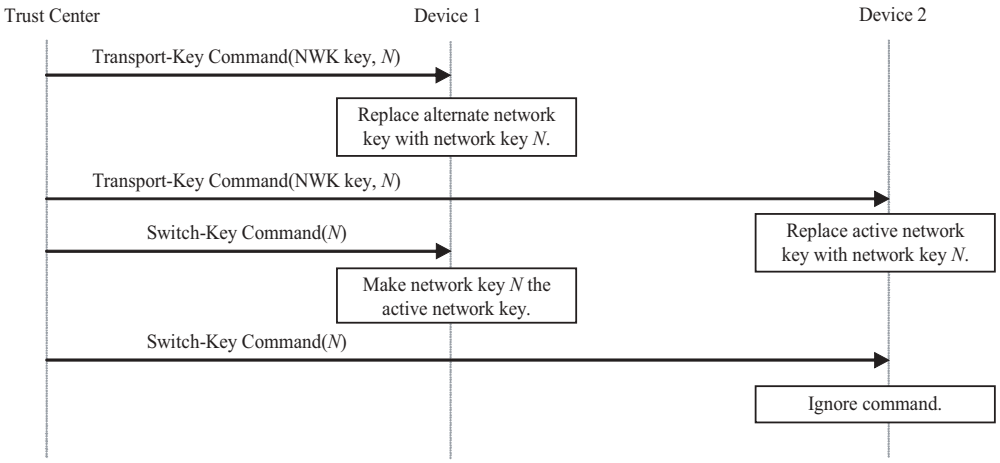


Figure 85 Example Network key-update procedure

3.7.3.4 Network Key Recovery

A network device and trust center shall follow the procedures described in this sub-clause when recovering the Network key.

3.7.3.4.1 Network device operation

When in the normal operating state for residential mode (i.e., a trust center master key is not present), a device shall not generate a request for an updated Network key.

When in the normal operating state for commercial mode (i.e., a trust center master key is present) a network device shall request the current Network key by issuing the APSME-REQUEST-KEY.request primitive

with the *DestAddress* parameter[204] set to the trust center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB), the *KeyType* parameter set to 0x01 (i.e., Network key), and the *PartnerAddress* parameter set to 0.

3.7.3.4.2 Trust Center operation

When operating in residential mode, the trust center shall ignore the receipt of APSME-REQUEST-KEY.indication primitives with the *KeyType* parameter set to 0x01[205] (i.e., Network key).

When operating in commercial mode and receipt of APSME-REQUEST-KEY.indication primitives with the *KeyType* parameter set to 0x01 (i.e., Network key), the trust center shall determine whether the device indicated by the *SrcAddress* parameter is present on its list of all device on the network. If the device is present on this list, the trust center shall issue the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the device requesting the key and the *KeyType* parameter set to 0x01 (i.e., Network key). The *TransportKeyData* sub-parameters shall be set as follows. The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to the Network key, and the *UseParent* sub-parameter shall be set to FALSE. Next, the trust center shall ask a device to switch to this new key by issuing the APSME-SWITCH-KEY.request primitive with the *DestAddress* parameter set to the address of the device that requested the key and the *KeySeqNumber* parameter set to the sequence count value for the updated Network key.

3.7.3.4.3 Message sequence chart

An example of a successful Network key-recovery procedure is shown in Figure 86. In this example, the network device requests the current Network key from the trust center. The trust center responds with current key and then tells the device to switch to this key.

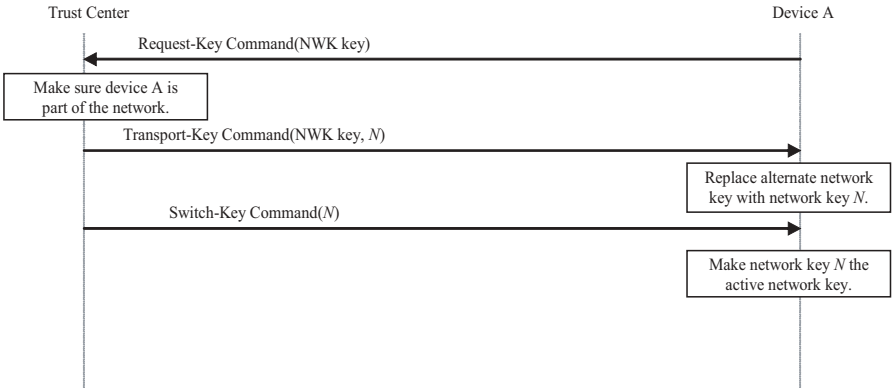


Figure 86 Example Network key-recovery procedure

3.7.3.5 End-to-End Application Key Establishment

An initiator device, a trust center, and a responder device shall follow the procedures described in this sub-clause when establishing a link key for purposes of end-to-end application security between initiator and responder devices.

3.7.3.5.1 Device operation

The initiator device shall begin the procedure to establish a link key with a responder device by issuing the APSME-REQUEST-KEY.request primitive. The *DstDevice* parameter shall be set to the address of its trust

[204] CCB Comment #219
[205] CCB Comment #220

center, the *KeyType* parameter shall be set to 0x02 (i.e., application key), and the *PartnerAddress* parameter shall be set to the address of the responder device.

3.7.3.5.1.1 Upon receipt of link key

Upon receipt of an APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x03 (i.e., application link key), a device may accept the *TransportKeyData* parameters as a link key with the device indicated by the *PartnerAddress* parameter only if the *SrcAddress* parameter is the same as the *apsTrustCenterAddress* attribute of the AIB. If accepted, the *DeviceKeyPairSet* attribute in AIB table will be updated. A key-pair descriptor in the AIB shall be created (or updated if already present), for the device indicated by the *PartnerAddress* parameter, by setting the *DeviceAddress* element to the *PartnerAddress* parameter, the *LinkKey* element to the link key from the *TransportKeyData* parameter, and the *OutgoingFrameCounter* and *IncomingFrameCounter* elements to 0.

3.7.3.5.1.2 Upon receipt of a master key

Upon receipt of an APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x02 (i.e., application master key), a device may accept the *TransportKeyData* parameters as a master key with the device indicated by the *PartnerAddress* sub-parameter only if the *SrcAddress* parameter is the same as the *apsTrustCenterAddress* attribute of the AIB. If accepted, the *DeviceKeyPairSet* attribute in AIB table will be updated. A key-pair descriptor shall be created (or updated if already present), for the device indicated by the *PartnerAddress* parameter, by setting the *DeviceAddress* element to the *PartnerAddress* parameter, the *MasterKey* element to the master key from the *TransportKeyData* parameter, and the *OutgoingFrameCounter* and *IncomingFrameCounter* elements to 0.

Next, if the *Initiator* sub-parameter of the *TransportKeyData* parameter of the APSME-TRANSPORT-KEY.indication primitive was TRUE, the device shall issue the APSME-ESTABLISH-KEY.request primitive. The *ResponderAddress*[206] parameter shall be set to the *PartnerAddress* sub-parameter of the *TransportKeyData* parameter, the *UseParent* parameter shall be set to FALSE, and the *KeyEstablishmentMethod* shall be set to 0x00 (i.e., SKKE).

Upon receipt of the APSME-ESTABLISH-KEY.indication primitive, the responder device shall be informed that the initiator device wishes to establish a link key. If the responder decides to establish a link key, it shall issue the APSME-ESTABLISH-KEY.response[207] primitive with the *InitiatorAddress* parameter set to the address of the initiator and the *Accept* parameter set to TRUE. Otherwise, it shall set the *Accept* parameter set to FALSE.

If the responder decided to set up a key with the initiator, the SKKE protocol will ensue and the APSME-ESTABLISH-KEY.confirm primitive will be issued to both the responder and initiator.

3.7.3.5.2 Trust center operation

Upon receipt of APSME-REQUEST-KEY.indication primitives with the *KeyType* parameter set to 0x02 (i.e., application key), the trust center behavior depends on if it has been configured to send out application link keys or master keys.

The trust center shall issue two APSME-TRANSPORT-KEY.request primitives. If configured to send out application link keys the *KeyType* parameter shall be set to 0x03 (i.e., application link key); otherwise, the *KeyType* parameter shall be set to 0x02 (i.e., application master key). The first primitive shall have the *DestAddress* parameter set to the address of the device requesting the key. The *TransportKeyData* sub-parameters shall be set as follows: the *PartnerAddress* sub-parameter shall be set to the *PartnerAddress* sub-parameter of the APSME-REQUEST-KEY.indication primitive's *TransportKeyData* parameter, the

[206] CCB Comment #221
[207] CCB Comment #222

Initiator sub-parameter shall be set to TRUE, and the *Key* sub-parameter shall be set to a new key *K* (a master or link key). The second primitive shall have the *DestAddress* parameter set to the *PartnerAddress* sub-parameter of the APSME-REQUEST-KEY.indication primitive's *TransportKeyData* parameter. The *TransportKeyData* sub-parameters shall be set as follows: the *PartnerAddress* sub-parameter shall be set to the address of the device requesting the key, the *Initiator* sub-parameter shall be set to FALSE, and the *Key* sub-parameter shall be set to *K*.

3.7.3.5.3 Message sequence chart

An example message sequence chart of the end-to-end application key establishment procedure is shown in Figure 87. The procedure begins with the transmission of the request-key command from the initiator to the trust center. Next, the trust center starts a time-out timer. For the duration of this timer (i.e., until it expires), the trust center shall discard any new request-key commands for this pair of devices unless they are from the initiator.

The trust center shall now send transport-key commands containing the application link or master key to the initiator and responder devices. Only the initiator's transport-key command will have the *Initiator* field set to 1 (i.e., TRUE), so if a master key was sent, only the initiator device will begin the key-establishment protocol by sending the SKKE-1 command. If the responder decides to accept establishing a key with the initiator, the SKKE protocol will progress via the exchange of the SKKE-2, SKKE-3, and SKKE-4 commands. Upon completion (or time-out), the status of the protocol is reported to the ZDO's of the initiator and responder devices. If successful, the initiator and responder will now share a link key and secure communications will be possible.

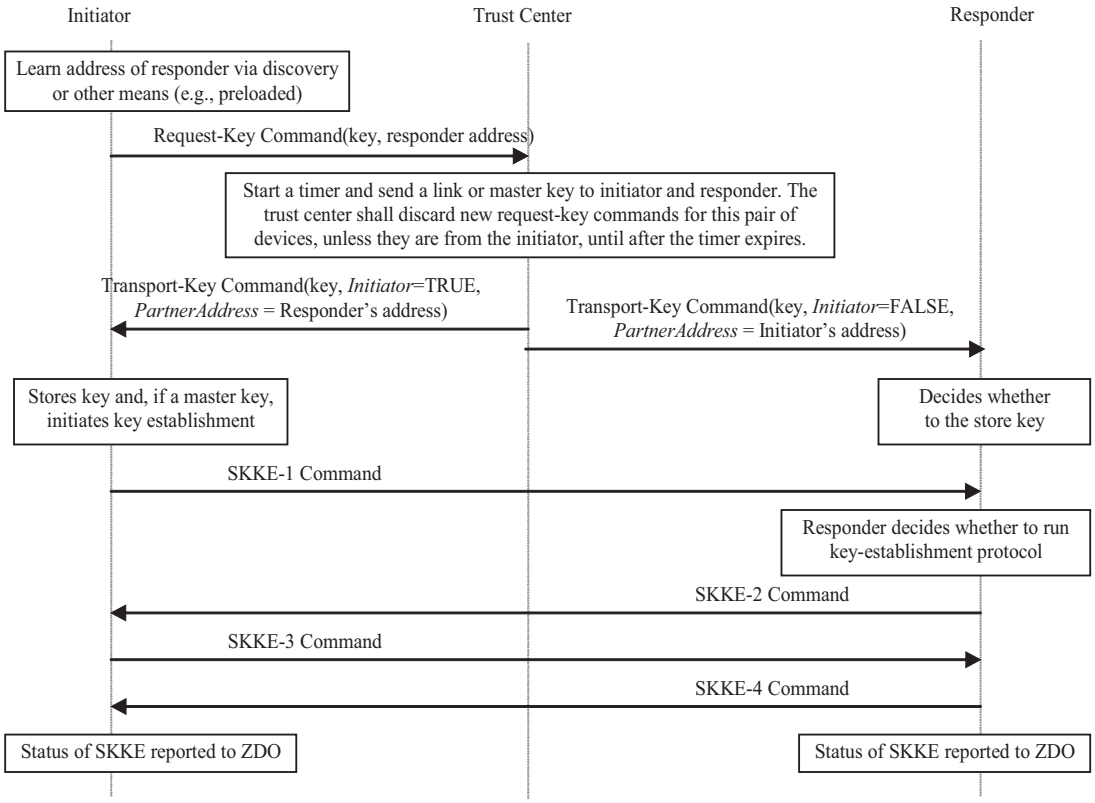


Figure 87 Example end-to-end application key establishment procedure

3.7.3.6 Network Leave

A device, its router, and the trust center shall follow the procedures described in this sub-clause when the device is to leave the network.

3.7.3.6.1 Trust center operation

If a trust center wants a device to leave and if the trust center is not the router for that device, the trust center shall send issue the APSME-REMOVE-DEVICE.request primitive, with the *ParentAddress* parameter set to the router's address and the *ChildAddress* parameter set to the address of the device it wishes to leave the network.

The trust center will also be informed of devices that leave the network. Upon receipt of an APSME-UPDATE-DEVICE.indication primitive with the *Status* parameter set to 0x02 (i.e., device left), the *DeviceAddress* parameter shall indicate the address of the device that left the network and the *SrcAddress* parameter shall indicate the address of parent of this device. If operating in commercial mode, the trust center shall delete the leaving device from its list of network devices.

