ZigBee IP Specification

ZigBee Public Document 13-002r00
February 2013
Sponsored by: ZigBee Alliance

Accepted by	This document has been accepted for release by the ZigBee Alliance Board of Directors
Abstract	The ZigBee IP Specification describes the protocol infrastructure and services available to applications operating on the ZigBee IP platform.
Keywords	ZigBee IP, 802.15.4, IPv6, 6LoWPAN

February 2013

This page is intentionally blank

Notice of use and disclosure

ZigBee Alliance, Inc.
2400 Camino Ramon, Suite 375
San Ramon, CA 94583, USA

This page is intentionally blank

Participants

The following is a list of those who were members of the ZigBee Alliance Core Stack Working Group leadership when this document was released:

Don Sturek: *Chair*
Tim Gillman: *Secretary*

The ZigBee IP task group leadership was composed of the following

Joseph Reddy *(Co-Chair)*
Robert Cragie *(Co-Chair)*

Many people contributed to development of this document as well as the development of interoperable platforms incorporating this specification. The following are especially acknowledged.

Skip Ashton	Richard Kelsey
Catalina Costin	Owen Kirby
Michael Cowan	Leslie Mulder
Ralph Droms	Colin O'Flynn
Paul Duffy	Matteo Paris
Daniel Gavelle	Kundok Park
Vlad Gherghisan	Robby Simpson
Juha Heiskanen	Dario Tedeschi
Tom Herbst	Mads Westergreen
Parag Kanekar	

This page is intentionally blank

Table of Contents

List of Figures

List of Tables

Change history

Table 1 shows the change history for this specification.

Table 1: Document revision change history

Revision	Description
00	Original version
01	Updated the list of technical editors only
02	Removed section on Backward Compatibility per Houston Members Meeting discussion; Removed section on network layer error correction; Removed section on Mesh to Bus routing; Updated list of technical editors
03 to 08	Miscellaneous internal editing team revisions leading to draft specification version 0.7
09	Internal editing team revision
10	Updated document based on comments received in the first letter ballot. The details of the comments and resolutions are available in the comment resolution database (ZigBee document number 105652)
11	Interim working copy with changes to sections on RPL, Node bootstrapping, PANA and Key update
12	Interim working copy incorporating contributions in Security, RPL, Sleepy node and MLE sections
13	Interim working copy incorporating changes from editing meeting. with changes to sections on beacon payload, network selection, node diagnostics, bootstrapping etc.
14	Interim working copy incorporating changes from editing meeting.
15	Updated based on comments received in informal review (see 12-0235 for comments and resolution)
16	Updated based on comments received in informal review (see 12-0235 for comments and resolution)
17	Updated based on comments received in informal review (see 12-0235 for comments and resolution)
18	Previous doc with updated revision and all “track changes” accepted
19	Updated based on comments received in letter ballot 2 (see 10-5934 for comments and resolution)
20	Accepted all “track changes” in previous version and converted to pdf.

21	Updates based on some comments received in the 0.9 letter ballot. See 12-0323-01 for details
22	Accepted previous changes and added text from v0.9 comment resolution. See 12-0323-03 for details
23	Additional comment resolution. See 12-0323-04 for details
24	Editorial comments resolved. See 12-0323-05 for details
25	Editorial updates – capitalizations for all keywords
26	Editorial updates – references
27	Updates to Multicast forwarding section to use new MPL draft and added clarification on Key pull behavior based on reflector email
28	Updates based on comments during SVE event
29	Accepted all changes in previous revision and converted to pdf
30	Fixed error (incomplete resolution of previous comment) in line 1710
31	Accept changes and convert to pdf
32	Formatted for release and document number updated to 12-0572-10

1 References

The following standards and specifications contain provisions, which through reference in this document constitute provisions of this specification. All the standards and specifications listed are normative references. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards and specifications indicated below.

[SE-TRD]	Smart Energy Profile Technical Requirements Document, ZigBee document 095449
[802.15.4]	IEEE Standards 802, Part 15.4-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)
[ECDP]	Standards for Efficient Cryptography Group, "SEC 2 - Recommended Elliptic Curve Domain Parameters", www.secg.org
[IANA]	Internet Assigned Numbers Authority Protocol Registries http://www.iana.org/protocols/
[MLE]	Mesh Link Establishment, IETF draft-kelsey-intarea-mesh-link-establishment-04
[MPL]	Multicast Protocol for Low Power and Lossy Networks (MPL), IETF draft-ietf-roll-trickle-mcast-03
[RFC 768]	User Datagram Protocol (UDP), IETF RFC 768
[RFC 793]	Transmission Control Protocol (TCP), IETF RFC 793
[RFC 2119]	Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119
[RFC 2460]	Internet Protocol, Version 6 (IPv6) Specification, IETF RFC 2460
[RFC 3748]	Extensible Authentication Protocol (EAP), IETF RFC 3748
[RFC 4007]	IPv6 Scoped Address Architecture, IETF RFC 4007
[RFC 4193]	Unique Local IPv6 Unicast Addresses, IETF RFC 4193
[RFC 4279]	Pre-Shared Key Ciphersuites for Transport Layer Security (TLS), IETF RFC 4279
[RFC 4291]	IP Version 6 Addressing Architecture, IETF RFC 4291
[RFC 4443]	Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, IETF RFC 4443
[RFC 4492]	Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), IETF RFC 4492
[RFC 4861]	Neighbor Discovery for IP version 6 (IPv6), IETF RFC 4861
[RFC 4862]	IPv6 Stateless Address Autoconfiguration, IETF RFC 4862
[RFC 4944]	Transmission of IPv6 Packets over IEEE 802.15.4 Networks (6LoWPAN), IETF RFC 4944
[RFC 5116]	An Interface and Algorithms for Authenticated Encryption, IETF RFC 5116

[RFC 5191]	Protocol for Carrying Authentication for Network Access (PANA), IETF RFC 5191
[RFC 5216]	The EAP-TLS Authentication Protocol, IETF RFC 5216
[RFC 5246]	The Transport Layer Security (TLS) Protocol Version 1.2, IETF RFC 5246
[RFC 5288]	AES Galois Counter Mode (GCM) Cipher Suites for TLS, IETF RFC 5288
[RFC 5289]	TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM), IETF RFC 5289
[RFC 5487]	Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode, IETF RFC 5487
[RFC 6282]	Compression Format for IPv6 Datagrams in 6LoWPAN Networks, IETF RFC 6282
[RFC 6345]	Protocol for Carrying Authentication for Network Access (PANA) Relay Element, IETF RFC 6345
[RFC 6550]	RPL: IPv6 Routing Protocol for Low power and Lossy Networks, IETF RFC 6550
[RFC 6553]	RPL Option for Carrying RPL Information in Data-Plane Datagrams, IETF RFC 6553
[RFC 6554]	An IPv6 Routing Header for Source Routes with RPL, IETF RFC 6554
[RFC 6655]	AES-CCM Cipher Suites for Transport Layer Security (TLS), IETF RFC 6655
[RFC 6719]	The Minimum Rank with Hystersis Objective Function, IETF RFC 6719
[RFC 6775]	Neighbor Discovery Optimization for IPv6 over Low Power Wireless Personal Area Networks (6LoWPANs), IETF RFC 6775
[RFC 6786]	Encrypting the Protocol for Carrying Authentication for Network Access (PANA) Attribute-Value Pairs, IETF RFC 6786
[TLS-ECC-CCM]	AES-CCM ECC Cipher Suites for TLS, IETF draft-mcgrew-tls-aes-ccm-ecc-05

2 Definitions

Authentication Server	The server implementation that is in charge of verifying the credentials of a ZIP node that is requesting the network access service. The AS is usually hosted on the ZIP Coordinator but may also be on a dedicated node on the access network, or on a central server in the Internet.
6LBR	6LoWPAN Border Router, as defined in [RFC 6775].
6LR	6LoWPAN Router, as defined in [RFC 6775].
DODAG Root	As defined in [RFC 6550].
Enforcement Point	The access control implementation that is in charge of allowing access (data traffic) of authorized clients while preventing access by others.
Global address	As defined in [RFC 4862].
Host	Any node that is not a router.
Link local address	As defined in [RFC 4862].
Node	A device that implements the protocols specified in this document.
Router	A node that forwards network layer packets not explicitly addressed to itself.
RPL	An IPv6 routing protocol designed for use in low-power and lossy networks and specified in IETF RFC 6550.
RPL Instance	As defined in [RFC 6550].
RPL Root	As defined in [RFC 6550].
ZIP	ZigBee IP Protocol, as defined in this document
ZIP Coordinator	A ZigBee IP node that is responsible for starting and maintaining a ZigBee IP network. This node implements the functionalities of a 802.15.4 PAN Coordinator, 6LoWPAN LBR, RPL Root, PAA and Authentication Server.
ZIP Host	Any ZigBee IP node that is not a ZIP router.
ZIP Node	A device that implements the protocol suite specified in this document.
ZIP Router	A ZigBee IP node that forwards network layer packets not explicitly addressed to itself.

3 Acronyms and abbreviations

AES	Advanced Encryption Standard
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
DAD	Duplicate address detection. An algorithm used to ensure the uniqueness of an address in an IP network. See [RFC 6775]
DAG	Directed Acyclic Graph. See [RFC 6550]
DODAG	Destination Oriented DAG. See [RFC 6550]
EAP	Extensible Authentication Protocol. See [RFC 3748]
ETX	Expected Transmission Count. See [RFC 6551]
EUI	Extended Unique Identifier. See [802.15.4]
FFD	Full Function Device. See [802.15.4]
GUA	Global Unicast Address. See [RFC 4291]
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
MAC	Medium Access Control
OCP	Objective Code Point. See [RFC 6550]
OF	Objective Function. See [RFC 6550]
PAA	PANA Authentication Agent. See [RFC 5191]
PaC	PANA Client. See [RFC 5191]
PAN	Personal Area Network. See [802.15.4]
PRE	PANA Relay Element. See [RFC 6345]
ULA	Unique Local Address. See [RFC 4193]

4 Introduction

4.1 Purpose

The purpose of the ZigBee IP specification is to define a standard, interoperable protocol stack using IETF-defined networking protocols for use in IEEE 802.15.4-based wireless mesh networks.

4.2 Scope

This document contains the specification for the ZigBee IP protocol stack for use in Smart Energy Profile 2.0 applications and other ZigBee applications that may migrate to a ZigBee IP stack. This specification is designed to meet the technical requirements described in the Smart Energy Profile 2.0 Technical Requirements Document [SE-TRD].

This specification utilizes protocols defined in subordinate specifications produced by the IETF and the IEEE. As such, it does not seek to describe any of the protocols in detail. Rather, it calls out the specific set of protocols that must be supported as well as any relevant operational modes and configurations. Any requirements specified as mandatory in the subordinate specifications that are not necessary to be supported in ZigBee IP shall be identified in this document. Any requirements specified as optional in the subordinate specifications that are necessary to be supported, shall be identified in this document.

4.3 Overview

The ZigBee IP protocol stack is illustrated in the figure below.

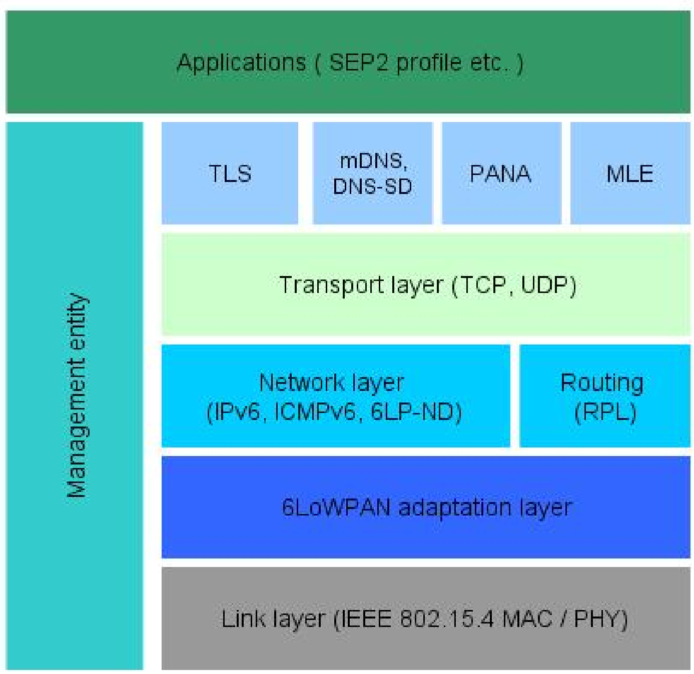

Figure 1: ZigBee IP protocol layers

The link layer provides the following services

- Discovery of IEEE 802.15.4 PAN's within radio range.

- Frame transmissions with a maximum MAC payload size of 118 bytes. Actual MAC payload in each frame can vary depending on addressing mode and security options.
- Support for frame transmissions to sleeping devices using frame buffering and polling.
- Frame security including encryption, authentication and replay protection. Note that key management is not performed at this layer.

The 6LoWPAN adaptation layer provides the following services

- Header compression and decompression for IPv6 and UDP headers.
- Fragmentation and reassembly of IPv6 packets that exceed the maximum payload size available in the link layer frame.

The Network layer provides the following services

- IPv6 addressing and packet framing.
- ICMPv6 messaging.
- Router and Neighbor discovery.
- IPv6 stateless address auto configuration and Duplicate address detection (DAD).
- Propagation of 6LoWPAN configuration information.
- Route computation and maintenance using RPL protocol.
- IPv6 packet forwarding.
- IPv6 multicast forwarding within the subnet.

The Transport layer provides the following services

- Guaranteed and non-guaranteed packet delivery service.
- Multiplexing of packets to multiple applications.

The Management entity is a conceptual function that is responsible for invoking and managing the various protocols in order to achieve the desired operational behavior by the node. It is responsible for

- Node bootstrapping procedure.
- Node power management.
- Non-volatile storage and restoration of critical network parameters.
- Authentication and network access control using PANA protocol.
- Network-wide key distribution using PANA protocol.
- Propagation of network configuration parameters using MLE protocol.

4.4 Document Organization

The rest of the document is organized as follows. Section 5 contains the ZigBee IP protocol specification. It describes the various IEEE and IETF standard protocols that must be supported by a ZigBee IP implementation along with details on the mandatory and optional features within each of them. Section 6 describes the functional behavior of a ZigBee IP node during various stages of network operation. Section 7 specifies the values for the various parameters that are defined in earlier sections. Section 8 contains informative material and examples of protocol message exchanges that may be useful to implementers of this specification.

4.5 Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

5 Protocol Specification

5.1 Physical Layer

A ZigBee IP node MUST support at least one physical interface conforming to one of the PHY specifications defined in the IEEE 802.15.4-2006 standard [802.15.4].

This specification describes the protocol operation on a single physical interface in a node. The support of multiple physical interfaces in a node is out of scope.

5.2 MAC Layer

A ZigBee IP node MUST implement the IEEE 802.15.4-2006 MAC specification [802.15.4]. A ZIP Host MUST implement the RFD (reduced function device) functionality while a ZIP Router and ZIP Coordinator MUST implement the FFD (full function device) functionality.

A ZIP Node is not required to implement all the available MAC features. Specifically, the periodic-beaconing mode of operation and the guaranteed timeslots (GTS) features are not required to be supported by nodes operating in a ZigBee IP network. Association and Disassociation command frames are not required to be supported.

A ZIP node MUST support the MAC frame security features as described in section 5.10 of this document.

A ZigBee IP node MUST be able to support the 64-bit and 16-bit MAC level addressing modes. A 64-bit IEEE address (also called EUI-64, MAC address or extended address) MUST be configured in each device at manufacture time. This address is globally unique and it is expected that this address is fixed during the lifetime of the device. A 16-bit short address MUST be assigned to each node after it has completed network admission. This address is unique within that particular IEEE 802.15.4 PAN.

5.3 6LoWPAN Adaptation Layer

The 6LoWPAN adaptation layer is defined by standards produced by the 6LoWPAN Working Group of the IETF.

The encapsulation of IPv6 packets in 802.15.4 frames MUST be performed as specified in [RFC 4944] and [RFC 6282]. The mesh addressing header is not required to be supported as ZigBee IP does not use the link-layer mesh-under routing configuration described in [RFC 4944] and instead rely on the route-over configuration.

5.3.1 6LoWPAN Fragmentation

The 6LoWPAN fragmentation scheme defined in [RFC 4944] MUST be supported.

The fragments composing a single IP datagram MUST be transmitted in order of increasing datagram_offset. In addition, the transmission of fragments of one datagram MUST not be interleaved with any other datagrams, fragmented or otherwise, to the same destination (while [RFC 4944] allows fragments and packets to be sent in any order, having fragments arrive in order and not interleaved simplifies both reassembly and detection of missing fragments. The MAC/PHYs used for ZigBee IP do not themselves reorder packets, so the above restrictions are sufficient to ensure in-order arrival.)

The link MTU for the 6LoWPAN interface MUST be set to 1280 octets (see 5.4.3 for exception).

5.3.2 Header Compression

The 6LoWPAN header compression scheme defined in [RFC 6282] MUST be supported by a ZigBee IP node. A ZigBee IP node MUST support all compression modes defined in [RFC 6282]. When transmitting an IPv6 packet, the most effective compression scheme SHOULD be used in order to minimize the size of the transmitted packet. A node SHOULD be able to receive an IPv6 packet with any or no header compression as long as the header is encoded using the format defined in [RFC 6282].

[RFC 6282] allows the use of pre-defined context identifiers for the purpose of compressing IPv6 addresses. These context identifiers are defined at the 6LBR and conveyed to the other nodes in the network via router advertisements [RFC 6775].

The 6LBR in a ZigBee IP network MUST NOT define more than MIN_6LP_CID_COUNT context identifiers for purposes of IP header compression. It MUST define the default context identifier (context zero) and set its value to the IPv6 prefix assigned to the 6LoWPAN, as defined in section 5.4.

All other ZIP nodes MUST support the configuration and use of at least MIN_6LP_CID_COUNT context identifiers for purposes of IPv6 header compression.

5.3.3 Neighbor Discovery

The neighbor discovery protocol MUST be implemented as defined in 6LoWPAN neighbor discovery specification [RFC 6775].

A ZigBee IP node MUST support the optional mechanisms defined in [RFC 6775] for multihop distribution of prefix and context information.

A ZigBee IP node MUST support the optional mechanisms defined in [RFC 6775] for multihop duplicate address detection.

A ZigBee IP node SHOULD suppress neighbor unreachability probes as the upper layer protocols specified in later sections include periodic packet transmissions that verify the bidirectional reachability of neighbor nodes as well as detecting new neighbor nodes. However, a node MUST respond appropriately to a neighbor unreachability probe.

5.4 Network Layer

A ZigBee IP node MUST support the IPv6 protocol [RFC 2460].

A ZigBee IP node is not required to support the Authentication Header (AH) and the Encapsulating Security Payload (ESP) IPv6 extension headers and this mode of operation is not described in this specification

A ZigBee IP node is not required to support the Fragment IPv6 extension header.

A ZigBee IP node MUST support the ICMPv6 protocol [RFC 4443]. Nodes MUST support the ICMPv6 error messages as well as the echo request and echo reply messages.

5.4.1 IP Addressing

All ZigBee IP nodes MUST support the IPv6 addressing architecture specified in [RFC 4291].

A ZigBee IP network will be assigned one or more /64 prefix(es), which will be announced as the prefix(es) throughout the entire 6LoWPAN (see [RFC 6775]). These prefix(es) MAY be either ULA [RFC 4193] or GUA prefix(es) and a node MUST be capable of supporting atleast MIN_6LP_PREFIX number of prefixes. For consistency with [ND], [RFC 4944] and other standards, the 6LoWPAN prefix(es) MUST always be a /64. A 6LoWPAN node can use either its EUI-64 address or its 16-bit short address to derive the interface identifier, as defined in section 6 of [RFC 4944]. When using the 16-bit short address to construct the interface identifier, the method specified in [RFC 6282] MUST be followed. When applied to header compression modes that are based on the 16-bit short address, the /64 prefix from the default context and the additional 48 bits that convert the 16-bit short address to a 64-bit IID are elided from the compressed address.

A ZigBee IP node MUST configure its IEEE 802.15.4 interface with at least the following addresses:

- A 128-bit link-local IPv6 address configured from the EUI-64 of the node as interface identifier using the well-known link-local prefix FE80::0/64 as described in [RFC 4862] and [RFC 4944]. When this type of address is compressed using [RFC 6282], it MUST be considered stateless compression. This type of address is known in its abbreviated form as LL64.

- A 128-bit link-local IPv6 address configured from the interface identifier based on the 16-bit short address of the node using the well-known link-local prefix FE80::0/64 as described in [RFC 4862] and [RFC 4944]. When this type of address is compressed using [RFC 6282], it MUST be considered stateless compression. This type of address is known in its abbreviated form as LL16
- One or more 128-bit unicast IPv6 addresses. The interface identifier used for address configuration is based on the 16-bit short address of the node. The prefix is the ULA or GUA prefix obtained from the 6LoWPAN Prefix information option (PIO) in the Router Advertisement (see [RFC 6775]). If multiple global prefixes are advertised, the node MAY choose to configure addresses with any or all of them based on local node policy. When this type of address is compressed using [RFC 6282], it MUST be considered stateful, context based compression. This type of address is known in its abbreviated form as GP16.

In addition, all nodes MUST join the appropriate multicast addresses as required by [ND].

DAD MUST NOT be performed on addresses configured from a EUI-64 interface identifier, as RECOMMENDED in [RFC 6775]. The GP16 address configured from the 16-bit short address MUST be tested for uniqueness using the DAD mechanism as defined in [RFC 6775].

5.4.2 Routing Protocol

All ZigBee IP routers MUST implement the RPL routing protocol [RFC 6550]. RPL establishes a destination oriented directed acyclic graph (DODAG) toward a Root node, called the DODAG Root. Packets are directed up the DODAG toward the Root using this graph. Packets are directed from the Root down the DODAG using routes established from Destination Advertisement Object (DAO). The following subsections describe how RPL is used in ZigBee IP to ensure compatibility between devices.

A ZigBee IP network MAY run multiple instances of RPL concurrently. Only global instances SHOULD be used. The 6LBR node MUST start a RPL instance. Other ZigBee IP routers MAY start their own RPL instance if they offer connectivity to external network or if they are administratively configured to do so. In this case, the RPL instance identifier SHOULD be selected so that it does not conflict with existing identifiers. This means that the router SHOULD first join the network and discover existing RPL instances before starting its own. The presence of DIOs with different DODAG id fields but equal instance id fields indicates a duplicate instance id. If a DODAG root detects an instance id conflict with its instance, it SHOULD reform the DODAG using a different instance id.

A ZigBee IP router MUST be capable of joining at least MIN_RPL_INSTANCE_COUNT RPL instances and SHOULD join all RPL instances that are available in the network subject to its memory constraints.

If a node loses connectivity to a RPL instance (that is, it cannot find a parent with finite rank) for over RPL_INSTANCE_LOST_TIMEOUT seconds, it SHOULD delete the instance. This may happen, for example, if the root of the instance is replaced.

Each DODAG root may be configured to include zero or more prefixes in the Route Information Option (RIO). Note that if the root wishes to advertise the default route (prefix 0::), it MUST include it in an RIO. The absence of any RIO prefixes indicates that the DODAG can route packets only to the root node. If the DODAG Root is also the Authoritative Border Router (as defined in [RFC 6775]), it MUST include the PIO information in both the RPL DIO packet as well as the Router Advertisement packet.

In a ZigBee IP network, a RPL Instance MUST contain a single DODAG with a single Root. A DODAG root MUST always grounded. Floating DODAGs MUST NOT be used.

RPL control messages are sent using “unsecured” RPL security mode. Link layer security is used to meet the security requirements.

In a ZigBee IP network, only the non-storing RPL mode of operation is used. In non-storing mode, all downward routes are managed by the DODAG root as source routes. Routers send DAO messages containing downward route information directly to the root, with the DAO-ACK (‘K’) flag enabled. DAO messages are not delayed at each hop as described in [RFC 6550] section 9.5. DAO messages SHOULD be jittered by the originating router to avoid multiple nodes sending simultaneously to the root. Multicast DAO messages are not used in a ZigBee IP network.

Every non-root router SHOULD be capable of having at least RPL_MIN_DAO_PARENT parents per DODAG, to be used for upward routing by the router itself, and downward routing by the root.

Metric Container and RPL Target Descriptor options MUST NOT be included in any RPL control messages.

5.4.2.1 Host Participation In RPL

A ZIP host does not participate in the RPL protocol.

5.4.2.2 Objective Function

The objective function defines the route selection objectives within a RPL Instance. The objective function is identified by the objective code point (OCP) field in the DODAG configuration option.

A ZigBee IP router MUST implement the MRHOF objective function [RFC 6719] using the ETX metric, without metric containers.

Zigbee IP routers MUST use the Mesh Link Establishment protocol [MLE] to determine the ETX of links to neighboring routers. Routers estimate the incoming delivery ratio for each neighbor in their neighbor table. The estimation method is implementation specific. The inverse of the delivery ratio is then communicated to the neighbor via the MLE Neighbor TLV. The ETX of the link is equal to the product of the forward and reverse inverse delivery ratios.

MRHOF parameters MUST be set as follows:

- MAX_LINK_METRIC: 16 * MinHopRankIncrease.
- MAX_PATH_COST: 256 * MinHopRankIncrease.
- MIN_PATH_COST: 0.
- PARENT_SWITCH_THRESHOLD: 1.5 * MinHopRankIncrease.
- PARENT_SET_SIZE: 2.
- ALLOW_FLOATING_ROOT: 0.

5.4.2.3 RPL Configuration

This section specifies the RPL configurations and the corresponding RPL control messages used by ZigBee IP. Any unspecified configurations are used as defined in [RFC 6550].

The DODAG root is authoritative for setting some information through DIO and the information is unchanged during propagation toward leaf nodes. This information is described below:

1. RIO(s)
2. DODAG configuration option
3. PIO(s), with the exception that if the ‘R’ flag is set, the last two bytes of the IPv6 address (the link layer short address) inside Prefix field will change
4. RPLInstanceID
5. DODAGID
6. DODAGVersionNumber
7. Grounded flag

8. Mode of operation field

5.4.2.3.1 DODAG Information Solicitation (DIS) Frame Format

The DIS messages MAY include the Pad1, PadN or Solicited Information options.

A ZIP router MAY transmit a DIS message with the Solicited Information option and the InstanceID predicate in order to limit the DIO responses to a specific RPL Instance.

5.4.2.3.2 Multicast DODAG Information Object (DIO) Frame Format

The multicast DIO message contains the DIO base object and the RIO objects.