3.7.3.6.2 Router operation

Routers are responsible for receiving remove-device commands and for sending update-device commands.

Upon receipt of an APSME-REMOVE-DEVICE.indication primitive, if the *SrcAddress* parameter is equal to the *apsTrustCenterAddress* attribute of the AIB, a router shall issue an NLME-LEAVE.request primitive with the *DeviceAddress* parameter the same as the *DeviceAddress* parameter of the APSME-REMOVE-DEVICE.indication primitive. The router shall ignore REMOVE-DEVICE.indication primitives with the *SrcAddress* parameter not equal to the *apsTrustCenterAddress* attribute of the AIB.

Upon receipt of an NLME-LEAVE.indication primitive with the *DeviceAddress* parameter set to one of its children, a router that is not also the trust center shall issue an APSME-UPDATE-DEVICE.request primitive with, the *DstAddress* parameter set to the address of the trust center, the *Status* parameter set to 0x02 (i.e., device left), and the *DeviceAddress* parameter set to the *DeviceAddress* parameter of the NLME-LEAVE.indication primitive. If the router is the trust center, it should simply operate as the trust center and shall not issue the APSME-UPDATE-DEVICE.request primitive (see sub-clause 3.7.3.6.1).

3.7.3.6.3 Leaving device operation

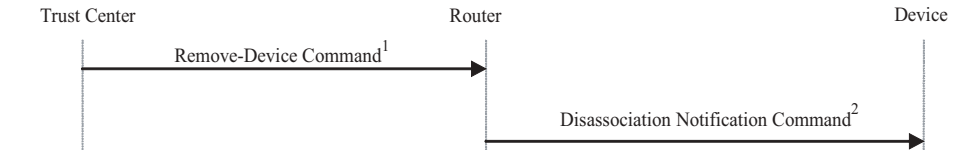
Devices are responsible for receiving and sending disassociation notification commands.

In a secured ZigBee network, disassociation notification commands shall be secured with the Network key and sent with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

In a secured ZigBee network, disassociation notification commands shall be received and processed only if secured with the Network key and received with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

3.7.3.6.4 Message sequence charts

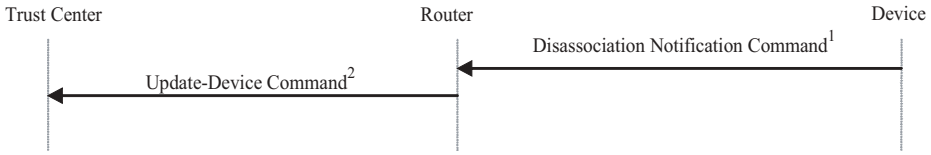
Figure 88 shows an example message sequence chart in which a trust center asks a router to remove one of its children from the network. If a trust center wants a device to leave and if the trust center is not the router for that device, the trust center shall send the router a remove-device command with the address of the device it wishes to leave the network. In a secure network, the remove-device command shall be secured with a link key if present; otherwise shall be secured with the Network key. Upon receipt of the remove-device command, a router shall send a disassociation notification command to the device to leave the network.



Note:
1. If a trust center wants a device to leave and if the trust center is not the router for that device, the trust center shall send the router a remove-device command with the address of the device it wishes to leave the network.
2. A router shall send a disassociation command to cause one of its children to leave the network.

Figure 88 Example remove-device procedure

Figure 89 shows an example message sequence chart whereby a device notifies its router that it is disassociating from the network. In this example, the device sends a disassociation notification command (secured with the Network key) to its router. The router then sends a device-update command to the trust center. In a secured network, the device-update command must be secured with the link key, if present, or the Network key.



Note:
1. A device leaving the network shall send a disassociation command to its router.
2. Upon receipt of a valid disassociation command, a router shall send an update-device command to the trust center to inform it that a device has left the network.

Figure 89 Example device-leave procedure

Annex A CCM* Mode of Operation

CCM* is a generic combined encryption and authentication block cipher mode. CCM* is only defined for use with block ciphers with a 128-bit block size, such as AES-128 [B8]. The CCM* ideas can easily be extended to other block sizes, but this will require further definitions.

The CCM* mode coincides with the original CCM mode specification [B20] for messages that require authentication and, possibly, encryption, but does also offer support for messages that require only encryption. As with the CCM mode, the CCM* mode requires only one key. The security proof for the CCM mode [B21], [B22] carries over to the CCM* mode described here. The design of the CCM* mode takes into account the results of [B23], thus allowing it to be securely used in implementation environments for which the use of variable-length authentication tags, rather than fixed-length authentication tags only, is beneficial.

Prerequisites: The following are the prerequisites for the operation of the generic CCM* mode:

1. A block-cipher encryption function *E* shall have been chosen, with a 128-bit block size. The length in bits of the keys used by the chosen encryption function is denoted by *keylen*.
2. A fixed representation of octets as binary strings shall have been chosen (e.g., most-significant-bit first order or least-significant-bit-first order).
3. The length *L* of the message length field, in octets, shall have been chosen. Valid values for *L* are the integers 2, 3,..., 8 (the value *L*=1 is reserved).
4. The length *M* of the authentication field, in octets, shall have been chosen. Valid values for *M* are the integers 0, 4, 6, 8, 10, 12, 14, and 16. (The value *M*=0 corresponds to disabling authenticity, since then the authentication field is the empty string.)

A.1 Notation and representation

Throughout this specification, the representation of integers as octet strings shall be fixed. All integers shall be represented as octet strings in most-significant-octet first order. This representation conforms to the conventions in Section 4.3 of ANSI X9.63-2001 [B7].

A.2 CCM* mode encryption and authentication transformation

The CCM* mode forward transformation involves the execution, in order, of an input transformation (A.2.1), an authentication transformation (A.2.2), and encryption transformation (A.2.3).

Input: The CCM* mode forward transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key sharing group member(s).
2. A nonce *N* of 15-*L* octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string *m* of length $l(m)$ octets, where $0 \leq l(m) < 2^{8L}$.
4. An octet string *a* of length $l(a)$ octets, where $0 \leq l(a) < 2^{64}$.

The nonce *N* shall encode the potential values for *M* such that one can uniquely determine from *N* the actually used value of *M*. The exact format of the nonce *N* is outside the scope of this specification and shall be determined and fixed by the actual implementation environment of the CCM* mode.

Note: The exact format of the nonce *N* is left to the application, to allow simplified hardware and software implementations in particular settings. Actual implementations of the CCM* mode may restrict the values of *M* that are allowed throughout the life-cycle of the encryption key *Key* to a strict subset of those allowed in the generic CCM* mode. If so, the format of the nonce *N* shall be such that one can uniquely determine from *N* the actually used value of *M* in that particular subset. In particular, if *M* is fixed and the value *M*=0 is not allowed, then there are no restrictions on *N*, in which case the CCM* mode reduces to the CCM mode.

A.2.1 Input transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

This step involves the following steps, in order:

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*, as follows:
   a) If *l(a)*=0, then *L(a)* is the empty string.
   b) If $0 < l(a) < 2^{16}-2^8$, then *L(a)* is the 2-octets encoding of *l(a)*.
   c) If $2^{16}-2^8 \leq l(a) < 2^{32}$, then *L(a)* is the right-concatenation of the octet 0xff, the octet 0xfe, and the 4-octets encoding of *l(a)*.
   d) If $2^{32} \leq l(a) < 2^{64}$, then *L(a)* is the right-concatenation of the octet 0xff, the octet 0xff, and the 8-octets encoding of *l(a)*.
2. Right-concatenate the octet string *L(a)* with the octet string *a* itself. Note that the resulting string contains *l(a)* and *a* encoded in a reversible manner.
3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.
4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16.
5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

$$AuthData = AddAuthData \,||\, PlaintextData. \quad (1)$$

A.2.2 Authentication transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field consisting of the 1-bit *Reserved* field, the 1-bit *Adata* field, and the 3-bit representations of the integers *M* and *L*, as follows:

$$Flags = Reserved \,||\, Adata \,||\, M \,||\, L. \quad (2)$$

   Here, the 1-bit *Reserved* field is reserved for future expansions and shall be set to '0'. The 1-bit *Adata* field is set to '0' if *l(a)*=0, and set to '1' if *l(a)*>0. The *L* field is the 3-bit representation of the integer *L*-1, in most-significant-bit-first order. The *M* field is the 3-bit representation of the integer (*M*-2)/2 if *M>0* and of the integer 0 if *M*=0, in most-significant-bit-first order.
2. Form the 16-octet B_0 field consisting of the 1-octet *Flags* field defined above, the 15-*L* octet nonce field *N*, and the *L*-octet representation of the length field *l(m)*, as follows:

   $B_0 = Flags \,||\, Nonce\ N \,||\, l(m)$.

3. Parse the message *AuthData* as $B_1 \| B_2 \| ... \| B_t$, where each message block B_i is a 16-octet string.

   The CBC-MAC value X_{t+1} is defined by

$$X_0 := 0^{128}; X_{i+1} := E(Key, X_i \oplus B_i) \text{ for } i=0, ..., t. \quad (3)$$

   Here, $E(K, x)$ is the cipher-text that results from encryption of the plaintext x using the established block-cipher encryption function E with key *Key*; the string 0^{128} is the 16-octet all-zero bit string.

   The authentication tag T is the result of omitting all but the leftmost M octets of the CBC-MAC value X_{n+1} thus computed.

A.2.3 Encryption transformation

The data *PlaintextData* that was established in sub-clause A.2.1 (step 4) and the authentication tag T that was established in sub-clause A.2.2 (step 3) shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet *Flags* field consisting of two 1-bit *Reserved* fields, and the 3-bit representations of the integers *0* and L, as follows:

$$Flags = Reserved \| Reserved \| 0 \| L. \quad (4)$$

   Here, the two 1-bit *Reserved* fields are reserved for future expansions and shall be set to '0'. The L field is the 3-bit representation of the integer L-1, in most-significant-bit-first order. The *'0'* field is the 3-bit representation of the integer 0, in most-significant-bit-first order.

   Define the 16-octet A_i field consisting of the 1-octet *Flags* field defined above, the 15-L octet nonce field N, and the L-octet representation of the integer i, as follows:

$$A_i = Flags \| Nonce\ N \| Counter\ i, \text{ for } i=0, 1, 2, \ldots \quad (5)$$

   Note that this definition ensures that all the A_i fields are distinct from the B_0 fields that are actually used, as those have a *Flags* field with a non-zero encoding of M in the positions where all A_i fields have an all-zero encoding of the integer 0 (see sub-clause A.2.2, step 2).

   Parse the message *PlaintextData* as $M_1 \| ... \| M_t$, where each message block M_i is a 16-octet string.

   The ciphertext blocks $C_1, ..., C_t$ are defined by

$$C_i := E(Key, A_i) \oplus M_i \text{ for } i=1, 2, ..., t. \quad (6)$$

   The string *Ciphertext* is the result of omitting all but the leftmost *l(m)* octets of the string $C_1 \| ... \| C_t$.

   Define the 16-octet encryption block S_0 by

$$S_0 := E(Key, A_0). \quad (7)$$

2. The encrypted authentication tag U is the result of XOR-ing the string consisting of the leftmost M octets of S_0 and the authentication tag T.

   Output: If any of the above operations has failed, then output 'invalid'. Otherwise, output the right-concatenation of the encrypted message *Ciphertext* and the encrypted authentication tag U.

A.3 CCM* mode decryption and authentication checking transformation

Input: The CCM* inverse transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key-sharing group member(s).
2. A nonce *N* of 15-*L* octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string *c* of length *l(c)* octets, where $0 \leq l(c)\text{-}M < 2^{8L}$.
4. An octet string *a* of length *l(a)* octets, where $0 \leq l(a) < 2^{64}$.

A.3.1 Decryption transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the right-most string *U* is an *M*-octet string. If this operation fails, output 'invalid' and stop. *U* is the purported encrypted authentication tag. Note that the leftmost string *C* has length *l(c)-M* octets.
2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.
3. Use the encryption transformation in sub-clause A.2.3, with as inputs the data *CipherTextData* and the tag *U*.
4. Parse the output string resulting from applying this transformation as *m* || *T*, where the right-most string *T* is an *M*-octet string. *T* is the purported authentication tag. Note that the leftmost string *m* has length *l(c)-M* octets.

A.3.2 Authentication checking transformation

The authentication checking transformation involves the following steps:

1. Form the message *AuthData* using the input transformation in sub-clause A.2.1, with as inputs the string *a* and the octet string *m* that was established in sub-clause A.3.1 (step 4).
2. Use the authentication transformation in sub-clause A.2.2, with as input the message *AuthData*.
3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in sub-clause A.3.1 (step 4). If *MACTag=T*, output 'valid'; otherwise, output 'invalid' and stop.

   Output: If any of the above verifications has failed, then output 'invalid' and reject the octet string *m*. Otherwise, accept the octet string *m* and accept one of the key sharing group member(s) as the source of *m*.

A.4 Restrictions

All implementations shall limit the total amount of data that is encrypted with a single key. The CCM* encryption transformation shall invoke not more than 2^{61} block-cipher encryption function operations in total, both for the CBC-MAC and for the CTR encryption operations.

At CCM* decryption, one shall verify the (truncated) CBC-MAC before releasing any information, such as, e.g., plaintext. If the CBC-MAC verification fails, only the fact that the CBC-MAC verification failed shall be exposed; all other information shall be destroyed.

Annex B Security Building Blocks

This annex specifies the cryptographic primitives and mechanisms that are used to implement the security protocols in this standard.

B.1 Symmetric-key cryptographic building blocks

The following symmetric-key cryptographic primitives and data elements are defined for use with all security-processing operations specified in this standard.