The configuration of the DIO base is as follows:

- The RPLInstanceID SHOULD be set to a global Instance with a value in the range of [0x00, 0x7F]
- The Version Number SHOULD be initialized to a value of 0xF0
- The Grounded (G) flag of the DIO MUST be always set. ZIP nodes MUST NOT create floating DODAGs.
- The Mode of Operation (MOP) field in the DIO MUST be set to 0x01. This indicates the non-storing mode in RPL.
- The DODAGPreference field SHOULD be set to 0. ZIP routers are not required to implement DODAG preference based on this field.
- The Destination Advertisement Trigger Sequence Number (DTSN) - The root node increments the DTSN field of the DIO when it wishes to receive fresh DAO messages from the network without incrementing the DODAG version number. ZIP routers MUST set their DTSN value to the same value as their parent router and update it whenever the parent router updates its value. This way the Root node can increment the value in its DTSN field and propagate that change through the entire DoDAG.

The configuration of the RIO is as follows:

- The Prefix Length SHOULD be set to the length of the prefix for which the route is being advertised
- The Route Preference (Prf) value SHOULD be set to 0 (medium) preference or administratively configured.
- The Prefix SHOULD be set to the value for which the route is being advertised

RPL allows the root to include multiple RIO options in a DIO frame to advertise external routes that are reachable through the root. A ZIP node operating as a RPL root SHOULD limit the number of RIO options included in the DIO packet to RPL_MAX_RIO. This is to ensure that all ZIP routers can process the necessary route information. Similarly, a RPL root SHOULD limit the number of PIO options included in the DIO packet to RPL_MAX_PIO.

5.4.2.3.3 Unicast DODAG Information Object (DIO) Frame Format

The unicast DIO message contains DIO base, RIO(s), PIO(s) and DODAG configuration option. The DIO base and the RIO used in unicast messages have the same format as in multicast messages.

The configuration of the PIO is as follows:

- The Prefix length MUST be set to 0x40, indicating a 64-bit prefix.
- The 'L' flag (On-link flag) MUST NOT be set (see [RFC 6775] section 6.1)
- The 'A' flag (Autonomous address-configuration flag) MUST be set if the prefix can be for stateless address autoconfiguration.

- The ‘R’ flag (Router address flag) MUST be set if the node has configured an address with this prefix. Otherwise, it MUST NOT be set.
- The Prefix field MUST contain the routable IPv6 address of the source node

The configuration of the DODAG configuration option is as follows:

- The Authentication Enabled (A) flag MUST NOT be set. ZigBee IP does not use RPL security and instead relies on MAC layer security.
- The Path Control Size (PCS) field MUST be set to a value of atleast 1. This controls the number of DAO parents and downward routes that are configured for a ZIP node.
- The trickle parameters that govern the DIO transmission SHOULD be set by the RPL root. The parameters SHOULD be set to balance the amount of traffic generated by the trickle timer reset against the joining startup time. The following parameter values are RECOMMENDED:
    - DIOIntervalDoublings value SHOULD be set to 12
    - DIOIntervalMin value SHOULD be set to 9
    - DIORedundancyConstant value SHOULD be set to 3

    The ZIP routers MUST configure their internal DIO trickle timer parameters based on the incoming DODAG configuration option and MUST NOT hardcode the above recommended values.
- The MaxRankIncrease field SHOULD be set to non-zero value. MaxRankIncrease is used to configure the allowable rank increase in support to local repair. If it is set to zero, local repair is disabled. A typical value for this field would be about 16 and a larger value SHOULD be in networks with more hops.
- The MinHopRankIncrease field SHOULD be set to 0x80
- The Objective Code Point (OCP) MUST be set to the assigned value in [RFC 6719]

5.4.2.3.4 Destination Advertisement Object (DAO) Frame Format

A Unicast DAO request is sent to the DODAG root node in order to establish the downward routes. This request is composed of DAO base, RPL target option(s) and Transit information option(s).

The configuration of the DAO base is as follows:

- The RPLInstanceID field MUST be a global RPLInstanceID which MUST be in the range [0x00, 0x7F] (inclusive).
- The ‘K’ flag SHOULD be set. This flag indicates that the DODAG root is expected to send a DAO-ACK back.
- The ‘D’ flag MUST be cleared as local RPLInstanceIDs are not used.
- The DAOSequence SHOULD be initially set to 0xF0 and incremented in a “lollipop” fashion afterwards. A node SHOULD increment the DAO sequence number when it retransmits a DAO due to lack of DAO-ACK.

At least one RPL target option MUST be present in the DAO request. RPL target option is used to inform the DODAG root node that a route to the target IPv6 address exists.

The configuration of the RPL target option is as follows:

- The Prefix Length SHOULD be set to 0x80 because an IPv6 address is present in Target Prefix
- The Target Prefix SHOULD be set either to the IPv6 address of the ZIP router that is sending the DAO router or to the IPv6 address of a ZIP host that is directly reachable by that router.

The Transit information option is used to indicate the DODAG parents to the DODAG root. The configuration of the Transit information option is as follows:

- The External (E) flag MUST be set to zero when the Target prefix contains the IPv6 address of the ZIP router that is sending the DAO packet. Otherwise, it MUST be set to one.
- The Path Control field is used for limiting the number of DODAG parents included in a DAO request and for setting a preference among them
- The Path Sequence SHOULD be updated for each new DAO packet.
- The Path Lifetime MUST be set to the lifetime for which the DAO parent is valid. It MUST be set to zero when the ZIP router wants to delete an existing DAO parent from its downward routing table entry at the DODAG Root.
- A single Parent Address MUST be present in Transit information option and it MUST contain the IPv6 address of the DODAG parent or the IPv6 address of the node generating the request when a DAO is sent on behalf of the host. Multiple parent addresses MAY be conveyed using multiple Transit options.

The RPL Root determines the freshness of the routing information received through a DAO packet before using it to update its source route entries. When the DAO carries route information for Host nodes, indicated by the setting of the 'E' flag, the Root MUST use time-of-delivery as the freshness indicator. That is, a DAO that arrives latter in time is assumed to contain more recent route information. Otherwise, the Root is free to determine the freshness using a combination of time-of-delivery, DAO sequence and Path sequence values.

5.4.2.3.5 Destination Advertisement Object Acknowledgement (DAO-ACK) Frame Format

The DAO-ACK request is sent from the DODAG root to the node generating the DAO request. The Root MUST acknowledge each received DAO packet irrespective of its sequence number.

The configuration of the DAO-ACK base object is as follows:

- The RPLInstanceID field MUST be set to the Instance
- The 'D' flag SHOULD be set to zero as local RPL Instances are not used
- The DODAGID field is not present when the "D" flag is zero.

5.4.3 IP Traffic Forwarding

A ZIP Router MAY forward unicast packets directly to the destination if the destination node is known to be directly reachable. Otherwise, it SHOULD forward unicast packets using the forwarding rules defined in the RPL protocol.

The RPL protocol requires that all data packets forwarded in the RPL domain MUST contain either the RPL Option [RFC 6553] or the RPL Source Route [RFC 6554] header.

The Source Routing header MAY only be inserted by the DODAG Root of the RPL Instance. The Source routing is used for P2MP (point to multipoint) traffic originating outside the DODAG and delivered through the DODAG root, and for P2P (point to point) traffic, which is forwarded from the source up the DODAG to the root and then forwarded back down the DODAG to the destination. The DODAG root will use the node specific routing information developed through information contained in the RPL DAO packets to forward IPv6 traffic to nodes in the DODAG. When the DODAG root initiates transmission or receives an IPv6 datagram with the destination address of one of the nodes in the DODAG, the root will add source routing information to the IPv6 datagram according to [RFC 6554].

The DODAG Root SHOULD insert the Source routing header directly only in the case where it is the source of the IPv6 packet and the destination is within the RPL domain (i.e., it is a ZIP router with the same prefix). In all other cases, it MUST use IPv6-in-IPv6 tunneling. The tunnel exit point MUST be set to the address of the final destination address if that node is within the RPL domain. Otherwise, it MUST be set to the parent address of the destination. The DODAG Root determines the parent address from the Transit information option in the DAO packet that has a Target option corresponding to the destination address.

A ZIP router that is originating a unicast IPv6 packet and forwarding it via RPL protocol MUST insert the RPL Option header. The header MUST be inserted using IPv6-in-IPv6 tunneling in all cases except when the destination address is the DODAG Root of the RPL Instance used by the packet. In that case, the header MAY be inserted either directly in the packet or by using IPv6-in-IPv6 tunneling. When the RPL Option header is inserted using tunneling, the tunnel exit point SHOULD be set to the next hop address along the route towards the DODAG Root. In the case where the final destination address of the packet is the DODAG Root of the RPL Instance used by the packet, the tunnel exit point MAY be set to that address.

A ZIP router that is using RPL to forward a unicast IPv6 packet originated by another node MUST insert the RPL Option header if the packet does not already contain either the RPL Option header or the Source routing header. The header MUST be inserted using IPv6-in-IPv6 tunneling. The tunnel exit point SHOULD be set to the next hop address along the route towards the DODAG Root. In the case where the final destination address of the packet is the DODAG Root of the RPL Instance used by the packet, the tunnel exit point MAY be set to that address.

A ZIP node MUST ensure that the insertion of a RPL extension header, either directly or via IPv6-in-IPv6 tunneling, does not cause IPv6 fragmentation. This is done by using a different MTU value for packets where the IPv6 header includes a RPL extension header. The RPL tunnel entry point SHOULD be considered as a separate interface whose MTU is set to the 6LoWPAN interface MTU plus RPL_MTU_EXTENSION bytes.

A ZIP Host node SHOULD forward packets to its default parent router (this is the router through which the Host has registered its address, as described in [RFC 6775]). If the parent router determines that the packet needs to be forwarded using the RPL forwarding rules, it inserts the necessary RPL extension header following the rules described above.

5.4.4 Multicast Forwarding

The multicast scope value of 3 [RFC 4291] is defined as a "subnet-local" scope that comprises of all the links and interfaces of all ZIP nodes within a single network. Thus a ZIP network forms a subnet-local multicast zone [RFC 4007] with scope value of 3.

All ZIP nodes MUST join the subnet-scope-all-nodes multicast group (`FF03:0:0:0:0:0:0:1`) and the subnet-scope-all-mpl-forwarders on their ZIP interface. All ZIP Routers MUST join the subnet-scope-all-routers multicast group (`FF03:0:0:0:0:0:0:2`) on their ZIP interface. ZIP nodes MAY join additional subnet-scope multicast groups based on administrative configuration.

ZIP nodes use the MPL protocol [MPL] for multicast IP packet dissemination. All ZIP nodes MUST configure the ZIP interface as an MPL interface. All ZIP nodes may originate and receive MPL data messages and ZIP routers also forward MPL data messages for other nodes.

The MPL protocol requires each forwarder to participate in at least one MPL domain identified by the subnet-scope-all-mpl-forwarders group. Additionally, ZIP nodes MUST participate in the MPL domains identified by each of the subnet-scope multicast addresses that are subscribed on the ZIP interface.

ZIP nodes must configure the MPL parameters as follows:

- PROACTIVE_PROPAGATION flag MUST be set to true. This indicates that Proactive Forwarding strategy is used.
- SEED_SET_LIFETIME MUST be set to value of at least 4 seconds.
- DATA_MESSAGE_IMIN = 512ms
- DATA_MESSAGE_IMAX = 512ms
- DATA_MESSAGE_K = infinite
- DATA_MESSAGE_TIMER_EXPIRATIONS = 0 for ZIP Hosts and 3 otherwise
- CONTROL_MESSAGE_TIMER_EXPIRATIONS = 0

Note that setting the DATA_MESSAGE_TIMER_EXPIRATIONS parameter to a value of 0 on ZIP Hosts results in disabling forwarding and retransmission of MPL data messages. Similarly, setting the CONTROL_MESSAGE_TIMER_EXPIRATIONS parameter to 0 on all ZIP nodes means that MPL control messages are not transmitted in a ZIP network.

MPL data messages contain the MPL Option in an IPv6 Hop-by-Hop header. ZIP nodes MUST configure the MPL Option as follows:

- The value of the S field must be set to 1 to indicate that the seed-id is a 16-bit value.
- The value of the seed-id field must be set to the MAC short address of the node originating the MPL data message.

5.5 Transport Layer

5.5.1 Connection Oriented Service

All ZigBee IP nodes MUST support the TCP (Transmission control protocol) protocol as defined in [RFC 793].

5.5.2 Connectionless Service

All ZigBee IP nodes MUST support the UDP (User Datagram Protocol) protocol as defined in [RFC 768].

5.6 PANA

The Protocol for Carrying Authentication for Network Access [RFC 5191] MUST be used as the EAP transport for carrying authentication data between a joining Node and the Authentication Server. This section defines constraints and specifications above and beyond those specified in [RFC 5191] and [RFC 6786], which MUST be the definitive documents.

5.6.1 PRF, Integrity and Encryption Algorithms

The following algorithm identifiers MUST be used:

Algorithm	**Type**	**Value**	**Comment**
PRF	PRF_HMAC_SHA2_256	5	IKEv2 Transform Type 2
AUTH	AUTH_HMAC_SHA2_256	12	IKEv2 Transform Type 3
Encryption	AES128-CTR	1	PANA Encryption-Algorithm AVP Values

Table 2: PANA algorithm identifiers

These identifiers are assigned through the IANA (Internet Assigned Numbers Authority) protocol registries [IANA] for IKEv2 and PANA protocols.

5.6.2 Network Security Material

The PANA protocol is used to transport the network security material from the Authentication server to each authenticated node in the ZigBee IP network. This security material is used by each node to further derive encryption keys that are used to provide security for other protocols. The network security material consists of the following parameters

Parameter	**Size**	**Comment**
Network Key	16 octets	The common network wide security key that is transported using PANA by the Authentication Server to all authenticated ZIP nodes in the network
Key sequence number	1 octet	The sequence number associated with this network key
Node Auth Counter	1 octet	The value of the authentication counter to be used by each node. This parameter is unique for each node in the network.