B.1.1 Block-cipher

The block-cipher used in this specification shall be the Advanced Encryption Standard AES-128, as specified in FIPS Pub 197 [B8]. This block-cipher has a key size *keylen* that is equal to the block size, in bits, i.e., *keylen*=128.

B.1.2 Mode of operation

The block-cipher mode of operation used in this specification shall be the CCM* mode of operation, as specified in Annex A, with the following instantiations:

1. Each entity shall use the block-cipher *E* as specified in sub-clause B.1.1;
2. All octets shall be represented as specified in section "Preface";
3. The parameter *L* shall have the integer value 2;
4. The parameter *M* shall have one of the following integer values: 0, 4, 8, or 16.

B.1.3 Cryptographic hash function

The cryptographic hash function used in this specification shall be the block-cipher based cryptographic hash function specified in clause B.6, with the following instantiations:

1. Each entity shall use the block-cipher *E* as specified in section sub-clause B.1.1;
2. All integers and octets shall be represented as specified in section "Preface".

The Matyas-Meyer-Oseas hash function (specified in clause B.6) has a message digest size *hashlen* that is equal to the block size, in bits, of the established block-cipher.

B.1.4 Keyed hash function for message authentication

The keyed hash message authentication code (HMAC) used in this specification shall be HMAC, as specified in the FIPS Pub 198 [B9], with the following instantiations:

1. Each entity shall use the cryptographic hash *H* function as specified in sub-clause B.1.3;
2. The block size *B* shall have the integer value 16 (this block size specifies the length of the data integrity key, in bytes, that is used by the keyed hash function, i.e., it uses a 128-bit data integrity key);
3. The output size *HMAClen* of the HMAC function shall have the same integer value as the message digest parameter *hashlen* as specified in sub-clause B.1.3.

B.1.5 Specialized keyed hash function for message authentication

The specialized[208] keyed hash message authentication code used in this specification shall be the keyed hash message authentication code, as specified in sub-clause B.1.4.

B.1.6 Challenge domain parameters

The challenge domain parameters used in the specification shall be as specified in sub-clause B.3.1, with the following instantiation: (*minchallengelen, maxchallengelen*)=(128,128).

All challenges shall be validated using the challenge validation primitive as specified in clause B.4.

B.2 Key Agreement Schemes

B.2.1 Symmetric-key key agreement scheme

The symmetric-key key agreement protocols in this standard shall use the full symmetric-key with key confirmation scheme as specified in clause B.7, with the following instantiations:

1. Each entity shall be identified as specified in "Preface";
2. Each entity shall use the HMAC-scheme as specified in sub-clause B.1.4;
3. Each entity shall use the specialized HMAC-scheme as specified in sub-clause B.1.5;
4. Each entity shall use the cryptographic hash function as specified in sub-clause B.1.3.,
5. The parameter *keydatalen* shall have the same integer value as the key size parameter *keylen* as specified in sub-clause B.1.1;
6. The parameter *SharedData* shall be the empty string; parameter *shareddatalen* shall have the integer value 0;
7. The optional parameters $Text_1$ and $Text_2$ as specified in sub-clause B.7.1 and sub-clause B.7.2 shall both be the empty string.
8. Each entity shall use the challenge domain parameters as specified in sub-clause B.1.6.
9. All octets shall be represented as specified in section "Preface".

B.3 Challenge Domain Parameter Generation and Validation

This section specifies the primitives that shall be used to generate and validate challenge domain parameters.

Challenge domain parameters impose constraints on the length(s) of bit challenges a scheme expects. As such, this determine a bound on the entropy of challenges and, thereby, on the security of the cryptographic schemes in which these challenges are used. In most schemes, the challenge domain parameters will be such that only challenges of a fixed length will be accepted (e.g., 128-bit challenges). However, one may define the challenge domain parameters such that challenges of varying length might be accepted. The latter is useful in contexts where entities that wish to engage in cryptographic schemes might have a bad random

[208]This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

number generator on-board. Allowing both entities that engage in a scheme to contribute sufficiently long inputs enables each of these to contribute sufficient entropy to the scheme at hand.

In this standard, challenge domain parameters will be shared by a number of entities using a scheme of the standard. The challenge domain parameters may be public; the security of the system does not rely on these parameters being secret.

B.3.1 Challenge Domain Parameter Generation

Challenge domain parameters shall be generated using the following routine.

Input: This routine does not take any input.

Actions: The following actions are taken:

1. Choose two nonnegative integers minchallengelen and maxchallengelen, such that minchallengelen ≤ maxchallengelen.

Output: Challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*).

B.3.2 Challenge Domain Parameter Verification

Challenge domain parameters shall be verified using the following routine.

Input: Purported set of challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*).

Actions: The following checks are made:

1. Check that *minchallengelen* and *maxchallengelen* are nonnegative integers.
2. Check that *minchallengelen* ≤ *maxchallengelen*.

Output: If any of the above verifications has failed, then output ‘invalid’ and reject the challenge domain parameters. Otherwise, output ‘valid’ and accept the challenge domain parameters.

B.4 Challenge Validation Primitive

Challenge validation refers to the process of checking the length properties of a challenge. It is used to check whether a challenge to be used by a scheme in the standard has sufficient length (e.g., messages that are too short are discarded, due to insufficient entropy).

The challenge validation primitive is used in clause B.7.

Input: The input of the validation transformation is a valid set of challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*), together with the bit string *Challenge*.

Actions: The following actions are taken:

1. Compute the bit-length *challengelen* of the bit string *Challenge*.
2. Verify that *challengelen* ∈ [*minchallengelen*, *maxchallengelen*]. (That is, verify that the challenge has an appropriate length.)

Output: If the above verification fails, then output ‘invalid’ and reject the challenge. Otherwise, output ‘valid’ and accept the challenge.

B.5 Secret Key Generation (SKG) Primitive

This section specifies the SKG primitive that shall be used by the symmetric-key key agreement schemes specified in this standard.

This primitive derives a shared secret value from a challenge owned by an entity U_1 and a challenge owned by an entity U_2 when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

Prerequisites: The following are the prerequisites for the use of the SKG primitive:

1. Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity U_1's identifier will be denoted by the bit string U_1. Entity U_2's identifier will be denoted by the bit string U_2.
2. A specialized[209] MAC scheme shall have been chosen, with tagging transformation as specified in Section 5.7.1 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the specialized MAC scheme is denoted by *mackeylen*.

Input: The SKG primitive takes as input:

1. A bit string *MACKey* of length *mackeylen* bits to be used as the key of the established specialized MAC scheme.
2. A bit string QEU_1 owned by U_1.
3. A bit string QEU_2 owned by U_2.

Actions: The following actions are taken:

1. Form the bit string consisting of U_1's identifier, U_2's identifier, the bit string QEU_1 corresponding to U_1's challenge, and the bit string QEU_2 corresponding to QEU_2's challenge:

$$MacData = U_1 \| U_2 \| QEU_1 \| QEU_2. \quad (8)$$

2. Calculate the tag *MacTag* for *MacData* under the key *MacKey* using the tagging transformation of the established specialized MAC scheme:

$$MacTag = MAC_{MacKey}(MacData). \quad (9)$$

3. If the tagging transformation outputs 'invalid', output 'invalid' and stop.
4. Set *Z=MacTag*.

Output: The bit string *Z* as the shared secret value.

[209]This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

B.6 Block-Cipher-Based Cryptographic Hash Function

This section specifies the Matyas-Meyer-Oseas hash function, a cryptographic hash function based on block-ciphers. We define this hash function for block-ciphers with a key size that is equal to the block size, such as AES-128, and with a particular choice for the fixed initialization vector *IV* (we take *IV*=0). For a more general definition of the Matyas-Meyer-Oseas hash function, we refer to Section 9.4.1 of [B18].

Prerequisites: The following are the prerequisites for the operation of Matyas-Meyer-Oseas hash function:

1. A block-cipher encryption function *E* shall have been chosen, with a key size that is equal to the block size. The Matyas-Meyer-Oseas hash function has a message digest size that is equal to the block size of the established encryption function. It operates on bit strings of length less than 2^n, where *n* is the block size, in octets, of the established block-cipher.
2. A fixed representation of integers as binary strings or octet strings shall have been chosen.

Input: The input to the Matyas-Meyer-Oseas hash function is as follows:

1. A bit string *M* of length *l* bits, where $0 \leq l < 2^n$.

Actions: The hash value shall be derived as follows:

1. Pad the message *M* according to the following method:
   a) Right-concatenate to the message *M* the binary consisting of the bit '1' followed by *k* '0' bits, where *k* is the smallest non-negative solution to the equation

$$l+1+k \equiv 7n \ (\textbf{mod}\ 8n). \tag{10}$$

   b) Form the padded message *M'* by right-concatenating to the resulting string the *n*-bit string that is equal to the binary representation of the integer *l*.
2. Parse the padded message *M'* as $M_1 \,||\, M_2 || \ldots ||\, M_t$ where each message block M_i is an *n*-octet string.
3. The output $Hash_t$ is defined by

$$Hash_0 = 0^{8n};\ Hash_j = E(Hash_{j-1}, M_j) \oplus M_j \text{ for } j=1,\ldots,t. \tag{11}$$

   Here, $E(K, x)$ is the ciphertext that results from encryption of the plaintext *x*, using the established block-cipher encryption function E with key K; the string 0^{8n} is the *n*-octet all-zero bit string.

Output: The bit string $Hash_t$ as the hash value.

Note that the cryptographic hash function operates on bit strength of length less than 2^n bits, where *n* is the block size (or key size) of the established block cipher, in bytes. For example, the Matyas-Meyer-Oseas hash function with AES-128 operates on bit strings of length less than 2^{16} bits. It is assumed that all hash function calls are on bit strings of length less than 2^n bits. Any scheme attempting to call the hash function on a bit string exceeding 2^n bits shall output 'invalid' and stop.

B.7 Symmetric-Key Authenticated Key Agreement Scheme

This section specifies the full symmetric-key key agreement with key confirmation scheme. A MAC scheme is used to provide key confirmation.

Figure 90 illustrates the messaging involved in the use of the full symmetric-key key agreement with key confirmation scheme.

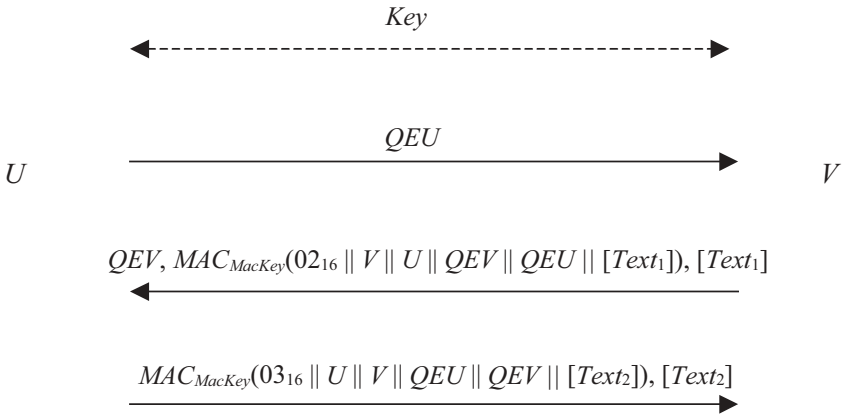


Figure 90 Symmetric-Key Authenticated Key Agreement Scheme

The scheme is ‘asymmetric’ so two transformations are specified. *U* uses the transformation specified in sub-clause B.7.1 to agree on keying data with *V* if *U* is the protocol’s initiator, and *V* uses the transformation specified in sub-clause B.7.2 to agree on keying data with *U* if *V* is the protocol’s responder.

The essential difference between the role of the initiator and the role of the responder is merely that the initiator sends the first pass of the exchange.

If *U* executes the initiator transformation, and *V* executes the responder transformation with the shared secret keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system’s challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*).
2. Each entity shall have access to a bit string *Key* of length *keylen* bits to be used as the key. Each party shall have evidence that access to this key is restricted to the entity itself and the other entity involved in the symmetric-key authenticated key agreement scheme.
3. Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*’s identifier will be denoted by the bit string *U*. Entity *V*’s identifier will be denoted by the bit string *V.*
4. Each entity shall have decided which MAC scheme to use as specified in Section 5.7 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the chosen MAC scheme is denoted by *mackeylen*.
5. A cryptographic hash function shall have been chosen for use with the key derivation function.
6. A specialized[210] MAC scheme shall have been chosen for use with the secret key generation primitive with tagging transformation as specified in Section 5.7.1 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the specialized MAC scheme is denoted by *keylen*.
7. A fixed representation of octets as binary strings shall have been chosen. (e.g., most-significant-bit-first order or least-significant-bit-first order).

[210]This refers to a MAC function with the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]. Specialized MAC functions allow key derivation in contexts where unilateral key control is undesirable.

B.7.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator. *U* shall obtain an authentic copy of *V*'s identifier and an authentic copy of the static secret key *Key* shared with *V.*

Input: The input to the initiator transformation is:

1. An integer *keydatalen* that is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits that consists of some data shared by *U* and *V.*
3. (Optional) A bit string $Text_2$ that consists of some additional data to be provided from *U* to *V.*

Ingredients: The initiator transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive in sub-clause B.3.2, the SKG primitive in sub-clause B.5, the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7], and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Keying data shall be derived as follows:

1. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEU* for the challenge domain parameters *D.* Send *QEU* to *V.*
2. Then receive from *V* a challenge *QEV'* purportedly owned by *V.* If this value is not received, output 'invalid' and stop.
3. Receive from *V* an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.
4. Verify that *QEV'* is a valid challenge for the challenge domain parameters *D* as specified in section sub-clause B.3.2. If the validation primitive rejects the challenge, output 'invalid' and stop.
5. Use the SKG primitive in clause B.5 to derive a shared secret bit string *Z* from the challenges Q_1=*QEU* owned by *U* and Q_2=*QEV'* owned by *V,* using as key the shared key *Key.* If the SKG primitive outputs 'invalid', output 'invalid' and stop.
6. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the established hash function to derive keying data *KKeyData* of length *mackeylen*+*keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
7. Parse the leftmost *mackeylen* bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
8. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*', the bit string *QEU*, and if present $Text_1$:

$$MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]. \quad (12)$$

9. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
10. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s challenge, the bit string *QEV*' corresponding to *V*'s challenge, and optionally a bit string $Text_2$:

$$MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]. \quad (13)$$

11. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.1 of ANSI X9.63-2001 [B7]:

$$MacTag_2 = MAC_{MacKey}(MacData_2). \quad (14)$$

12. If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and, if present, $Text_2$ to *V*.

Output: If any of the above verifications has failed, then output 'invalid' and reject the bit strings *KeyData* and $Text_1$. Otherwise, output 'valid', accept the bit string *KeyData* as the keying data of length *keydatalen* bits shared with *V* and accept *V* as the source of the bit string $Text_1$ (if present).

B.7.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder. *V* shall obtain an authentic copy of *U*'s identifier and an authentic copy of the static secret key *Key* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU*' purportedly owned by *U*.
2. An integer *keydatalen* that is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits that consists of some data shared by *U* and *V*.
4. (Optional) A bit string $Text_1$ that consists of some additional data to be provided from *V* to *U*.

Ingredients: The responder transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive in sub-clause B.3.2, the SKG primitive in clause B.5, the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7], and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Keying data shall be derived as follows:

1. Verify that *QEU*' is a valid challenge for the challenge domain parameters *D* as specified in sub-clause B.3.2. If the validation primitive rejects the challenge, output 'invalid' and stop.
2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEV* for the challenge domain parameters *D*. Send to *U* the challenge *QEV*.
3. Use the SKG primitive in clause B.5 to derive a shared secret bit string *Z* from the challenges Q_1=*QEU'* owned by *U* and Q_2=*QEV* owned by *V*, using as key the shared key *Key*. If the SKG primitive outputs 'invalid', output 'invalid' and stop.
4. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the established hash function to derive keying data *KKeyData* of length *mackeylen+keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
5. Parse the leftmost *mackeylen* bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
6. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*, the bit string *QEU*', and, optionally, a bit string $Text_1$:

$$MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]. \quad (15)$$

7. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7 of ANSI X9.63-2001 [B7]:

$$MacTag_1 = MAC_{MacKey}(MacData_1). \quad (16)$$

   If the tagging transformation outputs ‘invalid’, output ‘invalid’ and stop. Send to *U*, if present the bit string $Text_1$, and $MacTag_1$.

8. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$’. If this data is not received, output ‘invalid’ and stop.

9. Form the bit string consisting of the octet 03_{16}, *U*’s identifier, *V*’s identifier, the bit string *QEU*’ corresponding to *U*’s purported challenge, the bit string *QEV* corresponding to *V*’s challenge, and the bit string $Text_2$ (if present):

$$MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]. \quad (17)$$

10. Verify that $MacTag_2$’ is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7 ANSI X9.63-2001 [B7]. If the tag checking transformation outputs ‘invalid’, output ‘invalid’ and stop.

Output: If any of the above verifications has failed, then output ‘invalid’ and reject the bit strings *KeyData* and $Text_2$. Otherwise, output ‘valid’, accept the bit string *KeyData* as the keying data of length *keydatalen* bits shared with *U* and accept *U* as the source of the bit string $Text_2$ (if present).

Annex C Test Vectors for Cryptographic Building Blocks

This annex provides sample test vectors for the ZigBee community, aimed at assisting in building interoperable security implementations. The sample test vectors are provided as is, pending independent validation.

C.1 Data Conversions

For test vectors, see Appendix J1 of ANSI X9.63-2001 [B7].

C.2 AES Block Cipher

This annex provides sample test vectors for the block-cipher specified in sub-clause B.1.1.

For test vectors, see FIPS Pub 197 [B8].

C.3 CCM* Mode Encryption and Authentication Transformation

This annex provides sample test vectors for the mode of operation as specified in sub-clause B.1.2.

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF. (18)

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = A0 A1 A2 A3 A4 A5 A6 A7 || 03 02 01 00 || 06. (19)

3. The octet string *m* of length *l(m)*=23 octets to be used:

   m = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E. (20)

4. The octet string *a* of length *l(a)*=8 octets to be used:

   a = 00 01 02 03 04 05 06 07. (21)

C.3.1 Input Transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*:

L(a) = 00 08.

2. Right-concatenate the octet string *L(a)* and the octet string *a* itself:

   L(a) || *a* = 00 08 || 00 01 02 03 04 05 06 07.

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16:

   AddAuthData = 00 08 || 00 01 02 03 04 05 06 07 || 00 00 00 00 00 00.

4. Form the padded message *PlaintextData* by right-concatenating the octet string m with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16:

   PlaintextData = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 ||

   18 19 1A 1B 1C 1D 1E || 00 00 00 00 00 00 00 00 00.

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   AuthData = 00 08 00 01 02 03 04 05 06 07 00 00 00 00 00 00 ||

   08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17

   18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00.

C.3.2 Authentication Transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field as follows:

   Flags = 59.

2. Form the 16-octet B_0 field as follows:

   B_0 = 59 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 17.

3. Parse the message *AuthData* as B_1 || B_2 ||B_3, where each message block B_i is a 16-octet string.

4. The CBC-MAC value X_4 is computed as follows:

i	B_i	X_i
0	59 A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 00 17	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1	00 08 00 01 02 03 04 05 06 07 00 00 00 00 00 00	F7 74 D1 6E A7 2D C0 B3 E4 5E 36 CA 8F 24 3B 1A
2	08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17	90 2E 72 58 AE 5A 4B 5D 85 7A 25 19 F3 C7 3A B3
3	18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00	5A B2 C8 6E 3E DA 23 D2 7C 49 7D DF 49 BB B4 09
4	æ	B9 D7 89 67 04 BC FA 20 B2 10 36 74 45 F9 83 D6

The authentication tag *T* is the result of omitting all but the leftmost *M*=8 octets of the CBC-MAC value X_4:

T = B9 D7 89 67 04 BC FA 20.

C.3.3 Encryption Transformation

The data *PlaintextData* shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet Flags field as follows:

   Flags = 01.

2. Define the 16-octet A_i field as follows:

i	A_i
0	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 00
1	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 01
2	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 02

3. Parse the message *PlaintextData* as $M_1 \| M_2$, where each message block M_i is a 16-octet string.

4. The ciphertext blocks C_1, C_2 are computed as follows:

i	AES(Key,A_i)	$C_i = AES(Key,A_i) \oplus M_i$
1	12 5C A9 61 B7 61 6F 02 16 7A 21 66 70 89 F9 07	1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10
2	CC 7F 54 D1 C4 49 B6 35 46 21 46 03 AA C6 2A 17	D4 66 4E CA D8 54 A8 35 46 21 46 03 AA C6 2A 17

5. The string *Ciphertext* is the result of omitting all but the leftmost *l(m)*=23 octets of the string $C_1 \| C_2$:

   CipherText = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8.

6. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$ = B3 5E D5 A6 DC 43 6E 49 D6 17 2F 54 77 EB B4 39.

7. The encrypted authentication tag *U* is the result of XOR-ing the string consisting of the leftmost *M*=8 octets of S_0 and the authentication tag *T*:

   U = 0A 89 5C C1 D8 FF 94 69.

Output: the right-concatenation *c* of the encrypted message *Ciphertext* and the encrypted authentication tag *U*:

c = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 ||

0A 89 5C C1 D8 FF 94 69.

C.4 CCM* Mode Decryption and Authentication Checking Transformation

This annex provides sample test vectors for the inverse of the mode of operation as specified in sub-clause B.1.2.

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the inverse mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = A0 A1 A2 A3 A4 A5 A6 A7 || 03 02 01 00 || 06.

3. The octet string *c* of length *l*(*c*)=31 octets to be used:

   c = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 || 0A 89 5C C1 D8 FF 94 69.

4. The octet string *a* of length *l*(*a*)=8 octets to be used:

   a = 00 01 02 03 04 05 06 07.

C.4.1 Decryption Transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the right-most string *U* is an *M*-octet string:

   C = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8;

   U = 0A 89 5C C1 D8 FF 94 69.

2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.

   CipherTextData = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 || 00 00 00 00 00 00 00 00.

3. Form the 1-octet *Flags* field as follows:

   Flags = 01.

4. Define the 16-octet A_i field as follows:

i	A_i
0	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 00
1	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 01
2	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 02

5. Parse the message *CiphertextData* as C_1 ||C_2, where each message block C_i is a 16-octet string.

6. The ciphertext blocks P_1, P_2 are computed as follows:

i	AES(Key,A_i)	$P_i = AES(Key,A_i) \oplus C_i$
1	12 5C A9 61 B7 61 6F 02 16 7A 21 66 70 89 F9 07	08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
2	CC 7F 54 D1 C4 49 B6 35 46 21 46 03 AA C6 2A 17	18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00

7. The octet string *m* is the result of omitting all but the leftmost *l(m)*=23 octets of the string $P_1 \| P_2$:

   m = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 || 18 19 1A 1B 1C 1D 1E.

8. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$= B3 5E D5 A6 DC 43 6E 49 D6 17 2F 54 77 EB B4 39.

9. The purported authentication tag *T* is the result of XOR-ing the string consisting of the leftmost *M*=8 octets of S_0 and the octet string *U*:

   T = B9 D7 89 67 04 BC FA 20.

C.4.2 Authentication Checking Transformation

The authentication checking transformation involves the following steps:

1. Form the message *AuthData* using the input transformation in sub-clause C.3.1, with as inputs the string *a* and the octet string *m* that was established in sub-clause C.4.1(step 7.):

   AuthData = 08 00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 ||
   08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
   18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00.

2. Use the authentication transformation in sub-clause C.3.2, with as input the message *AuthData* to compute the authentication tag *MACTag*:

   MACTag = B9 D7 89 67 04 BC FA 20.

3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in sub-clause C.4.1(step 9.):

   T = B9 D7 89 67 04 BC FA 20 = *MACTag*.

Output: Since *MACTag=T*, output ‘valid’ and accept the octet string *m* and accept one of the key sharing group member(s) as the source of *m*.

C.5 Cryptographic Hash Function

This annex provides sample test vectors for the cryptographic hash function specified in clause C.5.

C.5.1 Test Vector Set 1

Input: The input to the cryptographic hash function is as follows:

1. The bit string *M* of length *l*=8 bits to be used:

   M=C0.

Actions: The hash value shall be derived as follows:

1. Pad the message *M* by right-concatenating to *M* the bit ‘1’ followed by the smallest non-negative number of ‘0’ bits, such that the resulting string has length 14 (**mod** 16) octets:

   C0 || 80 00 00 00 00 00 00 00 00 00 00 00 00.

2. Form the padded message *M'* by right-concatenating to the resulting string the 16-bit string that is equal to the binary representation of the integer *l*.

   M' = C0 || 80 00 00 00 00 00 00 00 00 00 00 00 00 || 00 08.

3. Parse the padded message *M'* as M_1, where each message block M_i is a 16-octet string.

4. The hash value $Hash_1$ is computed as follows:

i	$Hash_i$	M_i
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	æ
1	AE 3A 10 2A 28 D4 3E E0 D4 A0 9E 22 78 8B 20 6C	C0 80 00 00 00 00 00 00 00 00 00 00 00 00 00 08

Output: the 16-octet string *Hash* = $Hash_1$ = AE 3A 10 2A 28 D4 3E E0 D4 A0 9E 22 78 8B 20 6C.

C.5.2 Test Vector Set 2

Input: The input to the cryptographic hash function is as follows:

5. The bit string *M* of length *l*=128 bits to be used:

   M=C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

Actions: The hash value shall be derived as follows:

1. Pad the message *M* by right-concatenating to *M* the bit '1' followed by the smallest non-negative number of '0' bits, such that the resulting string has length 14 (**mod** 16) octets:

   C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ||
   80 00 00 00 00 00 00 00 00 00 00 00 00.

2. Form the padded message *M'* by right-concatenating to the resulting string the 16-bit string that is equal to the binary representation of the integer *l*.

   M' = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ||
   80 00 00 00 00 00 00 00 00 00 00 00 00 || 00 80.

3. Parse the padded message *M'* as M_1 || M_2, where each message block M_i is a 16-octet string.

4. The hash value $Hash_2$ is computed as follows:

i	$Hash_i$	M_i
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	æ
1	84 EE 75 E5 4F 9A 52 0F 0B 30 9C 35 29 1F 83 4F	C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
2	A7 97 7E 88 BC 0B 61 E8 21 08 27 10 9A 22 8F 2D	80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08

Output: the 16-octet string *Hash* = $Hash_2$ = A7 97 7E 88 BC 0B 61 E8 21 08 27 10 9A 22 8F 2D.

C.6 Keyed Hash Function for Message Authentication

This annex provides sample test vectors for the keyed hash function for message authentication as specified in clause C.6.

C.6.1 Test Vector Set 1

Input: The input to the keyed hash function is as follows:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F.

2. The bit string *M* of length *l*=8 bits to be used:

   M=C0.

Actions: The keyed hash value shall be derived as follows:

1. Create the 16-octet string *ipad* (inner pad) as follows:

   ipad = 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36.

2. Form the inner key Key_1 by XOR-ing the bit string *Key* and the octet string *ipad*:

   $Key_1 = Key \oplus ipad$ = 76 77 74 75 72 73 70 71 7E 7F 7C 7D 7A 7B 78 79.