Table 3: Network security material

The Network Key is owned and managed by the Authentication Server. Each Network Key has an associated sequence number which takes values between 1 and 255. The Authentication Server manages updates of the Network Key and associated sequence number and specifies which Network Key is active.

Additionally, the Authentication server manages an Auth Counter parameter for each node in the network. The combination of the Network Key, Key sequence number and Auth Counter is transported as a single entity by the Authentication server to each node.

5.6.3 Vendor-specific AVP's

The following ZigBee Alliance vendor-specific PANA AVP's are defined to support the transport and update of network security material. As these are vendor-specific AVP's, they shall be defined in this document and shall not be defined or referenced in any other document than this one.

The private enterprise number (PEN) for the ZigBee Alliance is 37244 [IANA].

5.6.3.1 Network Key AVP

The purpose of this AVP is to securely transport the network security parameters from the Authentication server to each node.

```
struct PANAAVP {
  uint16 code = 1; /* ZigBee Network Key */
  uint16 flags = 1; /* Vendor-specific */
  uint16 length = 18;
  uint16 rsvd = 0;
  uint32 vendor_id = 37244; /* ZigBee Alliance PEN */
  struct ZBNWKKEY {
    uint8 nwk_key[16]; /* NwkKey */
    uint8 nwk_key_idx; /* NwkKeyIdx */
    uint8 auth_cntr; /* AuthCntr */
  };
  struct AVPPad {
    uint8 bytes[2];
  };
};
```

5.6.3.2 Key Request AVP

The purpose of this AVP is to allow a PaC to request the PAA to transport either a new network key or an updated auth counter for the current network key. Support for this AVP is OPTIONAL for ZIP nodes.

```
struct PANAAVP {
  uint16 code = 2; /* ZigBee Key Request */
  uint16 flags = 1; /* Vendor-specific */
  uint16 length = 2;
  uint16 rsvd = 0;
  uint32 vendor_id = 37244; /* ZigBee Alliance PEN */
  struct ZBNWKKEYREQ {
      uint8 nwk_key_req_flags; /* request flags */
      uint8 nwk_key_idx; /* NwkKeyIdx */
  };
  struct AVPPad {
      uint8 bytes[2];
  };
};
```

5.6.4 Timeouts

Retransmission timers are specified in Section 9 of [RFC 5191]. The following values SHOULD be used:

Parameter	Value	Comment
PCI_IRT	1 sec	Initial PCI timeout.
PCI_MRT	120 secs	Max PCI timeout value.
PCI_MRC	5	Max PCI retransmission attempts.
PCI_MRD	0	Max PCI retransmission duration.
REQ_IRT	15 sec	Initial Request timeout.
REQ_MRT	30 secs	Max Request timeout value.
REQ_MRC	5	Max Request retransmission attempts.
REQ_MRD	0	Max Request retransmission duration.

Table 4: PANA timeout values

5.7 EAP

The Extensible Authentication Protocol (EAP) is an authentication framework which supports multiple authentication methods (known as EAP methods). This section defines constraints and specifications above and beyond those specified in [RFC 3748].

The ZIP Coordinator MUST function as an EAP authenticator while all other nodes MUST function as an EAP peer.

5.7.1 EAP Identity

The EAP Request/Identity message is OPTIONAL. However the EAP Response/Identity MUST be supported by the client in response to the Request/Identity. The EAP identity given in response to an EAP Request/Identity MUST be “anonymous” to prevent any information about the EAP client/peer being revealed in cleartext during the initial transactions of the authentication. The string MUST NOT be null-terminated, i.e. shall have a length of 9 octets.

5.8 EAP-TLS

EAP-TLS represents a specific type of EAP method (see [RFC 3748]). This section defines constraints and specifications above and beyond those specified in [RFC 5216].

5.8.1 EAP Key Expansion

[RFC 5216] specifies the key expansion for derivation of keying and IV material. This section defines the specific expansion for the cipher suites used and the use of the outputs.

```
MSK = PRF(master_secret, "client EAP encryption", ClientHello.random +
ServerHello.random);
```

Note the string "client EAP encryption" MUST NOT be null-terminated, i.e. it shall be a length of 21 octets.

The PRF function MUST be iterated twice as `MSK` length is 64 octets and the hash output from SHA-256 is only 32 octets. The EMSK MUST NOT be used and therefore does not need to be generated.

`MSK` MUST be used as specified in [RFC 5191] and [RFC 6786] to generate PANA_AUTH_KEY and PANA_ENCR_KEY.

5.8.2 EAP-TLS Fragmentation

It is mandatory for EAP-TLS peers and servers to support fragmentation as described in [RFC 5216] section 2.1.5. EAP peers and servers MUST support EAP-TLS fragmentation. When performing EAP-TLS fragmentation, ZIP nodes MUST ensure that the maximum size of TLS data in a single EAP packet is not greater than EAP_TLS_MTU octets. However ZIP nodes MUST still be capable of receiving EAP packets up to the maximum MTU size as they may originate from outside the ZigBee IP network. As the EAP fragments are transported over a reliable lower layer (PANA), retransmission at the EAP layer SHOULD be disabled as described in section 4.3 of [RFC 3748].

5.9 TLS

Transport Layer Security version 1.2 (TLS) is used in conjunction with PANA, EAP and EAP-TLS to provide authentication between a joining node and the Authentication Server. This section defines constraints and specifications above and beyond those specified in [RFC 5246].

5.9.1 TLS Cipher Suites

5.9.1.1 TLS-PSK Cipher Suite

The PSK cipher suite MUST be TLS_PSK_WITH_AES_128_CCM_8 as defined in [RFC 6655].

[RFC 4279] specifies the generation of the master secret from the pre-master secret. The specific generation for the PSK cipher suite used is described below:

```
master_secret = PRF(pre_master_secret, "master secret", ClientHello.random +
ServerHello.random);
```

Note the string “master secret” MUST NOT be null-terminated, i.e. it shall be a length of 13 octets.

The PRF function MUST be iterated twice as `master_secret` length is 48 octets and the hash output from SHA-256 is only 32 octets.

5.9.1.2 TLS-ECC Cipher Suite

The ECC cipher suite MUST be TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as defined in [TLS-ECC-CCM].

The only elliptic curve to be used with this cipher suite MUST be the secp256r1 curve (also known as the NIST-P256 curve) as defined in [ECDP].

The hash algorithm to be used with this cipher suite MUST be SHA-256.

5.9.2 TLS Key Expansion

[RFC 5246] specifies the key expansion for derivation of keying and IV material. This section defines the specific expansion for the cipher suites used and the use of the outputs.

```
key_block = PRF(master_secret, "key expansion", ServerHello.random +
ClientHello.random);
```

Note the string "key expansion" MUST NOT be null-terminated, i.e. shall be a length of 13 octets.

The PRF function MUST be iterated twice as `key_block` length is 40 octets and the hash output from SHA-256 is only 32 octets:

- `client_write_MAC_key` and `server_write_MAC_key` lengths are 0 due to use of AEAD cipher
- `client_write_key` and `server_write_key` lengths are 16 bytes (SecurityParameters `enc_key_length` for [RFC 6655] and [TLS-ECC-CCM])
- `client_write_IV` and `server_write_IV` lengths are 4 bytes (SecurityParameters `fixed_iv_length` for [RFC 6655] and [TLS-ECC-CCM])
- A total of 40 bytes shall therefore be required for keying material:
    - `client_write_key` MUST be `key_block[0:15]`
    - `server_write_key` MUST be `key_block[16:31]`
    - `client_write_IV` MUST be `key_block[32:35]`
    - `server_write_IV` MUST be `key_block[36:39]`

5.9.2.1 CCM Inputs

There is only one CCM-protected record in the TLS sequence. This section defines the inputs for the AEAD cipher as defined in section 2.1 of [RFC 5116]

5.9.2.1.1 CCM Key Input

The key 'K' is `client_write_key` or `server_write_key`, depending on whether the client or server is encrypting.

5.9.2.1.2 CCM Nonce Input

The nonce is 12 octets long, as specified in [RFC 5116] and MUST be as follows:

Field	**Octets**	**Value**	**Comment**
IV data	0:3	-	Client IV or server IV depending on which is encrypting
Explicit nonce	4:11	{0,0,0,0,0,0,0,0}	Sequence counter for `Finished` handshake

5.9.2.1.3 CCM Payload Input

The payload MUST be the TLS record including the header.

5.9.2.1.4 CCM Associated Data Input

The associated data (‘A’) MUST be 13 octets long:

Field	Octets	Value	Comment
Explicit nonce	0:7	{0,0,0,0,0,0,0,0}	Sequence counter for `Finished` handshake
TLS record type	8	22	TLS handshake identifier
TLS Protocol Major	9	3	TLS 1.2
TLS Protocol Minor	10	3	TLS 1.2
TLS length MSB	11	-	Length of TLS record MSB
TLS length LSB	12	-	Length of TLS record LSB

5.10 MAC Layer Security

5.10.1 MAC Security Material

The MAC security material is derived by each node from the network security material (see Section 5.6.2) received through the PANA authentication or PANA key update process as described below

The `MAC Key` is set to the higher 16 octets of the result of

`HMAC-SHA256(Network Key,"ZigBeeIP")`

The `Key Index` is set to the `Network Key sequence number`

The initial value of `Outgoing frame counter` is set to the following

`Node Auth counter || 00 00 00`

where `||` is the concatenation operator and `Node Auth counter` is in the most significant byte position. The value of this field MUST be incremented by one each time the associated Key is used to secure a message.

The MAC security material is used to create a KeyDescriptor entry in the MAC Key Table described below. If the MAC Key Table is full, an existing entry, which is not the current active key, MUST be deleted to store the new KeyDesciptor entry.

Each ZIP node MUST maintain an attribute containing the Key Index of the current active MAC key.

When a first MAC KeyDescriptor entry is created, the active key index is set to the value of its key index. The active key index is updated subsequently through the network keys update mechanism (see Section 6.10.3).

The IEEE address-based EUI-64 MAC address of the originator, the active MAC Key and the active MAC Key Index MUST be used to secure outgoing MAC data packets.

The procedures identified in Section 7.5.8 of [802.15.4] MUST be followed for applying MAC security. The following sections indicate the mode of operation applied to MAC layer security

Note that the MAC security attribute data that is described in the subsequent sections reflects the functional specification in [802.15.4]. The organization of the data is not optimized for storage space and does not imply any particular method of implementation.

5.10.1.1Default Key Source

A participating Node (i.e. one which has joined and has been authenticated and authorized) MUST have the following set.

PIB attribute	Value	Comment
macDefaultKeySource	0xff00000000000000	Arbitrary value indicating the MAC Key. There is no need to store the actual IEEE address of the originator of the Network Key, as this may not be known

5.10.1.1.1Use of Key Identifier Mode 1

The Key Identifer Mode 1 MUST be used in conjunction with a MAC Key. This implies the use of *macDefaultKeySource*. For a global MAC Key used in conjunction with a MAC Key Index, this often means the lookup data required to be stored reduces to the MAC Key Index only as there is no need to store the value of *macDefaultKeySource* along with the Network Key Index. This mechanism is used as a convenience to limit the number of Key ID modes in [802.15.4].

5.10.2 MAC Key Table

Note that [802.15.4] separates key storage from device descriptor storage and uses handles in key storage to point to the relevant device descriptors.

A participating Node SHOULD have the following set. There is one active MAC Key and (MAC_MAX_NWK_KEYS – 1) backup MAC Keys.

PIB attribute	Value	Comment
macKeyTable	KeyDescriptor entries	One entry for the active MAC Key, additional entries for backup MAC Keys
macKeyTableEntries	MAC_MAX_NWK_KEYS	One entry for the active MAC Key, additional entries for backup MAC Keys

Table 5: Participating Node key table

A ZIP node SHOULD have the following KeyDescriptor entry set for each MAC Key:

KeyDescriptor attribute	Value	Comment
KeyIdLookupList	One KeyIdLookupList entry	Entry for this MAC Key
KeyIdLookupListEntries	1	One entry for this MAC Key

KeyDeviceList	KeyDeviceList entries	Entries in the MAC device table
KeyDeviceListEntries	(variable)	Number of entries in the MAC device table
KeyUsageList	KeyUsageList entries	One key usage for MAC data frames
KeyUsageListEntries	1	One key usage for MAC data frames
Key	(variable)	The MAC Key value

Table 6: Key descriptor

The KeyIdLookupList entry SHOULD have the following set:

KeyIdLookupDescriptor attribute	**Value**	**Comment**
LookupData	*macDefaultKeySource* || KeyIndex	Only the KeyId needs to be stored. KeyIndex is the MAC Key Index associated with this MAC Key
LookupDataSize	0x01	Size 9 octets

Table 7: KeyID lookup descriptor

A KeyDeviceList entry points to a Device Descriptor. Each KeyDeviceList entry SHOULD have the following set:

KeyDeviceDescriptor attribute	**Value**	**Comment**
DeviceDescriptorHandle	Implementation-specific	Points to the appropriate Device Descriptor
UniqueDevice	0	The key is not unique per Node
Blacklisted	Boolean	Initially set to FALSE

Table 8: KeyDeviceList entry

ZIP nodes SHOULD have one KeyUsageList entry that indicates that the MAC key is valid to be used for MAC data frames. Due to a static policy, this data can be implied and no storage is needed. The entry for MAC data frames MUST have the following set:

KeyUsageDescriptor attribute	**Value**	**Comment**
FrameType	0x02	MAC data frame

Table 9: KeyUsageList entry for MAC data frames

5.10.3 MAC Device Table

A ZIP node SHOULD have the following set. There is one DeviceDescriptor entry for each neighbor node this node is in communication with. A ZIP Router SHOULD be capable of having atleast MAC_MIN_DEV_TBL entries in the MAC device table

PIB attribute	Value	Comment
macDeviceTable	DeviceDescriptor entries	One entry for each neighbor Node this Node is in communication with
macDeviceTableEntries	(variable)	One for each neighbor Node this Node is in communication with

Table 10: MAC device table entry

The DeviceDescriptor entry for each neighbor node contains the following information

DeviceDescriptor attribute	Value	Comment
PANId	2 bytes	The PAN ID of the neighbor Node. Note this data can be implied and no storage is needed as the neighbor Node will have the same PAN ID as this Node
ShortAddress	2 bytes	The short address allocated to the neighbor Node
ExtAddress	8 bytes	The IEEE address of the neighbor Node
FrameCounter	4 bytes	The incoming frame counter of the most recently received MAC frame from the neighbor Node
Exempt	FALSE	Exempt flag irrelevant as no security policy at the MAC layer is in place, therefore this data can be implied and no storage is needed

Table 11: Participating node device descriptor entry

Note that [802.15.4] allows each of the KeyDecriptors to have a separate KeyDeviceList (list of DeviceDescriptors) that indicate the neighbor nodes that are eligible to use the particular key. A ZIP node MUST maintain the same DeviceDescriptor list, consisting of all entries in the MAC Device table, as the KeyDeviceList for each of its KeyDescriptors. This implies that each Key is valid to be used with any of the neighbor nodes.

5.10.4 Security Level Table

There is no security policy at the MAC layer. The Enforcement Point performs policing based on the specification in section 6.9.4. Therefore, all ZIP nodes MUST have the following set:

PIB attribute	Value	Comment
macSecurityLevelTable	Empty	No security policy at MAC layer

macSecurityLevelTableEntries	0	No security policy at MAC layer

Table 12: Security level table

5.10.5 Auxiliary Security Header Format

The MAC frame Auxiliary Security Header (see Section 7.6.2 of [802.15.4]) is used when a MAC frame is secured to provide additional data required for security.

5.10.5.1.1Security Control Field

The Security Control field MUST have the following values:

Field	Value	Comment
Security Level	0x05	ENC-MIC-32 is the default value for ZigBee IP link-layer security
Key Identifier Mode	0x01	Key is determined from the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header in conjunction with *macDefaultKeySource*

Table 13: Security control field

5.10.5.1.2Frame Counter Field

The Frame Counter field MUST assume the value of the *macFrameCounter* PIB attribute

5.10.5.1.3Key Identifier Field

The Key Identifier MUST be the MAC Key Index associated with the active MAC Key.

5.11 Mesh Link Establishment

The mesh link establishment protocol [MLE] provides a mechanism for nodes in a mesh network to exchange node and link properties with their neighbor nodes using the UDP protocol. Additionally, it is used to propagate link configuration information to all nodes in the ZigBee network.

All ZigBee IP nodes MUST implement the MLE protocol.

5.11.1 MLE Link Configuration

All ZIP nodes MUST support the transmission and reception of MLE link configuration messages. This includes the Link Request, Link Accept, Link Accept and Request, Link Reject messages. These messages are used to exchange the 802.15.4 interface properties and authenticate the frame counter value used by a neighbor node. These messages MAY include the following TLV options in the payload

- Source address (TLV type = 0) TLV is used by a node to communicate its 16-bit short address and 64-bit EUI-64 address of the 802.15.4 interface.
- Mode (TLV type = 1) TLV is used by a node to communicate the node capability information. The Value field MUST be 1 octet in length and formatted as shown below.

bits: 0	1	2	3	4 – 7

Reserved	FFD	Reserved	RxOnIdle	Reserved

The FFD bit MUST be set to one by all nodes that are not ZIP Hosts. The RxOnIdle bit MUST be set to one by all nodes that have the radio enabled continuously (i.e., non-sleepy nodes). The reserved bits MUST be set to zero on transmission and ignored on reception

- Timeout (TLV type = 2) TLV is used by a sleepy Host node to communicate the period of inactivity after which the Host can be considered unreachable by its parent node. A sleepy Host node SHOULD perform periodic MAC polls with period lower than this value.
- Challenge (TLV type = 3) and Response (type = 4) TLV's are used by a pair of nodes to authenticate each other's MAC frame counter values. The Value field in the Challenge TLV MUST be set to a random value that is 8 octets long.
- Replay counter (TLV type = 5) TLV is used to communicate the value of the MAC outgoing frame counter.

5.11.2 MLE Advertisement

All ZIP routers MUST support the transmission and reception of the MLE Advertisement messages. This message is used to exchange bidirectional link quality with neighbor routers. The bidirectional link quality is used to improve the quality of the RPL parent selection. Additionally, this message is used to detect changes in the set of neighboring routers.

A ZIP router that has joined the network MUST periodically transmit the MLE Advertisement message every MLE_ADV_INTERVAL.

The MLE Advertisement message MUST contain the Link quality (TLV type = 6) TLV in its payload. The neighbor records in this TLV MUST be populated with information about the nodes in the MAC device table of the originating node. The Neighbor Address field in each of the neighbor records MUST be populated with the 16-bit short address of the particular neighbor node. The P (priority) flag SHOULD be set for neighbor nodes that are part of the RPL parent set. This is to give an indication to those neighbors that they SHOULD prioritize maintenance of link with this node.

A ZIP router MUST remove the MAC device table entry corresponding to a neighbor router if it did not receive an MLE Advertisement message from that neighbor router containing a neighbor record for itself in MLE_ADV_TIMEOUT.

5.11.3 MLE Update

The ZIP coordinator MUST support origination of MLE Updates messages. All ZIP nodes MUST support the reception of the MLE Update messages.

The MLE Update message is used by the ZIP coordinator to configure the value of various link layer specific parameters in the network. The MLE Update message MUST contain only one instance of the Network Parameter TLV. This TLV MUST contain one of the following parameters

- The Channel network parameter is used to configure the channel that MUST be used by the node. It MUST contain a Value field of length 2 octets. The higher order byte of the Value field contains the channel page number and the lower order byte contains the channel number. The definition of the channel pages and channel numbers for each physical layer is in [802.15.4].
- The PAN ID network parameter is used to configure the 802.15.4 PAN identifier value that MUST be used by the nodes in the network. It MUST contain a Value field of length 2 octets that contains the new Pan Identifier. A receiving node MUST use this value to update the corresponding attribute in its MAC layer. Additionally, it MUST update the corresponding field in each of the MAC device descriptor entries (see Table 12).

- The Permit joining network parameter is used to configure the Allow Join field that SHOULD be used by the node (see Section 6.3.1). It MUST contain a Value field of length 1 octet. A ZIP Router MUST use the value of the lowest significant bit in this octet to set the value of the Allow Join parameter in its beacon payload. The other bits in the Value field MUST be set to zero on transmission and ignored on reception.
- The beacon payload network parameter is used to configure the optional fields in the beacon payload (see Section 6.3.1). The receiving node replaces all the Optional fields in its current beacon payload (see Table 16) with the contents of the Value field in this message. Since only a single Parameter TLV can be included in an MLE Update message, the ZIP Coordinator MUST ensure that it includes the complete concatenated set of all the Optional fields in a single TLV. Note that this can also be a zero length value if no Optional fields are to be included in the beacon payload.

The Network parameter TLV format contains a Delay field that is used to specify the delay value before the receiving node takes action to configure the appropriate parameter. When the parameter is either the Channel or Pan ID, the Delay field SHOULD be larger than the time it takes for the multicast packet propagation in the network. This is to ensure that all nodes receive the MLE Update packet before any of them change their parameter. A RECOMMENDED value value is 5 seconds.

ZIP nodes MAY ignore an MLE Update message with a Network Parameter TLV if a previous message with the same Parameter has not yet been acted upon. A ZIP Coordinator SHOULD ensure that successive MLE Update messages with the Network Parameter have sufficient delay between them to avoid this scenario.

In rare situations, a ZIP node may become stranded if the MLE Update message with channel or pan-id change is not received correctly by all nodes. The detection of this state on each node is out-of-scope of this specification. The recovery procedure is to perform a network discovery on all channels to find the network and then attempt a network rejoin.

MLE Update messages MUST be sent to the subnet-local all-routers or subnet-local all-nodes multicast address.

5.11.4 MLE Message Security

MLE messages are sometimes exchanged before a node has joined the network and configured secure links with its neighbor nodes. Therefore, MLE messages cannot always rely on MAC security and MLE protocol defines its own mechanism to secure its payload.

MLE Link configuration messages SHOULD be secured at the MLE layer and unsecured at the MAC layer. A Link configuration message without any security is possible during the initial phase of the node bootstrapping process when the new node has not yet acquired the security material. Subsequently, a node MUST always apply security to Link configuration messages. A ZIP node MUST ensure that an incoming Link configuration message that does not have MLE security does not change any state information for existing node entries. The sender MUST use its LL64 IP address as the source address for these packets.

MLE Link Advertisement messages MUST be secured at the MLE layer and SHOULD be sent unsecured at the MAC layer. The sender MUST use its LL64 IP address as the source address for these packets. An incoming MLE Link Advertisement packet that does not have MLE security MUST be discarded. A node SHOULD verify the freshness of MLE Link Advertisement messages from nodes with which it has configured a secure link.

MLE Update messages SHOULD not be secured at the MLE layer and MUST be secured at the MAC layer. These messages are only sent to nodes that are already part of the network, so it is possible to apply MAC layer security. Additionally, since MLE Update messages are sent to a subnet local multicast address, it MUST use MAC security or the packets would not be forwarded by the other ZIP nodes (see Section 6.9.4). Also, it not possible to use MLE security for these packets as the sending and receiving nodes may not have a secure link configured with each other unless they are in direct radio range.

5.11.5 MLE Security Material

The security material used for securing MLE packets contains the following parameters

Parameter	Size	Comment
MLE Key	16 octets	The MLE Key
Key Index	1 octet	The key index associated with this Key
Outgoing frame counter	4 octets	The value of the frame counter used to secure outgoing MLE messages with this key

Table 14: MLE security material

The MLE security material is derived by each node from the network security material (see Section 5.6.2) received through the PANA authentication or PANA key update process as described below

The `MLE Key` is set to the lower 16 octets of the result of

```
HMAC-SHA256(Network Key,"ZigBeeIP")
```

The `Key Index` is set to the `Network Key sequence number`

The initial value of `Outgoing frame counter` is set to the following

```
Node Auth counter || 00 00 00
```

where `||` is the concatenation operator and `Node Auth counter` is in the most significant byte position. This value of this field MUST be incremented by one each time the associated Key is used to secure a message.

A ZIP node MUST store the MLE security material derived from the two most recent network security materials that originated from the Authentication server. These are designated as active and alternate MLE security material.

When new security material is received originating from the Authentication server, it MUST be stored in the active location if that is empty. Otherwise, it MUST be stored in the alternate location.

Security for outgoing MLE packets MUST be applied by using the active MLE security material. Security for incoming MLE packets MUST be applied by using the MLE security material with the index that matches the index contained in the MLE auxiliary security header of the incoming message.

The security control field in the MLE message auxiliary header MUST use the same values as used for MAC layer security. The security level MUST be 5 (CCM encryption with 4 byte MAC) and the key identifier mode MUST be 1. The address used for the CCM nonce MUST be the node’s 64-bit MAC address. The frame counter MUST be the MLE outgoing frame counter.

6 Functional Description

6.1 Overview

A ZigBee IP network consists of a set of nodes that include a single ZIP Coordinator node and multiple ZIP Router and ZIP Host nodes. These nodes form a single PAN from an IEEE 802.15.4 perspective. From an IPv6 perspective, they form a single multilink subnet with a common prefix.

A ZigBee IP network is formed by the ZIP Coordinator when it starts operation as an IEEE 802.15.4 PAN coordinator and configures its IEEE 802.15.4 interface as an IPv6 router.

Once the network is created, other nodes can join the network as either ZIP Routers or ZIP Hosts, depending on their capabilities.

A new node can join the network through a three step process of network discovery, network admission and network authentication that are described in more detail in later sections. Once a node has joined the network, it may allow other nodes to join through it if it is a ZIP router. This allows the formation of a wireless mesh network that extends beyond the radio range of the ZIP Coordinator.

Nodes that are part of a ZigBee IP network share a unique network key that is used to derive other encryption keys which are then used to secure all packets at the link layer. A node acquires this key during the initial join process and it may be updated over time.

6.2 Network Formation

6.2.1 MAC Configuration

A node that is administratively configured to form a new IEEE 802.15.4 PAN will perform the following steps.

- The node conducts a MAC energy detect scan on all the preconfigured channels and identifies channels with energy level below a configured threshold. The list of channels to scan is administratively configured.
- The node conducts a MAC active scan using the standard beacon request on the channels selected in the previous step.
- The node then selects a channel with the least number of existing IEEE 802.15.4 networks.
- The node chooses a PAN Identifier that does not conflict with any networks discovered in the previous steps and also configures a randomly generated 16-bit short address.
- The node starts an IEEE 802.15.4 PAN on the selected channel and PAN Identifier.