3. Form the padded message M_1 by right-concatenating the bit string Key_1 with the bit string *M*:

   $M_1 = Key_1 \,||\, M$ = 76 77 74 75 72 73 70 71 7E 7F 7C 7D 7A 7B 78 79 || C0.

4. Compute the hash value $Hash_1$ of the bit string M_1:

   $Hash_1$ = 3C 3D 53 75 29 A7 A9 A0 3F 66 9D CD 88 6C B5 2C.

5. Create the 16-octet string *opad* (outer pad) as follows:

   opad = 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C.

6. Form the outer key Key_2 by XOR-ing the bit string *Key* and the octet string *opad*:

   $Key_2 = Key \oplus opad$ = 1C 1D 1E 1F 18 19 1A 1B 14 15 16 17 10 11 12 13.

7. Form the padded message M_2 by right-concatenating the bit string Key_2 with the bit string $Hash_1$:

   $M_2 = Key_2 \,||\, Hash_1$ =1C 1D 1E 1F 18 19 1A 1B 14 15 16 17 10 11 12 13 ||
   3C 3D 53 75 29 A7 A9 A0 3F 66 9D CD 88 6C B5 2C.

8. Compute the hash value $Hash_2$ of the bit string M_2:

   $Hash_2$ = 45 12 80 7B F9 4C B3 40 0F 0E 2C 25 FB 76 E9 99.

Output: the 16-octet string *HMAC* = $Hash_2$ = 45 12 80 7B F9 4C B3 40 0F 0E 2C 25 FB 76 E9 99.

C.6.2 Test Vector Set 2

Input: The input to the keyed hash function is as follows:

1. The key *Key* of size *keylen*=256 bits to be used:

   Key = 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F ||
   50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F.

2. The bit string *M* of length *l*=128 bits to be used:

   M = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

Actions: The keyed hash value shall be derived as follows:

1. Compute the hash value Key_0 of the bit string *Key*:

   Key_0 = 22 F4 0C BE 15 66 AC CF EB 77 77 E1 C4 A9 BB 43.

2. Create the 16-octet string *ipad* (inner pad) as follows:

   ipad = 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36.

3. Form the inner key Key_1 by XOR-ing the bit key Key_0 and the octet string *ipad*:

   $Key_1 = Key_0 \oplus ipad$ = 14 C2 3A 88 23 50 9A F9 DD 41 41 D7 F2 9F 8D 75.

4. Form the padded message M_1 by right-concatenating the bit string Key_1 with the bit string *M*:

   $M_1 = Key_1 \,||\, M$ = 14 C2 3A 88 23 50 9A F9 DD 41 41 D7 F2 9F 8D 75 ||
   C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

5. Compute the hash value $Hash_1$ of the bit string M_1:

   $Hash_1$ = 42 65 BE 29 74 55 8C A2 7B 77 85 AC 73 F2 22 10.

6. Create the 16-octet string *opad* (outer pad) as follows:

   opad = 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C.

7. Form the outer key Key_2 by XOR-ing the bit string Key_0 and the octet string *opad*:

   $Key_2 = Key_0 \oplus opad$ = 7E A8 50 E2 49 3A F0 93 B7 2B 2B BD 98 F5 E7 1F.

8. Form the padded message M_2 by right-concatenating the bit string Key_2 with the bit string $Hash_1$:

   $M_2 = Key_2 \,||\, Hash_1$ = 7E A8 50 E2 49 3A F0 93 B7 2B 2B BD 98 F5 E7 1F ||
   42 65 BE 29 74 55 8C A2 7B 77 85 AC 73 F2 22 10.

9. Compute the hash value $Hash_2$ of the bit string M_2:

   $Hash_2$ = A3 B0 07 99 84 BF 15 57 F7 4A 0D 63 87 E0 A1 1A.

Output: the 16-octet string *HMAC* = $Hash_2$ = A3 B0 07 99 84 BF 15 57 F7 4A 0D 63 87 E0 A1 1A.

C.7 Specialized Keyed Hash Function for Message Authentication

This annex provides sample test vectors for the specialized keyed hash function for message authentication as specified in clause C.7.

For test vectors, see clause C.6.

C.8 Symmetric-Key Key Agreement Scheme

This annex provides sample test vectors for the symmetric-key key agreement scheme as specified in clause C.8.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. The unique identifiers of the entities *U* and *V* to be used:

   U's identifier: *U*=55 73 65 72 20 55 0D 0A;

   V's identifier: *V*=55 73 65 72 20 56 0D 0A.

2. The key *Key* of length *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF.

3. The optional parameter *SharedData* of length *shareddatalen*=48 bits to be used:

   SharedData = D0 D1 D2 D3 D4 D5.

C.8.1 Initiator Transformation

U obtains an authentic copy of *V*'s identifier and an authentic copy of the static secret key *Key* shared with *V*.

Input: The input to the initiator transformation is:

1. The length *keydatalen* in bits of the keying data to be generated: *keydatalen*=128.
2. The optional bit string $Text_2$ to be used is not present, i.e., $Text_2 = \varepsilon$ (the empty string).

Actions: *U* derives keying data as follows:

1. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEU* for the challenge domain parameters *D*. Send *QEU* to *V*.

   QEU = 9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96.

2. Then receive from *V* a challenge *QEV'* purportedly owned by *V*. If this value is not received, output 'invalid' and stop.

   QEV' = BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.

3. Receive from *V* an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.

   $Text_1 = \varepsilon$ (the empty string);

   $MacTag_1'$ = E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44 FF 63 D8 D0.

4. Verify that *QEV'* is a valid challenge for the challenge domain parameters *D* as specified in sub-clause B.3.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

5. Use the SKG primitive in clause B.5 to derive a shared secret bit string *Z* from the challenges Q_1=*QEU* owned by *U* and Q_2=*QEV'* owned by *V*, using as key the shared key *Key*. If the SKG primitive outputs 'invalid', output 'invalid' and stop.

   a) Form the bit string *MACData* = *U* || *V* || *QEU* || *QEV'*:

   ```
   MACData = 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
             9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
             BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.
   ```


b) Calculate the *MACTag* for *MACData* under the key *Key* using the tagging transformation of the HMAC-Matyas-Meyer-Oseas MAC scheme:

$MACTag=MAC_{Key}(MACData)$= 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D.

c) Set *Z*=*MACTag*:

Z = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D.

6. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 **[B7]** with the Matyas-Meyer-Oseas hash function to derive keying data *KKeyData* of length 256 bits from the shared secret value *Z* and the shared data *SharedData*:

a) The hash values $Hash_1$, $Hash_2$ are computed as follows:

i	$Hash_i=H(X_i)$	X_i
1	E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 01 || D0 D1 D2 D3 D4 D5
2	72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 02 || D0 D1 D2 D3 D4 D5

b) Set $KKeyData=Hash_1 \| Hash_2$:

KKeyData = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50 ||
72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F.

7. Parse the leftmost 128 bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.

MacKey = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50;

KeyData = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F.

8. Form the bit string $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$:

$MacData_1$ = 02 || 55 73 65 72 20 56 0D 0A || 55 73 65 72 20 55 0D 0A ||
BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53 ||
9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96.

9. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

a) Calculate $MacTag_1=MAC_{MacKey}(MacData_1)$ = E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44 FF 63 D8 D0.

b) Verify that $MacTag_1=MacTag_1$'.

10. Form the bit string $MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]$:

$MacData_2$ = 03 || 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.

11. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation specified in Section 5.7.1 of ANSI X9.63-2001 [B7] with the HMAC-Matyas-Meyer-Oseas MAC scheme:

$$MacTag_2 = MAC_{MacKey}(MacData_2) = 66\ 36\ 8D\ 61\ 0F\ E0\ 0B\ 7F\ 06\ 3E\ 74\ C4\ 78\ 0A\ A3\ 6D. \quad (22)$$

If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and, if present, $Text_2$ to *V*.

Output: output ‘valid’ and accept the 128-bit string *KeyData* = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F as the keying data shared with *V*.

C.8.2 Responder Transformation

V obtains an authentic copy of *U*’s identifier and an authentic copy of the static secret key *Key* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU*’ purportedly owned by *U*.

   QEU’ = 9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96.

2. The length *keydatalen* in bits of the keying data to be generated: *keydatalen*=128.

3. The optional bit string $Text_1$ to be used is not present, i.e., $Text_1 = \varepsilon$ (the empty string).

Actions: *V* derives keying data as follows:

1. Verify that *QEU*’ is a valid challenge for the challenge domain parameters *D* as specified in sub-clause B.3.2. If the validation primitive rejects the challenge, output ‘invalid’ and stop.

2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEV* for the challenge domain parameters *D*. Send to *U* the challenge *QEV*.

   QEV = BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.

3. Use the SKG primitive in clause B.5 to derive a shared secret bit string *Z* from the challenges Q_1=*QEU’* owned by *U* and Q_2=*QEV* owned by *V*, using as key the shared key *SharedKey*. If the SKG primitive outputs ‘invalid’, output ‘invalid’ and stop.

   a) Form the bit string *MACData*=*U* || *V* || *QEU’* || *QEV*:

      MACData= 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
      9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
      BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.

   b) Calculate the *MACTag* for *MACData* under the key *Key* using the tagging transformation of the HMAC-Matyas-Meyer-Oseas MAC scheme:

      $MACTag = MAC_{Key}(MACData)$ = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14
      91 F8 6D.

   c) Set *Z*=*MACTag*:

      Z = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D.

4. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 **[B7]** with the Matyas-Meyer-Oseas hash function to derive keying data *KKeyData* of length 256 bits from the shared secret value *Z* and the shared data *SharedData*:

   a) The hash values $Hash_1$, $Hash_2$ are computed as follows:

i	$Hash_i=H(X_i)$	X_i
1	E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 01 || D0 D1 D2 D3 D4 D5
2	72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 02 || D0 D1 D2 D3 D4 D5

   b) Set *KKeyData*=$Hash_1$ || $Hash_2$:

KKeyData = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50 ||
72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F.

5. Parse the leftmost 128 bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.

   MacKey = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50;

   KeyData = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F.

6. Form the bit string $MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]$:

   $MacData_1$ = 02 || 55 73 65 72 20 56 0D 0A || 55 73 65 72 20 55 0D 0A ||
   BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53 ||
   9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96.

7. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation specified in Section 5.7 of ANSI X9.63-2001 **[B7]** with the HMAC-Matyas-Meyer-Oseas MAC scheme:

   $MacTag_1 = MAC_{MacKey}(MacData_1)$= E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44
   63 D8 D0.

8. If the tagging transformation outputs ‘invalid’, output ‘invalid’ and stop. Send to *U*, if present the bit string $Text_1$, and $MacTag_1$.

9. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$’. If this data is not received, output ‘invalid’ and stop.

   $Text_2$ = ε (the empty string);

   $MacTag_2'$ = 66 36 8D 61 0F E0 0B 7F 06 3E 74 C4 78 0A A3 6D.

10. Form the bit string $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$:

$MacData_2$ = 03 || 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53.

11. Verify that $MacTag_2$’ is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation specified in Section 5.7 ANSI X9.63-2001 **[B7]**. If the tag checking transformation outputs ‘invalid’, output ‘invalid’ and stop.

    a) Calculate $MacTag_2 = MAC_{MacKey}(MacData_2)$ = 66 36 8D 61 0F E0 0B 7F 06 3E 74 C4 78 0A A3 6D.

    b) Verify that $MacTag_2 = MacTag_2$’.

Output: output ‘valid’ and accept the 128-bit string *KeyData* = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F as the keying data shared with *U*.

Annex D ZigBee Protocol Stack, Settable Values (Knobs)

This white paper details the settable parameters within the ZigBee protocol stack.

The goal of this document is to list the various settings that need to be chosen so that differing ZigBee implementations and networks will be able to interoperate. Along with the specific “knobs”/settings a description and potential “cost” be that volatile or non-volatile memory, network constraints or other costs.

These settings fall into three major categories: Network settings, Application Settings, and Security Settings. Each will be covered in a separate section.

D.1 Network Settings

The settable parameters for the Network Layer include:

- *nwkMaxDepth* and *nwkMaxChildren*
- *nwkMaxRouters*
- Size of the routing table
- Size of neighbor table
- Size of route discovery table
- Number of reserved routing table entries
- How many packets to buffer pending route discovery
- How many packets to buffer on behalf of end devices
- Routing cost calculation
- *nwkSymLink*

D.1.1 nwkMaxDepth and nwkMaxChildren

D.1.1.1 Description

The network formation procedure in ZigBee naturally forms trees of association starting with the ZigBee coordinator. In star mode, of course, the tree is a degenerate one with unit depth but tree and mesh mode allow for deeper trees. The NWK Information Base (NIB) attribute nwkMaxDepth specifies the maximum number of allowable levels in a particular tree and the attribute nwkMaxChildren specifies the maximum number of children that any node in the tree may have. By convention, values for these NIB attributes are chosen by the ZigBee coordinator at network startup and shared by all devices in the network.

The network diameter, which is the maximum number of hops a packet will have to travel to reach any other device in the network, is just 2*nwkMaxDepth, while the total address block size, assuming that all the children are routers, which is given by:

$$\frac{1 - nwkMaxChildren^{nwkMaxDepth+1}}{1 - nwkMaxChildren}$$

must be less than or equal to the size of the address space. See sub-clause D.1.2 for a discussion of nwkMaxRouters and of networks containing ZigBee end devices.

D.1.1.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Network size.	**High** – As a network grows, the amount of address space that is allocated at each level is exponential over nwkMaxChildren. Thus a large nwkMaxChildren will rapidly deplete whatever address space is available. Conversely, a “rangy” network with a large diameter must have a relatively small value for nwkMaxChildren.
Device cost.	The chosen values of nwkMaxChildren and nwkMaxDepth may affect device cost in two ways. **High** – A device must have enough RAM to store a neighbor table entry for each of its children and its parent. Large values for nwkMaxChildren will mandate large neighbor tables. **Medium** – Network designers may wish to maintain a high ratio of inexpensive end devices to routers in order keep total installation cost low. This, in turn, mandates a large value for nwkMaxChildren.