6.2.2 IP Configuration

Upon starting a new PAN, the ZIP Coordinator shall prepare to configure the 6LoWPAN with 64-bit IPv6 global prefix(es) that are either globally unique or ULA [RFC 4193]. The prefix(es) are configured administratively or acquired from an upstream network via DHCPv6 prefix delegation or other means that are out-of-scope of this specification.

After the 6LoWPAN IPv6 prefix(es) have been configured, the ZIP Coordinator configures its IEEE 802.15.4 interface with IPv6 address(es) composed of the 6LoWPAN prefix(es) and the interface identifier created from the node 16-bit MAC short address.

Note that the ZIP Coordinator may have other interfaces besides the IEEE 802.15.4 interface and the initialization of those interfaces is out of scope of this specification.

Once the IPv6 configuration is complete, the ZIP Coordinator participates in Neighbor Discovery (ND) protocol exchanges according to [RFC 6775]. The ZIP Coordinator configures the default context identifier as the /64 prefix assigned for use throughout the 6LoWPAN. The ZIP Coordinator MAY maintain other context identifiers up to a maximum of MIN_6LP_CID_COUNT, including the default context. The ZIP Coordinator uses multi-hop prefix and context distribution as specified in [RFC 6775].

The ZIP Coordinator initiates a new RPL Instance and forms a DODAG with the operational parameters from section 5.4.2.3. As additional nodes join the network, the ZIP Coordinator begins participating in RPL protocol exchanges according to [RFC 6550].

The ZIP Coordinator initializes the PANA authentication service. The network security material (see Section 5.6.2) is generated with a random 128-bit network key and a key sequence number of one. The MAC layer and MLE layers begin to use key material derived from the network security material. Additionally the Authentication server configures the network security material disseminated through the ZigBee vendor specific Network Key AVP (see Section 5.6.3).

6.3 Network Discovery

The network discovery procedure is used to discover other IEEE 802.15.4 networks that are within radio range. For each network, the NetworkID along with some associated information is discovered in this process.

ZigBee IP nodes perform network discovery using the MAC beacon functionality.

All ZigBee IP nodes MUST be capable of transmitting the MAC beacon request command packet. The ZIP Coordinator and all ZigBee IP Routers MUST be capable of processing a beacon request command and transmitting a beacon packet in response.

To perform network discovery, a ZigBee IP node transmits a beacon request packet and collects all the responses. This is typically used by a node before starting a new network so that it can identify existing PAN identifiers and channels that are being used locally.

The network discovery process also allows a node to discover the router nodes that are in radio range. One of these routers is selected as a "parent" router for the purpose of joining the network.

6.3.1 Beacon Payload

The MAC beacon command packet is transmitted in response to a beacon request packet. The beacon packet contains an application-configurable payload field that is used to convey information about the network. A ZigBee IP router MUST configure its beacon payload field as follows:

Octets: 0	1	2 – 17	18 – variable
ZigBee protocol identifier	Control field	ZIP NetworkID	Optional fields

Table 15 : Beacon payload format

- 1- octet Protocol ID – This field MUST be set to the value of 0x02 and is used to discover ZigBee IP networks and helps to distinguish them from other 802.15.4-based networks that are located in radio range.
- 1- octet Control field – This field is used to convey information to a joining device so that it can choose an appropriate network and parent router to join. It contains multiple sub-fields that are formatted as shown below.

Bits: 0	1	2	3 – 7
Allow join	Router capacity	Host capacity	Reserved

Table 16: Beacon payload control field format

- The Allow Join bit provides a hint to new joining nodes if this network is currently allowing new nodes to join the network. It is set to value of one to indicate if this network is currently allowing new device joins. The value of this field is configured by the node management application on the ZIP Coordinator and propagated through the network using upper layer protocols (see Section 5.11.3). When a ZIP Router initially joins the network, it sets the value of this field to the same value that was used by its parent router. Subsequently, the value of this field is configured based on incoming MLE Update messages received from the ZIP Coordinator. In order to protect against loss of an MLE Update message, a ZIP Router MUST automatically set this field to zero if it has been set to one for a time greater than MLE_MAX_ALLOW_JOIN_TIME.
- The Host capacity and Router capacity bits are used to indicate if the source of the beacon packet has the capacity to accept a new Host or Router node to join the network through it. The value of these bits are set by the management entity on each node depending on its resource availability (for example, depending on availability of space in neighbor cache and MAC device table).
- The reserved bits MUST be set to zero on transmission and ignored upon reception.

- NetworkID – A 16-octet field, interpreted as ASCII characters, that is used to identify a specific network to a user. The value of this field is administratively configured on the ZIP Coordinator. Other ZIP Routers learn the value of this field from the beacon payload of the parent router through which they join the network.

- Additional OPTIONAL fields of variable length MAY be included in the beacon payload using the type-length-value format. Each optional field is formatted as shown below

Octets: 1		2 – Length
Bits: 0 – 3	4 – 7	
Length	Type	Value

Table 17: Beacon payload optional field format

- The Type subfield is 4-bits in length and identifies the type of the field. The following values are defined

Type	Description
0	A 4-octet value that can be used as a node identifier to steer a specific node to join the network. As an example, this can be set to

	the truncated hash of the device certificate.
1 – 15	Reserved

Table 18: Beacon payload optional field types

- The Length subfield is 4-bits in length and identifies the length of the Value subfield in octets.
- The Value subfield contains the value of the field.

A node MUST ignore any optional fields in the beacon payload that it does not support and continue to process the others.

6.4 Network Selection

The discovery procedure can result in discovery of multiple ZigBee IP networks in radio range. The selection of the actual network that a node MUST attempt to join is done via application-specific means. However the ZigBee IP specification provides various tools that can be used to “steer” a joining node towards the correct network that it MUST join. Some of these are described further below in this section.

- “Allow Join” flag indication – This flag is present in the beacon payload of all ZigBee IP routers. A joining node can examine this flag for all neighboring ZigBee IP routers to select the appropriate network. The routers in a network would normally set this flag to zero. When a new node is expected to join the network (as determined by application-specific means), this flag would be set to true for a specific period of time. The ZIP coordinator is responsible for propagating the value to be used in field to all routers in the network.

  Note that this parameter is only a hint to the joining nodes. The behavior of a ZIP Router does not change based on the value of this field. Specifically, if a ZIP Router has this flag set to zero, it MUST still continue to allow new nodes to join through it. Only the ZIP Coordinator may reject the join attempt.

- “User selection” - The joining node would perform a beacon scan and discover all ZigBee IP networks in its radio range. It would then display information about the networks and allow a user to select the network it SHOULD join.
- “Preconfigured information” - The joining node could be configured with information about the specific network it MUST join. This information could be, for example, the NetworkID field in the beacon payload.
- “Device identifier” – The identifier of the joining node is included in the beacon payload. This method can be used if the identity of the joining node is known to the ZIP Coordinator, so that it can propagate this information to all the routers in the network for inclusion in the beacon payload.

Note that this is not an exhaustive list and an application may implement other means for selecting the network to join. Additionally, it SHOULD be noted that these mechanisms only provide “hints” to the joining node to aid in network selection. It is expected that after selecting a network and joining it, the node would use an application level registration mechanism to validate that it has joined the correct network. If the node fails application validation, the management entity SHOULD blacklist that network and repeat the network selection and joining process.

6.5 Node Joining

After network discovery and selection, the joining node performs the bootstrap procedure to gain access to the network. The typical joining sequence is described in more detail in the following subsections.

6.5.1 Host Bootstrapping

The ZigBee IP host node bootstrapping sequence is described below.

1. The node performs the network discovery and selection procedure as described previously and selects the appropriate network to join.
2. A parent router is chosen from among the ZIP Routers that belong to the selected network. This is usually the router that has available host capacity, which is indicated by setting the Host capacity subfield in the beacon payload to 1, and whose beacon was received with the best LQI (link quality indicator).
3. The node configures its 802.15.4 MAC PAN Identifier to that of the selected target network.
4. The node configures an IPv6 link local address for its 802.15.4 interface using the LL64 address format.
5. If the node is a sleepy Host, it MUST use the MLE protocol exchange to inform the parent router that it is a sleeping device and will use MAC polling feature for Layer-2 packet transmission. This information is included in the Mode TLV option of the MLE Link request packet.

   The parent router configures MAC polling for the node's EUI-64 address. If the parent router has no capacity to accept a sleepy child node, it MUST reject the link request and the joining node SHOULD then select another parent router and continue from step 2 of this process.

   If the node is a sleepy Host, it MUST perform the MAC polling using its EUI-64 address until after it has configured a unique short address and registered it with its parent router using the MLE protocol (see step 11 in this sequence).

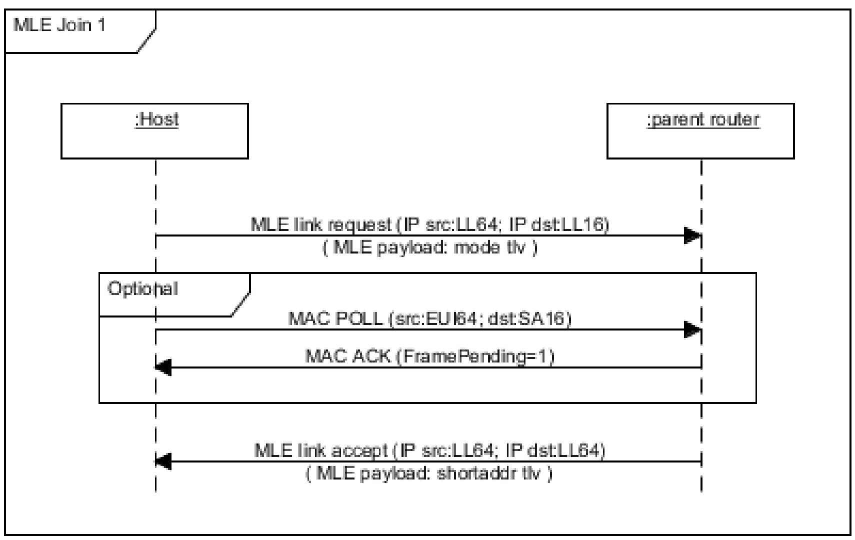

Figure 2: Join sequence – MLE 1

6. The node performs network authentication using the PANA protocol. Upon successful completion of this procedure, the node is admitted into the network and acquires the network security material. See Section 8.3.4 for an example message sequence.
7. The node performs a 3 way secured MLE handshake to synchronize frame counters with the parent router. At the end of this procedure, the node knows the parent router's frame counter and the parent router knows the node's frame counter.

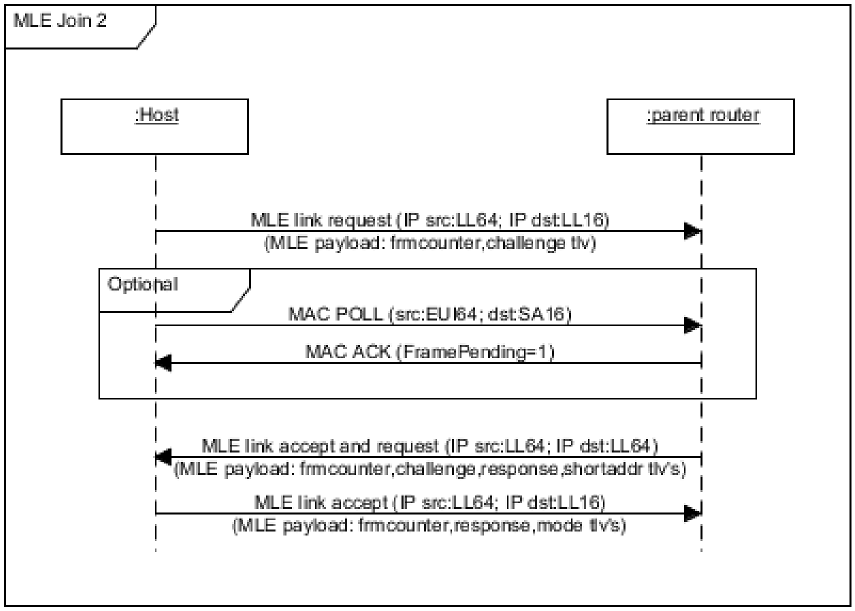

Figure 3: Join sequence – MLE 2

8. The node performs IPv6 router discovery described in [RFC 6775] by transmitting a Router Solicitation packet and waiting for Router Advertisement in response. The IPv6 prefix that is in use in the ZigBee IP network is extracted from the PIO option of the received Router Advertisement packets.

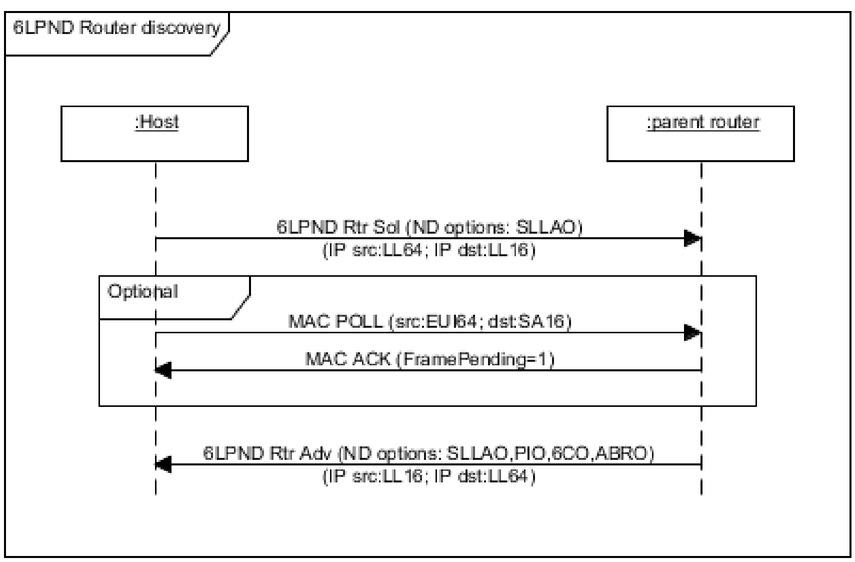

Figure 4: Join sequence - Router discovery

9. The node configures a randomly generated 16-bit address as its MAC short address. This address MUST NOT take the values 0xfffe or 0xffff, in accordance with the [802.15.4] specification. The node then configures an IPv6 global unicast address (GP16) and an IPv6 link local address (LL16) using the IID formed from this 16-bit MAC short address.

10. The node performs DAD (duplicate address detection) procedure for the IPv6 global unicast address as described in [RFC 6775]. The parent router uses the DAR (Duplicate address request) and DAC (Duplicate address confirmation) messages to register the GP16 address with the ZIP coordinator and check for uniqueness. Note that this also implies that the 16-bit MAC short address is unique within the ZigBee IP network. If the GP16 address is determined to be a duplicate, the node chooses a different GP16 address and repeats this process. Note that the node needs to use the GP16 address it is claiming as its IPv6 source address (as required by [RFC 6775]) during the 6LoWPAN neighbor discovery protocol exchange. However it MUST NOT use the corresponding 16-bit MAC short address until it has been confirmed as unique. Therefore, this message exchange contains use of mixed 64/16 addressing modes (i.e. the IPv6 address is formed using the 16-bit MAC address as the IID, however, the MAC address used is the 64-bit address).

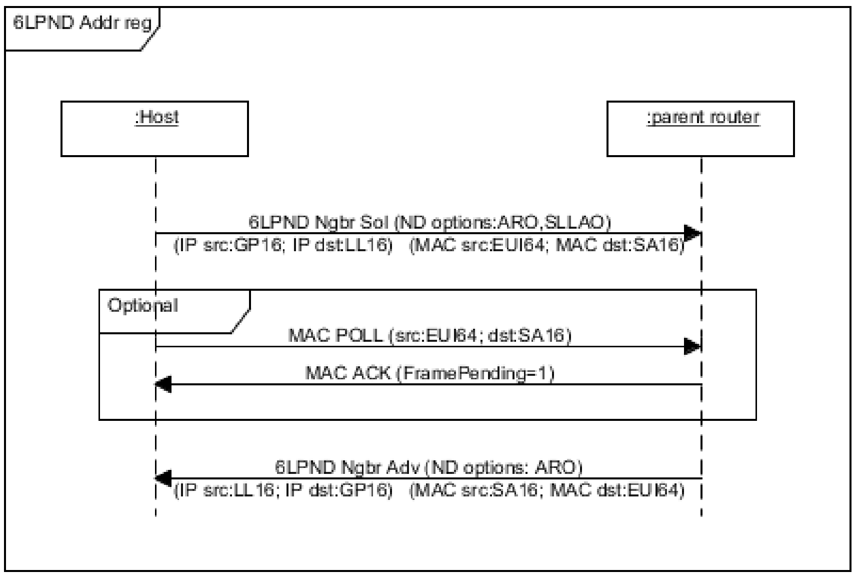

Figure 5: Join sequence - Address registration

11. The node performs a 3 way secured MLE handshake to securely exchange short addresses with the parent router. The node MUST include its 16-bit short address in the MLE payload in either the Link Request or Link Accept packets. At the end of this procedure, the node securely knows the parent router’s short address and the parent router securely knows the nodes short address. If the node is a sleepy Host, it MUST begin to use its short address to perform MAC poll as soon as it has updated the parent node with its short address.

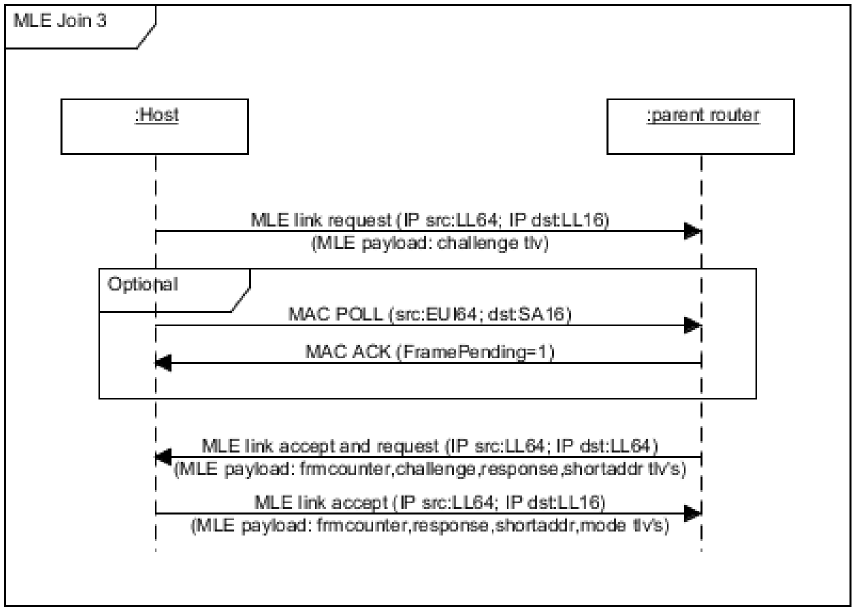

Figure 6: Join sequence - MLE 3

12. The parent router MUST check if the new node is a ZIP host. The Mode TLV in the MLE message SHOULD be used to make this determination (See Section 5.11.1). If the joining node is a host, the parent router MUST send RPL DAO messages to the DODAG Roots to create downward routes to the new node. The DAO message MUST contain the GP16 address of the joining node in the Target Prefix option and the GP16 address of the parent node in the Transit option. The External (E) flag MUST be set to one.

This concludes bootstrapping for Hosts. The Host node can now send and receive IP packets through its parent router.

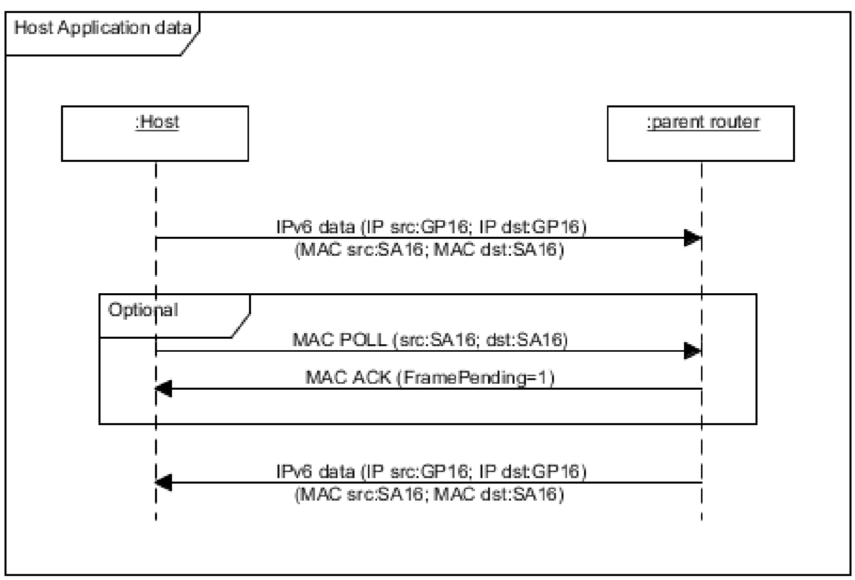

Figure 7: Join sequence - Application data

6.5.2 Router Bootstrapping

The bootstrapping sequence for a ZIP Router is described below.

1. The ZIP Router bootstrap sequence follows the sequence described in the previous section for the Host node with the following exceptions: A ZIP router MUST select its initial parent router from among those routers that have indicated available router capacity, which is indicated by setting the router capacity subfield in the beacon payload to 1. Since a ZIP router cannot be a sleepy node, the initial MLE exchange before PANA authentication (step 5 in the Host sequence) is OPTIONAL. It follows the Host sequence up until the final step (step 11 in the host sequence) and then continues as follows.

2. The ZIP router discovers its neighboring ZIP router nodes and configures secure Layer-2 links with each of them. This is accomplished using the MLE handshake exchange.

   The initial MLE link request packet is transmitted using the MAC broadcast address. All ZIP Routers that are in range will receive this packet and MAY respond with an MLE Link accept and request packet, depending on their available capacity to configure additional layer-2 links (note that the capacity to configure layer-2 links is limited by the size of the MAC device table).

   The joining router selects a subset from the responding ZIP routers and completes the MLE link establishment process with each of them. The selection of this subset is out of scope of this specification. This will cause the MAC device table in the joining router to be populated with entries for the selected neighboring routers. The joining router SHOULD ensure that it does not use up all of MAC device table capacity at this time. In order to allow other joining nodes to join the network later, it SHOULD ensure that it has some spare capacity in its MAC device table.

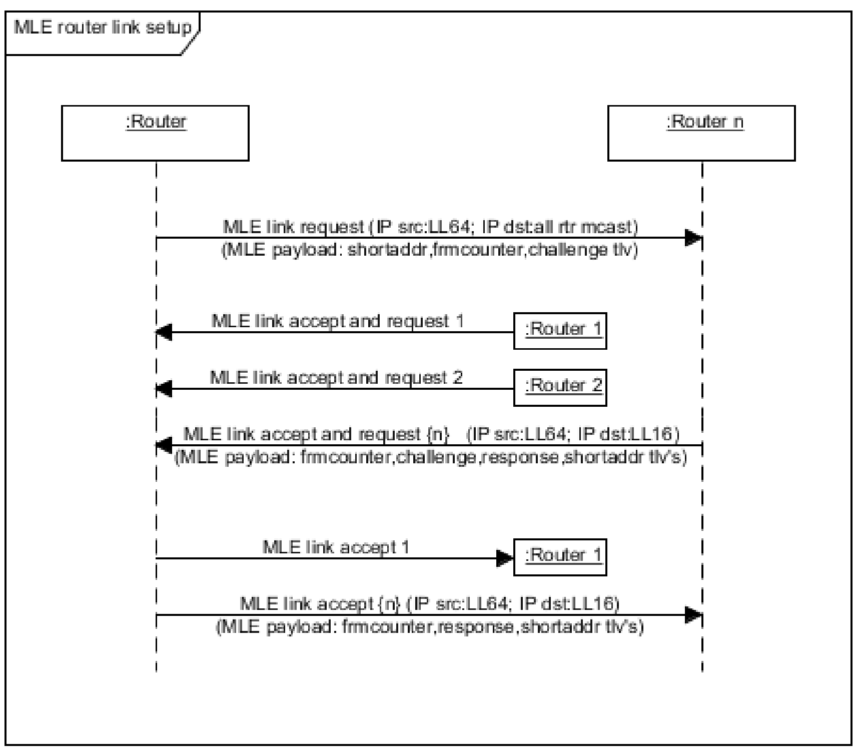

Figure 8: Join sequence - Router link setup

3. Next, the ZIP router begins configuration of the RPL routing protocol. The node transmits a multicast DIS packet to discover all available RPL instances. The node joins each RPL instance in turn using the sequence of messages below.

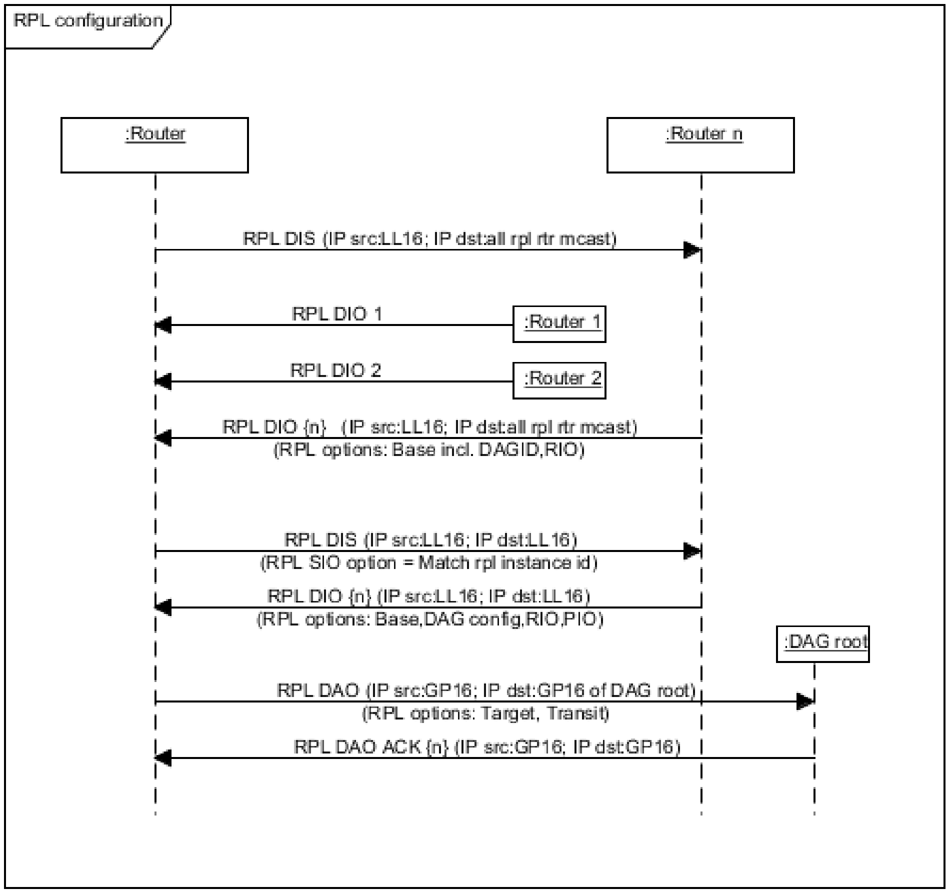

Figure 9: Join sequence - RPL configuration

4. The ZIP router is now part of the network and has full communication ability. The final step in the bootstrapping sequence is for the ZIP router to configure itself to function as an access router so that it can admit new nodes into the network. For this, it MUST configure the MAC beacon payload as described in section 6.3.1 and MUST start the MAC coordinator service so that it can transmit beacon packets in response to incoming beacon request packets. The association permit flag in the beacons MUST be set to false. It MUST enable the PANA Relay service. It MUST begin periodic transmission of MLE Link advertisement packets. It MUST update the Authentication server with its new GP16 address as described in section 6.9.3.6

6.6 Network Admission

When a new node joins the ZigBee IP network, it uses the PANA protocol to authenticate itself to the ZIP coordinator and gain access to the network security material. Once a node is admitted into the network it has full access to all communication capabilities on the network.