D.1.1.3 Value range

NwkMaxDepth	2…7
NwkMaxChildren	1…32

Network sizes given here assume a total address space of 4K addresses corresponding to a 16-bit address word with 4 reserved bits and further assume that all devices are ZigBee routers.

Value Setting	Tradeoff
“rangy” network {nwkMaxChildren = 3, nwkMaxDepth = 7}	Network diameter = 14 Maximum devices in network = 3280 Neighbor table entries per router = 4
Nominal network {nwkMaxChildren = 15, nwkMaxDepth = 3}	Network diameter = 6 Maximum devices in network = 3616 Neighbor table entries per router = 16
“bushy” network {nwkMaxChildren = 32, nwkMaxDepth = 2}	Network diameter = 4 Maximum devices in network = 1057 Neighbor table entries per router = 33

D.1.2 NwkMaxRouters

D.1.2.1 Description

The NIB attribute nwkMaxRouters may be used to limit the number of ZigBee routers that a ZigBee router or ZigBee coordinator may take on as children thereby permitting a degree of control over the ratio of ZigBee routers to Zigbee end devices. Like nwkMaxDepth and nwkMaxChildren its value is assigned by the ZigBee coordinator at network startup time and distributed to all other devices in the network.

D.1.2.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Network coverage,	**Medium** – Routers are needed to move traffic around the network. End devices, by definition, perform no routing. Thus, in order to assure that traffic is able to move around the network and to prevent communications bottlenecks, the largest possible value for nwkMaxRouters should be used.
Total installation cost.	**High** – It is presumed that ZigBee routers will be more expensive than ZigBee end devices. One way to control the total cost of an installation is to make as many of the devices in a network as possible, end devices.
Power consumption.	**High** – ZigBee routers are generally presumed to be mains-powered devices. In fact, ZigBee supports beacon-enabled routers and, for some applications, this may be enough to allow battery-powered routers as well, but, certainly for the residential and commercial building control application areas, routers will be mains-powered. End devices, on the other hand, may be battery powered and so, if the application calls for a device to be battery powered, it will most likely be an end device.

D.1.2.3 Value range

NwkMaxRouters	1-32

The values in the following table the effect of varying nwkMaxRouters in what might be considered a typical home control network – nwkMaxChildren = 20, nwkMaxDepth = 4. In each case, the value for end devices per router holds for routers in the “middle” of the tree. All leaf nodes may be end devices if so desired.

Value Setting	Tradeoff
4	Maximum devices in network = 1701 End devices per router = 16
6	Maximum devices in network = 5181 End devices per router = 14
8	Maximum devices in network = 11701 End devices per router = 12

The values in the table below are comparable values for wider-ranging network that might be used in building control {nwkMaxChildren = 22, nwkMaxDepth = 5}.

Value Setting	Tradeoff
4	Maximum devices in network = 7503 End devices per router = 18
6	Maximum devices in network = 34211 End devices per router = 16
7	Maximum devices in network = 61623 End devices per router = 15

D.1.3 Size of routing table

D.1.3.1 Description

ZigBeeevices may set aside storage for routing entries that record the next hop in the multi-hop chain required to deliver a packet to a particular destination. The minimum required information to be stored in routing tables, as described in the current version of the specification, is as follows:

Field Name	Size	Description
Destination address	2 bytes	The 16-bit network address of this route.
Status	3 bits	The status of the route. See sub-clause D.1.3.3 for values.
Next-hop address	2 bytes	The 16-bit network address of the next hop on the way to the destination.

In estimating the size of the entries we can a 5-byte entry.

D.1.3.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**Medium** – Every routing table entry adds 5 bytes of RAM to the ZigBee device.
Routing optimality.	**Low** – A small number of routing entries will result in a greater preponderance of tree routing. The routes chosen in this way may require more hops and therefore generate more traffic than the tree routes that would be used if devices had room for them.
Network reliability.	**Medium** – Tree routes have no choice but to use tree links and are therefore more liable to fail in the case of a flaky or asymmetrical tree link. Mesh routes should be more robust since they have the opportunity to pick the best available links and apply LQI measurement to improve their performance over time.

D.1.3.3 Value range

Routing Table Size	0…Unspecified Maximum.

Value Setting	Tradeoff
0 (minimum)	0 bytes of RAM This minimum value may be sufficient for networks where the placement of routers is flexible enough that the resulting tree routes may be tuned for optimal performance and the traffic load is low enough that bottlenecks are not likely to develop near the ZigBee coordinator.
32 (typical)	160 bytes of RAM. This is a number that will probably suffice for Resi-Light-Comm installations with localized control.

D.1.4 Size of neighbor table

D.1.4.1 Description

The neighbor table is used for keeping track of a device's neighbors in the network. There are several classes of neighbor table entry that may be used by implementers for this task.

— Every device must keep track of its children in the tree and its parent. It may want to track the link quality of packets received from each to support routing cost calculation.

— A device may keep track of neighbors from which it has received or may receive route requests. It may also track LQI for packets received from each of these neighbors.

— When a device joins (or forms) a network it performs a sequence of scans and the data from these scans must be stored, at least temporarily, while it evaluates which network to join and so on.

In a practical implementation, the information stored here may be stored in a single table or multiple tables at the implementer's discretion. A rough size for each of these types of table entry follows:

Entry	Size in bytes
Parent/Child.	11 bytes without LQI. 12 bytes with LQI.
Routing neighbor.	2 bytes without LQI. 3 bytes with LQI.
Temporary entry during startup and joining.	15 bytes.

Notes that these are minimum numbers. Implementers may choose to store more information about a device's neighbors, e.g. a timestamp recording when how recently the device has been heard from.

The discussion here will center on permanent storage only.

D.1.4.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**Medium** – Every neighbor table entry requires a fixed amount of RAM depending on its type as described above. The number of entries for storing parents and children is not completely at the discretion of the developer since it is equal to nwkMaxChildren + 1. An implementer may set aside any number of 3-byte entries for additional neighbors to be used in routing.

D.1.4.3 Value range

Neighbor Table Size	NwkMaxChildren + 1 entries for children/parent. [0… Unspecified Maximum] entries for other neighbors.

Value Setting	Tradeoff
Sample minimum for nwkMaxChildren = 15	192 bytes of RAM including LQI values.
nwkMaxChildren = 15, 16 additional neighbors	240 bytes of RAM including LQI values.

D.1.5 Size of route discovery table

D.1.5.1 Description

The route discovery table is used to temporarily store information that is needed during route discovery. The contents of the route discovery table, as excerpted from the most recent version of the specification are:

Field Name	Size	Description
Route request ID	1 byte	A sequence number for a route request command frame that is incremented each time a device initiates a route request.
Source address	2 bytes	The 16-bit network address of the route request's initiator.
Sender address	2 bytes	The 16-bit network address of the node that has sent the most recent lowest cost route request command frame corresponding to this entry's Route request ID and Source address. This field is used to determine the path that an eventual route reply command frame should follow
Residual Cost	1 byte	The accumulated path cost from source of the route request to the current device
Forward routing cost	1 byte	The accumulated path cost from the current device to the destination device
Path cost	1 byte	The accumulated PCM value
Expiration time	2 bytes	A countdown timer indicating the number of milliseconds until route discovery expires. The initial value is *nwkcRouteDiscoveryTime*.

Minor changes may occur in the definition table during the next few revisions of the specification but it will stay at roughly this size, i.e. around 10 bytes per entry.

Entries in this table are only valid during route discovery and may be reused.

D.1.5.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**Low** – Every route discovery table entry takes up about 10 bytes of RAM.

D.1.5.3 Value range

Route Discovery Table Size	1…Unspecified Maximum

Value Setting	Tradeoff
1 (minimum)	10 bytes of RAM. The device may only process one route discovery at a time and some routes will not be discovered.
8 (recommended)	80 bytes of RAM. This is enough to handle 8 route discoveries at once and should be enough for most networks. More testing should be performed to fine-tune this number.

D.1.6 Number of reserved routing table entries

D.1.6.1 Description

A device may set aside some number of routing table entries to be used in the “dire” case where a route is broken and the device that wants to repair it has no routing capacity to do so. This is most likely to happen when the link is a parent-child link.

D.1.6.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**Low** – As with the standard routing table, every reserved entry takes up about 5 bytes of RAM.
Network reliability.	**High** – For tree networks and mesh networks containing a large number of devices with small routing capacity, having a small repair table may spell the difference between having a portion of the network become available due to a broken tree link and being able to repair around that link.

D.1.6.3 Value range

Repair Table Size	0…Unspecified Maximum

Value Setting	Tradeoff
0 (minimum)	With no repair table, a device will not be able to participate in tree repair and, if one of its forward links breaks, network partition may result.
1	5 bytes of RAM. This is enough to repair a single route across a single broken link and may be sufficient in some cases.
8 (recommended)	40 bytes of RAM More testing is require to fine tune this number.

D.1.7 Buffering pending route discovery

D.1.7.1 Description

An RN+ may choose to buffer a frame pending route discovery or it may relay it directly along the tree and, preferably after a short delay, initiate route discovery.

D.1.7.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**High**– Each frame buffered may take up as much as the standard ZigBee payload size (TBD) plus the network header. This may be on the order of 100 bytes per frame.
Network reliability.	**Low** – Frames relayed in this way stand a slightly larger chance of being dropped in transit due to bad tree links or interference from route discovery traffic.

D.1.7.3 Value range

Number of frames buffered	0…Unspecified Maximum

Value Setting	Tradeoff
0 (minimum)	With no buffering, all frames are relayed along the tree before route discovery is initiated.
2 (suggested maximum for 8-bit implementations)	Most 8-bit processors with a RAM complement of 2-4K will not be able to afford more than 200 bytes of buffering for this purpose.

D.1.8 Buffering on behalf of end devices

D.1.8.1 Description

A ZigBee coordinator or ZigBee router may choose to buffer broadcast frames on behalf of sleeping end devices to be transmitted in response to a later data request or a poll request. The alternative is to leave the responsibility of relaying broadcasts to end devices to the application.

D.1.8.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**High**– Each frame buffered may take up as much as the standard ZigBee payload size (TBD) plus the network header. This may be on the order of 100 bytes per frame.

D.1.8.3 Value range

Number of frames buffered	0…Unspecified Maximum

Value Setting	Tradeoff
0 (minimum)	With no buffering, broadcasts to end devices must be relayed by the application. Another way of saying this is that routers and coordinators must act as proxies for sleeping end devices at the application layer.
2 (suggested maximum for 8-bit implementations)	Again, most 8-bit processors with a RAM complement of 2-4K will not be able to afford more than 200 bytes of buffering for this purpose. In this case, two broadcasts may be held until they are delivered to all sleeping children.

D.1.9 Routing cost calculation

D.1.9.1 Description

In order to allow the comparison of possible routes, ZigBee routers are required to add a link cost value to the path cost field of route request and route reply command frames. Implementers are allowed wide latitude with regard to the technique for producing this link cost value. They may actual opt out and report a fixed value (TBD) or they may use instantaneous LQI, an LQI value that has been averaged over time or, in fact, any other scheme to derive the probability that a packet will be delivered over the link in question. The link cost, then should be the reciprocal of that probability.

D.1.9.2 Cost impact

Cost Item	Impact (High/Medium/Low)
Device cost.	**Low** – A device that calculates link cost must perform a simple computation for each packet received and store the result in the neighbor table entry corresponding to the sender of that packet. It must also use table-lookup or some other method to derive a link cost from that value at route discovery time. The example table in the specification for this purpose is 8 bytes long.
Network reliability.	**High –** The primary reason to rate link quality is that it protects the routing algorithm from choosing unreliable routes that are shorter over reliable routes that happen to be longer and it gives the algorithm a rationale for choosing between multiple paths of the same length some of which may be more reliable than others.

D.1.9.3 Value range

Calculate link cost	YES or NO

Value Setting	Tradeoff
Yes	A device may rate the quality of a link and that rating may improve over time so that routing choices reflect the operational conditions of the network.
No	A device can neither rate link reliability or improve its rating over time. In a network where all devices use this technique the probability that flaky, unreliable and expensive routes will be chosen is greatly increased.

D.1.10 nwkSymLink

D.1.10.1 Description

In most network environments, we can assume that some percentage of the links presented to any given device in the network will be asymmetrical in the sense that the link quality to be had by communicating in one direction will differ, often substantially, from the link quality in the other direction. The reasons for this are unsurprising and have to do with the physical characteristics of the wireless medium as well as with the characteristics of the devices being employed. In the case where link symmetry can, for the most part, be

assumed, an optimization becomes available whereby the forward and reverse path of a route can be established at the same time. The *nkSymLink* NIB attribute determines whether this assumption, and the resulting optimization are made during route discovery.

D.1.10.2 Cost Impact

Cost Item	Impact (High/Medium/Low)
Network traffic load	**Medium** – The traffic load on a network that is performing route discovery is substantial. This traffic may cause regular data traffic in transit at the same time to be lost. A mitigating factor is that the bulk of route establishment operations will happen at network startup or at times when the network restarts due to wide-area failures.
Network reliability	**High –** Erroneous assumptions about link symmetry can have disastrous results since it may cause the establishment of unviable routes, which may be impossible to repair since forward and reverse route discovery will always be performed together but link quality will only be measured in the forward direction.

D.1.10.3 Value Range

nwkSymLink	TRUE or FALSE

Value Setting	Tradeoff
TRUE	A device may assume link symmetry and perform forwards and backwards route discovery at the same time, at the risk of establishing unusable routes.
FALSE	A device must perform forward and reverse route discovery separately thereby loading the network with more discovery traffic in the case where routes are needed in both directions but both routes are much more likely to be viable in the presence of asymmetric links

D.2 Application Settings

The settable parameters for the Application Layer include:

- Logical device type
- Stack profile and beacon payload parameters
- Number of active endpoints per device (maximum)
- Discovery information cache size (minimum)
- Binding table size (minimum)
- End to end response messaging
- Acknowledged service in APS

D.2.0.1 Logical device type

D.2.0.2 Description

ZigBee coordinator – Device will scan to find an unused channel and start a new network.

ZigBee router – Device will scan to find an existing network and join as a router

ZigBee end device – Device will scan to find an existing network and join as an end device.

D.2.0.3 Cost impact

Cost Item	Impact (High/Medium/Low)
ZigBee coordinator	**High** – Designating a specific device to be the ZigBee coordinator places a requirement on system deployment to install a special device in the network capable of starting the network. Additionally, the ZigBee coordinator must have resources for a binding table and trust center (if security is used). The amount of resources allocated must match the expected size of the network it serves (including growth).
ZigBee router	**High** – To provide mesh routing, ZigBee routers must be deployed such that at least one has connectivity to the ZigBee coordinator and there is continuous connectivity from router to router to the edge of the network. This implies a set of installation and deployment requirements tied to logical device type. Each ZigBee router must contain network routing software and some resource allocation for routing tables or tree repair tables.
ZigBee end device	**Low** – Each ZigBee end device may be minimally configured as long as provisions have been made above for the ZigBee coordinator and routers.

D.2.0.4 Value Range

Logical Device Type	Coordinator, router or end device

Value Setting	Tradeoff
ZigBee coordinator	Need at least 1
ZigBee router	Need to deploy such that entire network spans no more than nwkMaxDepth
ZigBee end device	Need to deploy in such a way that all ZigBee end devices are serviced by some router out to nwkMaxDepth.

D.2.0.5 Stack profile and beacon payload parameters

D.2.0.6 Description

This setting is applicable to all ZigBee devices. The application will have some desired network settings provided as configuration settings to ZDO. These settings will either be used to configure the network to be created (ZigBee coordinator) or will be used to select a network to join (ZigBee router and end device). These parameters will be established by the specific needs of the Profile or Profiles supported in the device. For devices with multiple applications running on different endpoints, there must be agreement on a single stack profile and set of network settings. The following parameters are settable:

— Stack Profile – Network Specific, Home Controls, Building Automation or Plant Control

— NwkcProtocolVersion – Specifies the ZigBee protocol version. The rules for joining (or not joining) networks with specific Protocol Versions will be established in later versions of this specification.

— NwkSecurityLevel – Specifies the security level of the network.

D.2.0.7 Cost impact

Cost Item	Impact (High/Medium/Low)
Stack Profile of Network Specific	**High** – For devices employing network specific stack profiles or wanting to join networks advertised as network specific, the device must first join the network to determine whether parameters not advertised in the beacon payload are operationally acceptable. Parameters such as the (minimum) size of the neighbor table, (minimum) size of the route discovery table, etc. are key to a fully interoperable network.
NwkcProtocolVersion	**Low** – For now, this is a benign parameter value. If it is needed in later versions, it could become a parameter with very high cost (for example, if v1.1 features are defined such that they are not compatible with v1.0).
NwkSecurityLevel	**High -** This parameter has values from 0x0 (security off) to 0x7 (highest security). Each of the security levels applies to each of the stack profiles. If application profiles are written such that they require a specific security level, then we will end up with another dimension on the stack profile that will complicate interoperability.

D.2.0.8 Value Range

Stack Profile	Network Specific – 0x0 Home Controls – 0x1 Building Automation – 0x2 Plant Control – 0x3
NwkSecurityLevel	Security off – 0x0 Security level – 0x1 – 0x7

Value Setting	Tradeoff
Stack Profile	Selection of a single stack profile, assuming there are a small number of stack profiles, greatly simplifies network selection and aids in interoperability. If, however, stack profiles proliferate and are used as multiple variations on stack parameter settings, then the same interoperability concerns will surface which led to creation of stack profiles to begin with.
Stack Profile of Network Specific	This parameter setting should really only be selected for closed networks. Use of this parameter for networks where interoperability is desired will result in complex join procedures where devices must determine if the network settings can support their applications.
NwkSecurityLevel	Rules must be established on how this parameter is used. For example, suppose a Home Controls stack profile specifies nwkSecurityLevel of 0x3 (vs. 0x4 wanted by a prospective joining device). The nwkSecurityLevel should not be permitted to become another dimension on the stack profile (else, the security level should become a setting WITHIN the stack profile and not a separate parameter).

D.2.0.9 Number of active endpoints per device (maximum)

D.2.0.10 Description

Each endpoint needs a descriptor/description for use with Service Discovery. These descriptors can be up to Zigbee network payload size. For any sleeping devices the coordinator must cache these values so it can act as a proxy for Service Discovery.

D.2.0.11 Cost impact

Cost Item	Impact (High/Medium/Low)
Non-volitile memory on end device to store description for each interface	**High** – Each endpoint must have a Simple Descriptor. Worst case, the Simple Descriptor can be as large as a single ZigBee application packet (64 bytes with security). With a maximum 240 endpoints/ interfaces * 64 bytes this is a storage requirement of 15.360 bytes!
Non-volitile or volitile memory on coordinator to be able to cache descriptors for each “child” device and each interface on each child	**High** – Each coordinator/router can have nwkMaxChildren. If they all wanted to sleep, they would want their coordinator/ router to cache their service discovery information. This would include a Node Descriptor, Power Descriptor for each device plus as many Simple Descriptors as each device had for every active endpoint (see above item – this is a very substantial number).

D.2.0.12 Value Range

Number of endpoints/interfaces per device	1-240

Value Setting	Tradeoff
1 (minimum)	
2? (typical)	
240 (maximum)	

D.2.0.13 Discovery Information Cache Size

D.2.0.14 Description

Each device holds the following descriptors:

- Node Descriptor – 6 bytes (mandatory)
- Power Descriptor – 2 bytes (mandatory)
- Simple Descriptor – variable, one per active endpoint/interface (mandatory)
- Complex Descriptor – variable (optional)
- User Descriptor – variable (optional).

For each end device that intends to sleep, the ZigBee coordinator or router it associates to must cache the above information for the device and respond to service discovery. In addition to the nwkMaxChildren and nwkMaxRouters parameters, this cache size must be considered when permitting a device to associate.

D.2.0.15 Cost impact

Cost Item	Impact (High/Medium/Low)
Cache size	**High** – Though nwkMaxChildren may indicate that a given router or coordinator could support additional children, the size of the cache available to support sleeping devices along with the application requirements of the sleeping device must be considered.

D.2.0.16 Value Range

Discovery Information cache size	

Value Setting	Tradeoff
?	Need to establish values for this parameter. Currently, there is no indication from a joining device as to the number and size of Simple Descriptors they support.

D.2.0.17 Binding Table Size

D.2.0.18 Description

The Binding Table is held by the ZigBee coordinator and contains entries with the following information:

— Source address (64 bits)

— Source endpoint/interface (8 bits)

— Cluster ID (8 bits)

— Destination address (64 bits)

— Destination endpoint/interface (8 bits)

The above information is provided per entry. The number of entries is a function of the number of devices in the network and the number of expected bindings per device.

D.2.0.19 Cost impact

Cost Item	Impact (High/Medium/Low)
Binding Table Size	**High** – Assume a network of 400 devices, what assumptions are required to set an appropriate binding table size? The size is a function of the devices in the network and the application needs of those devices. For example, a sensor application may use 0 Binding Table entries. A Lighting solution may have 1 entry per device. In some extreme cases (like theatre lighting), there may be multiples of Binding Table entries per device (if a given set of lights were controlled by multiples of switches).

D.2.0.20 Value Range

Number of pairings the binding coordinator can hold	

Value Setting	Tradeoff
0 (minimum)	Devices requiring Binding Table support should not join this type of network.
1 per network device? (typical)	It must be known if this type of Binding Table size can support the needs of the application on the device joining the network.
? (maximum)	For some specialized applications, the Binding Table size may need to be larger than 1 per device on the network.

D.2.0.21 End to End Response Messaging

D.2.0.22 Description

According to the Application Framework, response messaging is optional. It would appear to be perfectly legal to define all messaging within a given application as not requiring responses. In fact, given that the NWK layer is non-guaranteed delivery, it would not be possible to determine if the application was successfully sending any messages to its intended destination.

D.2.0.23 Cost impact

Cost Item	Impact (High/Medium/Low)
End to end messaging on all requests	**High** – Each application would be responsible for creating a timer to ensure that response messages are received for every command. A retry mechanism would need to be instituted for messages that are not acknowledged along with error handling in cases where the retry limit is exceeded.
End to end messaging on some requests	**Medium** – The application could utilize APS level acknowledgement that provides assurance that messages are being received at the destination, then use non-guaranteed delivery for intervening commands. Use of this feature would depend on the application nature of the commands being sent and the relative importance on the destination receiving all commands reliably (ie. Whether the message is repeated like a measurement or whether it is a control command)

D.2.0.24 Value Range

End to End Response Messaging	

Value Setting	Tradeoff
End to end application messaging used at all times	Application must implement timeouts and retries. Application must handle error conditions when retry limits are exceeded.
End to end application messaging is used periodically	Application must still implement timeouts and handle error conditions. Additionally, application must assume that failures of application commands/responses are also occurring and be designed to be immune from such failure.
End to end application messaging is not used.	Application has no feedback that any messages sent are received at the destination.

D.2.0.25 Acknowledged Service in APS

D.2.0.26 Description

An acknowledged service was added to APS. This optional service is required in cases such as replies to broadcast service or device discovery commands, however, may be employed for other application messaging under application control.

D.2.0.27 Cost impact

Cost Item	Impact (High/Medium/Low)
Acknowledgements on all APS data requests	**Medium** – Each APS data request would need to receive an acknowledgement. This could cause a need to either buffer requests or to discard data requests within APS (depends on applications communication requirements)
Acknowledgements on some APS data requests	**Low** – The only required acknowledgements will be for unicast responses to broadcast requests (such as for the Device Profile primitives NWK_addr_req and Match_Desc_req).

D.2.0.28 Value Range

Acknowledgements on APS Data Requests	

Value Setting	Tradeoff
Acknowledgements used at all times	Application must implement timeouts and retries. Application must handle error conditions when retry limits are exceeded.
Acknowledgements are used only for required actions (NWK_addr_rsp, Match_Desc_rsp and any other unicast responses to a broadcast device or service discovery request).	None.

D.3 Security Settings

The settable parameters for the Security Services Provider include:

- Security level
- Master key source
- Always use NWK-layer security
- Number of NWK keys
- Number of application keys
- Number of frame counters used
- SecurityTimeoutPeriod

D.3.0.1 Security level

D.3.0.2 Description

The type of security used by the device (if any).

D.3.0.3 Cost impact

Cost Item	Impact (High/Medium/Low)
NVM if security is used	

D.3.0.4 Value Range

Security level	0x00-0x07

Value Setting	Tradeoff
0x00-none	No security; no cost of security
0x01-MIC-32 (32-bit Message Integrity Code)	Moderate integrity protection; longer packet length, NVM key storage needed
0x02-MIC-64 (64-bit Message Integrity Code)	Strong integrity protection; longer packet length, NVM key storage needed
0x03-MIC-128 (128-bit Message Integrity Code)	Strongest integrity protection; longest packet length, NVM key storage needed
0x04-ENC (Encryption only)	Message privacy; NVM key storage needed
0x05-ENC-MIC-32 (Encryption and 32-bit Message Integrity Code)	Encryption with moderate integrity protection; longer packet length, NVM key storage needed
0x06-ENC-MIC-64 (Encryption and 64-bit Message Integrity Code)	Encryption with strong integrity protection; longer packet length, NVM key storage needed
0x07-ENC-MIC-128 (Encryption and 128-bit Message Integrity Code)	Encryption with strongest integrity protection; longer packet length, NVM key storage needed

D.3.0.5 Master key source

D.3.0.6 Description

The master key for the trust center may come from several places; this affects the behavior of the network device.

D.3.0.7 Cost impact

Cost Item	Impact (High/Medium/Low)
Factory installation	High-difficult to control through distribution chain
User interface	High-effects BoM, product design, user experience

D.3.0.8 Value Range

Master key source	Factory installation, installed by network trust center, or entered by user

Value Setting	Tradeoff
Factory installation	Easy to use by user, as long as network is as envisioned by factory; difficult to track through distribution chain, difficult to add new devices, difficult to deploy in industrial settings
Installed by trust center	Easy to use by user; onus on ZigBee to minimize algorithm complexity
Entered by user	Flexible, responsive to varied network designs

D.3.0.9 Always use Network layer security – on or off

D.3.0.10 Description

Network layer security is needed to prevent theft of network service—“freeloading” devices using the network to route frames between themselves.

D.3.0.11 Cost impact

Cost Item	Impact (High/Medium/Low)
	?

D.3.0.12 Value Range

Network layer security	ON or OFF

Value Setting	Tradeoff
ON	No theft of service; longer frames
OFF	Shorter frames; possible theft of service

D.3.0.13 Number of NWK Keys

D.3.0.14 Description

The network key is used to secure MAC and NWK-layer frames.

D.3.0.15 Cost impact

Cost Item	Impact (High/Medium/Low)
NVM storage	Low

D.3.0.16 Value Range

Keys	0, 1, 2

Value Setting	Tradeoff
0	No NVM; network not secure
1	Network packets secure; NVM, possible loss of network function for short period while key updates
2	Network packets secure, no loss of network function for short period while key updates; NVM

D.3.0.17 Number of Application Keys

D.3.0.18 Description

Application keys are used to secure end-to-end links.

D.3.0.19 Cost impact

Cost Item	Impact (High/Medium/Low)
NVM storage	Medium

D.3.0.20 Value Range

Application Keys	0-16?

Value Setting	Tradeoff
?	The more keys, the more NVM, but the more flexible and powerful the device.

D.3.0.21 Number of Frame Counters Used

D.3.0.22 Description

A frame counter must be used for each device with which a network node communicates securely.

D.3.0.23 Cost impact

Cost Item	Impact (High/Medium/Low)
NVM storage	Medium

D.3.0.24 Value Range

Frame counters--RFD	1
Frame counters--FFD	16

Value Setting	Tradeoff
1	Low cost; minimal functionality
16	Can communicate securely with 15 children plus parent; more NVM

D.3.0.25 Security timeout periods

D.3.0.26 Description

— Maximum length of time either an initiator (e.g., a joining device) or a responder (e.g., a beaconing device) may wait for an expected incoming SKKE message before generating an error code.

— Maximum length of time either an initiator or a responder may wait for an expected incoming message in the entity authentication protocol before generating an error code.

Annex E ZigBee Stack Profiles

This Annex details the stack profiles for ZigBee protocol stack.

E.1 Stack Profiles

Stack Profiles are a convention on specific ZigBee stack settable values established to provide interoperability in specified markets. See Annex D for descriptions on the various settings.

The following stack profiles have been identified:

a) Home Controls

b) Building Automation

c) Plant Control

Additionally, a category of stack profile called “Network Specific” is proposed which indicates that no specific Stack Profile is in use, rather, the stack parameters are defined by the elemental values employed as stack parameters.

E.2 Stack Profile Definitions

The ZigBee Network (NWK) Specification provides for identification of the Stack Profile within the beacon payload.

Stack Profile Name	Stack Profile Identifier (02130r7)
Network Specific	0x0
Home Controls	0x1
Building Automation	0x2
Plant Control	0x3
Reserved	0x4-0xf

E.3 Home Controls Stack Profile

The Home Controls Stack Profile is intended for use with the Home Controls-Lighting profile and all profiles written for complementary use with Home Controls-Lighting.

E.3.1 Network Settings

Parameter Name	Setting
Beacon Order Superframe Order	0x0f (no beacon) 0x0f (ignored)
NwkMaxDepth and nwkMaxChildren	5 20
NwkMaxRouters	6
Size of the routing table (minimum)	8
Size of the neighbor table (minimum)	ZigBee coordinator: 24 ZigBee router: 25 ZigBee end device: 1
Size of the route discovery table (minimum)	4
Number of reserved routing table entries (minimum)	8
Number of packets buffered pending route discovery (minimum)	0
Number of packets buffered on behalf of end devices (minimum)	1
Routing cost calculation	True
NwkSymLink	False

E.3.2 Application Settings

Parameter Name	Setting
Logical device type	ZigBee Coordinator – 1 ZigBee Router – no more than 6 per coordinator/router to join at the higher level of the tree ZigBee End Device – no more than 20 per coordinator/router
Stack profile and beacon payload parameters	Stack profile – 0x1 NwkcProtocolVersion – 0x0 NwkSecurityLevel – 0x5
Number of active endpoints per device (minimum)	3
Discovery information cache size (minimum, coordinator/routers only, per sleeping child devices)	1036 bytes[a]

Binding table size (minimum, coordinator only)	100 entries (1900 bytes)[b]
End to end response messaging (per cluster/attribute per profile)	Agreed to for each cluster in each profile within Home Controls.
Acknowledged service in APS	Agreed to for each cluster in each profile within Home Controls.

[a]Assumptions: nwkMaxChildren of 20 minus nwkMaxRouters of 6 (net of 14), each with: Node Descriptor – 6 bytes, Power Descriptor – 2 bytes, Simple Descriptor (3 each, max of 10 input/output clusters per), User Descriptor of 12 = 1036 bytes

[b]Assumptions: Each Binding Table entry is: Source Address (8 bytes)+Source Endpoint(1 byte)+ClusterID(1 byte)+Dest Address (8 bytes)+Dest Endpoint (1 byte) = 19 bytes

E.3.3 Security Settings

Parameter Name	Setting
Security Level	0x5
Master Key Source	Entered by user (includes key pad, button press, low RF “learn mode” or other user initiated action)
Always use Network layer security – on or off	ON
Number of NWK Keys	2
Number of Application Keys	0
Number of Frame Counters used	1 for RFD, 20 for FFD
Security timeout periods	50ms * (2*nwkMaxDepth) + (AES Encrypt/Decrypt times)

E.4 Building Automation Stack Profile

The Building Automation Stack Profile is intended for use with future profiles targeted to building automation solutions.

E.4.1 Network Settings

Parameter Name	Setting
Beacon Order Superframe Order	0x0f (no beacon) 0x0f (ignored)
NwkMaxDepth and nwkMaxChildren	9 (2nd choice: 7) 6 (2nd choice: 12)

NwkMaxRouters	3 (2^{nd} choice: 4)
Size of the routing table (minimum)	16
Size of the neighbor table (minimum)	ZigBee coordinator: 15 ZigBee router: 16 ZigBee end device: 1
Size of the route discovery table (minimum)	8
Number of reserved routing table entries (minimum)	8
Number of packets buffered pending route discovery (minimum)	0
Number of packets buffered on behalf of end devices (minimum)	1
Routing cost calculation	True
NwkSymLink	False

E.4.2 Application Settings

Parameter Name	Setting
Logical device type	ZigBee Coordinator – 1 ZigBee Router – no more than 3 per coordinator/router to join at the higher level of the tree ZigBee End Device – no more than 9 per coordinator/router
Stack profile and beacon payload parameters	Stack profile – 0x2 NwkcProtocolVersion – 0x0 NwkSecurityLevel – 0x6
Number of active endpoints per device (minimum)	7
Discovery information cache size (minimum, coordinator/routers only, per sleeping child devices)	588 bytes
Binding table size (minimum, coordinator only)	50 entries (950 bytes)
End to end response messaging (per cluster/attribute per profile)	Agreed to for each cluster in each profile within Building Automation.
Acknowledged service in APS	Agreed to for each cluster in each profile within Building Automation.

E.4.3 Security Settings

Parameter Name	Setting
Security Level	0x6
Master Key Source	Installed by trust center
Always use Network layer security – on or off	ON
Number of NWK Keys	2
Number of Application Keys	20
Number of Frame Counters used	1 for RFD, 6 for FFD
Security timeout periods	50ms * (2*nwkMaxDepth) + (AES Encrypt/Decrypt times)

E.5 Plant Control Stack Profile

Editor's Note: This section was not reviewed as of Release 1 of the document.

The Plant Control Stack Profile is intended for use with future profiles targeted to plant control solutions.

E.5.1 Network Settings

Parameter Name	Setting
Beacon Order Superframe Order	0x0f (no beacon) 0x0f (ignored)
NwkMaxDepth and nwkMaxChildren	5 22
NwkMaxRouters	7
Size of the routing table (minimum)	16
Size of the neighbor table (minimum)	ZigBee coordinator: 30 ZigBee router: 31 ZigBee end device: 1
Size of the route discovery table (minimum)	8
Number of reserved routing table entries (minimum)	8
Number of packets buffered pending route discovery (minimum)	0

Number of packets buffered on behalf of end devices (minimum)	3
Routing cost calculation	True
NwkSymLink	False

E.5.2 Application Settings

Parameter Name	Setting
Logical device type	ZigBee Coordinator – 1 ZigBee Router – no more than 7 per coordinator/router to join at the higher level of the tree ZigBee End Device – no more than 22 per coordinator/router
Stack profile and beacon payload parameters	Stack profile – 0x3 NwkcProtocolVersion – 0x0 NwkSecurityLevel – 0x7
Number of active endpoints per device (minimum)	7
Discovery information cache size (minimum, coordinator/routers only, per sleeping child devices)	2940 bytes
Binding table size (minimum, coordinator only)	100 entries (1900 bytes)
End to end response messaging (per cluster/attribute per profile)	Agreed to for each cluster in each profile within Plant Control.
Acknowledged service in APS	Agreed to for each cluster in each profile within Plant Control.

E.5.3 Security Settings

Parameter Name	Setting
Security Level	0x6
Master Key Source	Installed by trust center
Always use Network layer security – on or off	ON
Number of NWK Keys	2
Number of Application Keys	23
Number of Frame Counters used	1 for RFD, 22 for FFD
Security timeout periods	50ms * (2*nwkMaxDepth) + (AES Encrypt/Decrypt times)

Annex F KVP XML schemas

This annex contains the XML schemas for the ZigBee ZVP commands.

F.1 XML schema for the get command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF" xmlns="http://www.zigbee.org/v1.0/AF"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="GetCommand">
        <xs:annotation>
            <xs:documentation>Schema for AF get command</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="AttribDataType" type="xs:unsignedByte"/>
                <xs:element name="AttribId" type="xs:unsignedShort"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

F.2 XML schema for the get response command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF “xmlns="http://www.zigbee.org/v1.0/AF"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Get_ResponseCommand">
        <xs:annotation>
            <xs:documentation>Schema for AF get response command</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="AttribDataType" type="xs:unsignedByte"/>
                <xs:element name="AttribId" type="xs:unsignedShort"/>
                <xs:element name="ErrorCode" type=" xs:unsignedByte "/>
                <xs:element name="AttribData" type="xs:anyType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

F.3 XML schema for the set command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF “xmlns="http:/
/www.zigbee.org/v1.0/AF" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="SetCommand">
      <xs:annotation>
         <xs:documentation>Schema for AF set command</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="AttribDataType" type="xs:unsignedByte"/>
            <xs:element name="AttribId" type="xs:unsignedShort"/>
            <xs:element name="AttribData" type="xs:anyType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

F.4 XML schema for the set response command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF “xmlns="http:/
/www.zigbee.org/v1.0/AF" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Set_ResponseCommand">
      <xs:annotation>
         <xs:documentation>Schema for AF set response command</xs:docu-
mentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="AttribDataType" type="xs:unsignedByte"/>
            <xs:element name="AttribId" type="xs:unsignedShort"/>
            <xs:element name="ErrorCode" type=" xs:unsignedByte "/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

F.5 XML schema for the event command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF “xmlns="http:/
/www.zigbee.org/v1.0/AF" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="EventCommand">
      <xs:annotation>
         <xs:documentation>Schema for AF event command</xs:documenta-
tion>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="AttribDataType" type="xs:unsignedByte"/>
            <xs:element name="AttribId" type="xs:unsignedShort"/>
            <xs:element name="AttribData" type="xs:anyType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

F.6 XML schema for the event response command

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.zigbee.org/v1.0/AF “xmlns="http:/
/www.zigbee.org/v1.0/AF" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Event_ResponseCommand">
      <xs:annotation>
         <xs:documentation>Schema for AF event response command</
xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="AttribDataType" type="xs:unsignedByte"/>
            <xs:element name="AttribId" type="xs:unsignedShort"/>
            <xs:element name="ErrorCode" type=" xs:unsignedByte "/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

F.7 Example KVP commands

Consider a lighting profile in which a device description for a lamp defines a cluster with an unsigned 8-bit attribute called “LampOnOff”, which has an identifier of 0x0000. This attribute can be set to either 0x00, to represent its off state, or 0xff, to represent its on state. In order for a light switch to turn on the lamp, it would need to send a command to the lamp device such as the set with acknowledgement command illustrated in Figure 91.

Bits: 8	4	4	16	8
Transaction sequence number	Command type identifier $b_3b_2b_1b_0$	Attribute data type $b_3b_2b_1b_0$	Attribute identifier	Attribute data
0x42	0101	0001	0x0000	0xff

Figure 91 Example of a set with acknowledgement command frame

As the command was a set with acknowledgement command, the lamp responds with the set response command illustrated in Figure 92.

Bits: 8	4	4	16	8
Transaction sequence number	Command type identifier $b_3b_2b_1b_0$	Attribute data type $b_3b_2b_1b_0$	Attribute identifier	Error code
0x42	1001	0001	0x0000	0x00

Figure 92 Example of a set response command frame

The device description for the lamp also defines another cluster with a character string attribute called “LampMoreInfo”, which has an identifier of 0x0001. In order for a PDA to set this attribute to the 4-character ASCII character string “3NSF”, it would need to send a command to the lamp device such as the set command illustrated in Figure 93.

Bits: 8	4	4	16	8	4
Transaction sequence number	Command type identifier $b_3b_2b_1b_0$	Attribute data type $b_3b_2b_1b_0$	Attribute identifier	Attribute data	
0x56	0001	1110	0x0001	0x04	0x334e5346

Figure 93 Example of a KVP set command frame

Note that the character string type requires a character length field (equal to 0x04, in this case) as the first octet of the attribute data and that no acknowledgement is required with this command.

F.8 Example MSG command

Consider an HVAC profile in which a device description for a cooling fan defines a message to set the fan speed to a number of settings. To set the fan to its third speed, a remote control device would need to send a message as illustrated in Figure 94.

Bits: 8	8	8
Transaction sequence number	Transaction length	Transaction data
0x7d	0x01	0x02

Figure 94 Example of an MSG command frame to set the speed of a fan

Consider an agricultural sensor profile in which a device description for a soil moisture sensor defines a message to configure the relative coordinates of the device using two 16-bit values. To set the coordinates on the sensor, a configuration device would need to send a message as illustrated in Figure 95.

Bits: 8	8	32
Transaction sequence number	Transaction length	Transaction data
0xe8	0x04	0x02fe3321

Figure 95 Example of an MSG command frame to set x and y coordinates of a sensor