The authentication server can choose to eject an already admitted node from the network. It can do so by performing a selective update of the network key to all nodes except those that it has revoked access. It MUST perform the network key update twice in order to completely revoke network access for that node. See Section 6.10 for details on the updating network keys.

6.7 6LoWPAN Fragment Reassembly

ZIP nodes MUST transmit 6LoWPAN fragments in order and MUST complete transmission of one IP datagram before beginning transmission of another to the same next hop node. This allows a number of optimizations on the receiving node.

A ZIP node SHOULD buffer at most one incoming fragmented message from each neighbor node. When receiving a fragmented message from a neighbor, if a 6LoWPAN packet arrives from that neighbor that is not the expected next fragment, the partial message MAY be discarded. Also, if a non-initial fragment arrives that is not the expected next fragment, both that fragment and any partially received message from that neighbor MAY be discarded.

6.8 Sleepy Node Support

Hosts in a ZigBeeIP network MAY be battery-operated and can operate their radio for only a small fraction of time. Such Hosts are called sleepy hosts. A ZIP router is not allowed to be sleepy and MUST always have its radio enabled.

A sleepy host node receives data at the MAC layer using the indirect transmission scheme defined in [802.15.4]. In this scheme, the sending node buffers the outgoing MAC packet. When the sleepy Host activates its radio, it transmits a MAC POLL command packet to its parent router and then enables its radio receiver. The parent router transmits an acknowledgement packet in response and indicates within that (in the frame pending field of the MAC header) if it has any buffered packets that are pending transmission to the sleepy node. The sleepy host would continue to keep its receiver enabled for an additional period of time if it sees that the parent router has buffered packets for it. This allows the parent router to transmit the buffered packets to the sleepy host right after sending the acknowledgement packet.

ZIP routers MUST keep track of which of their neighbor nodes are sleepy host nodes. The ZIP router acquires this information through the Mode type option in the MLE message. The packet transmission to those nodes SHOULD use the MAC indirect scheme as defined in [802.15.4]. A ZIP router MUST have the ability to buffer at least MAC_MIN_INDIRECT_BUFFER number of full IPv6 packets. Each packet that is buffered for indirect transmission MUST be queued for a period of at least MAC_MIN_INDIRECT_TIMEOUT or until successfully transmitted to the intended destination. A ZIP router can prevent sleepy hosts from selecting them as the parent router by clearing the Host capacity bit in the MAC beacon payload. This SHOULD be done if the ZIP router has reached an internal limit on the number of sleepy host nodes it can service reliably.

Note that a sleepy host MAY change its sleepy nature dynamically. It MUST update its status with the parent router every time it changes its sleepy status. This is done using the Mode type option in the MLE message. As an example, if the application on the sleepy host is aware that there is a large amount of incoming data (as is the case if the node is receiving a new firmware update), it MAY change its state to a non-sleepy ZIP host and receive the packets using direct transmission. This will reduce the strain on the parent router buffers and also make the data transfer faster and more reliable.

It is expected that sleepy host nodes are usually the initiator of application-level transactions. They SHOULD typically not expect to receive packets unexpectedly. When a sleepy host node is expecting to receive packets, it SHOULD be able to poll its parent router at a faster rate than usual so that it can improve the probability that its parent router will be able to buffer the packet and deliver it successfully.

Special measures are necessary to accommodate sleepy hosts in a ZigBeeIP network, measures which are described below and which allow a host to communicate using indirect transmission even during the joining process.

6.8.1 Sleepy Host Joining

The initial node bootstrapping process is described in section 6.5.1 and the following text provides additional details.

A sleepy host starts the joining process without a configured MAC short address, so the source address of the MAC data request command packets is initially its extended address.

A sleepy host SHOULD indicate its sleepy nature to its parent router during the initial bootstrapping process. This is done through an MLE Link request message (see step 5 in Section 6.5.1). The Mode TLV is included in the link request message and the value contains the "Capability Information" field as defined in [802.15.4].

The parent router MUST respond with a MLE Link accept or reject message. It MUST transmit the response to the joining host using MAC indirect transmission, as this allows the host to poll for it. A ZIP router MUST NOT accept a sleepy host as a child, unless it has the capability to buffer at least one full IPv6 packet for a specified amount of time, in addition to the other requirements of establishing a new link (space in the mac device tables etc.). If a ZIP router does not have the necessary capacity to service a sleepy Host node, it MUST send a MLE link reject message in response to the link request.

Note that even though the sleepy node confirms a unique short address in step 10 (neighbor discovery) of the bootstrapping sequence described in 6.5.1, it MUST NOT configure the short address in its MAC layer until after it has updated its parent node with this information, which happens during step 11 of the bootstrapping sequence. The node MUST use its extended address for the MAC polling until then and it MUST use its short address afterwards.

6.8.2 Polling Rate

A sleepy host node can exist in one of two modes of sleeping, hereafter called *fast poll* and *slow poll*. The difference between the two modes is the MAC polling rate.

During fast poll, a sleepy node SHOULD be polling its parent router with sufficient frequency in order to receive its packets in a reasonable amount of time. What is reasonable depends largely on the retransmission timers in the various upper layers. In TCP, for example, the initial retransmission timeout is set at 3 seconds and increases with each successive retransmission. In order not to trigger unnecessary retransmissions, a sleepy host MUST poll its parent router at least once every MAC_MAX_FAST_POLL_TIME when it is in the fast poll state.

During slow poll state, a sleepy host can slow its polling rate significantly. A sleepy device MAY enter slow poll state at any time (or not at all). If a device wants to be able to enter slow poll state at all, it MUST communicate this to the parent during the link establishment process, by including a Timeout TLV in the MLE exchange. The timeout TLV indicates the maximum interval between successive polls (i.e. the polling period during the slow poll state). The value of the timeout field MUST be less than MAC_MAX_POLL_TIME. Note that the requirement on the parent router to buffer the IP packets for at least MAC_MIN_INDIRECT_TIMEOUT does not change when the sleepy host is in slow poll state. For this reason, there is very high chance that a sleepy host node will not be able to receive packets when it is in slow poll state.

A sleepy node SHOULD be in fast poll state if it expects to receive packets, and MAY enter slow poll otherwise. For example, it SHOULD be in fast poll state during the network joining process, after it has sent an MDNS or HTTP request and is waiting for the response.

The applications operating on ZIP nodes SHOULD be aware that sleepy host nodes are not always reachable reliably as they may be in slow poll state. It is typically safe to respond to queries (e.g., mDNS or HTTP) that are initiated by a sleepy host as the node would be expected to be in fast poll for a reasonable duration after sending the query.

6.8.3 MAC Data Request Command Frame Security

MAC data request command frames (i.e. polls) are always sent unencrypted at the MAC layer. More specifically, a parent MUST NOT discard unsecured polls from its children at the MAC layer, even if a fully established link exists with the originating child. The reason for this is that the child may be rejoining the network or performing key pull after a key switch, and may not have the current network key. Since parents always accept unsecured polls, there is no reason for sleepy children to secure them, even if they do have the network key.

6.8.4 Sleepy Host Link Maintenance

The network may undergo changes while a node is sleeping, especially if node is in deep sleep. For example, the network key may have changed, or the radio link with the parent router may have been lost. This section describes the symptoms and remedial actions that a sleepy host node SHOULD use to maintain its network status.

The typical behavior of a sleepy host node is to wakeup periodically and transmit a MAC Poll command packet to its parent router and receive the MAC acknowledgement packet in response. It may also transmit application packets at this time. If the application is expecting a response, the node SHOULD enter the fast poll state until the expected response is received or it has timed out.

If the sleepy host transmits application data packets and receives the expected response, that is sufficient confirmation that its network status has not changed and it can continue to operate normally.

If the sleepy node transmits application packets but cannot receive the response packets correctly due to update of the network security material, that can be detected by the management entity on the node through the internal MAC comm-status-indication with a status of UNAVAILABLE_KEY [802.15.4]. This SHOULD cause the sleepy host node to begin the PANA network key update process and retrieve the new security material from the authentication server. A sleepy node can also proactively check for new security material by doing a periodic key pull operation as described in section 6.10.2.

The management entity on the sleepy host can also detect loss of radio link with its parent router if it receives an internal MAC data-confirm with status of NO_ACK. [802.15.4]. This SHOULD cause the sleepy host node to attempt discovery and registration with a new parent router. The sleepy host can discover new parent routers through the MAC beacon mechanism as described in step 2 of section 6.5.1. After selecting the parent router, the sleepy host can proceed to register its address and perform a secured MLE exchange with the new parent router (Steps 10, 11 in section 6.5.1) as it already has access to the necessary security material and IPv6 address configuration information.

If the sleepy host did not transmit any application data packets for a long duration, it MAY proactively attempt to verify its network status. This can be done, for example, by transmitting an ICMPv6 echo request to its parent router. This SHOULD result in either the expected application response or one of the above error indications. The benefit of doing this is earlier detection of serious network changes, like a key update. The cost is an extra packet exchange. The cost-benefit depends on the actual deployment scenario and is therefore left up to the application.

If a sleepy host transmits application packets (including ICMPv6 Echo) to its parent node and does not get the expected response, and it also does not receive any MAC error indications, that is an indication that the network security material has been updated more than once. To recover from this state, the sleepy node cannot use the normal key update procedure. Instead it MUST rejoin the network which consists of searching for new parents by requesting beacons, performing the initial unsecured MLE exchange (Step 5 in section 6.5.1) with the new parent, performing a key pull instead of a PANA authentication to get the network key, and then performing a full secured MLE exchange with the new parent (Step 11 in section 6.5.1). The network rejoin procedure involves a number of packet exchanges, so a sleepy node SHOULD not perform this until after it has tried unsuccessfully to communicate with its parent node a few times.

6.9 Network Authentication

During the network join process, the node performs network authentication to ensure it is on the right network and acquire the necessary security credentials. Similarly, the network authenticates the node to ensure that the node is trusted and has the necessary security credentials to join the network.

The purpose of the authentication procedure is to provide mutual authentication resulting in:

- Preventing untrusted nodes without appropriate credentials from joining a trusted ZigBee IP network
- Preventing trusted nodes with appropriate credentials from joining an untrusted ZigBee IP network.

The Authentication Server resides on the ZIP Coordinator and is responsible for authenticating the nodes on the network. If the authentication is successful, the Authentication server sends the network security material to the joining node through the PANA protocol. The joining node becomes a full participating node in the ZigBee IP network and is able to exchange IP packets with all other nodes in the network.

The authentication attempt MUST fail on the Authentication server if the EAP-TLS server cannot successfully authenticate the new node. This depends on the security credentials that are presented during the EAP-TLS handshake.

Additionally, the authentication attempt can fail based on application logic that is out of scope of this specification. An example of such application logic is a user button on the ZIP Coordinator, where all join attempts are rejected unless they happen within a brief period of time after the button is pressed. Note that in such a scenario, a ZIP Coordinator SHOULD still accept join attempts from nodes that have dropped off the network and are performing a rejoin. Another example of application logic is an explicit whitelist or blacklist of node identities.

The joining node does not initially have access to the network security material. Therefore, it is not able to apply MAC layer security for the packets exchanged during the authentication process. The enforcement point rules in the ZIP routers are described in section 6.9.4 and they ensure that the packets involved in the PANA authentication are processed even though they are unsecured at MAC layer. The rules also ensure that any other incoming traffic that is not secured at the MAC layer is discarded by a ZIP node and is not forwarded.

6.9.1 Authentication Stack

Authentication can be viewed as a protocol stack as a layer encapsulates the layers above it. The ZIP authentication protocols are shown in relation to each other in the figure below.

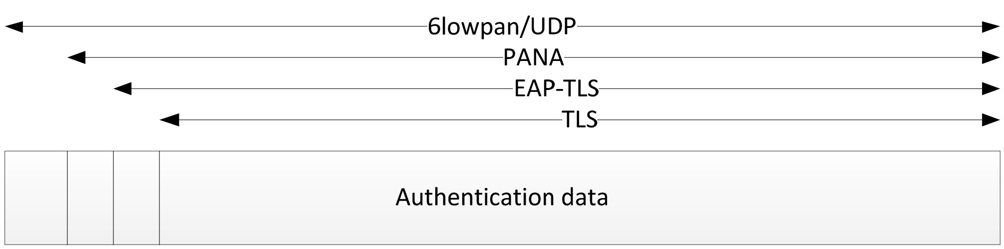

Figure 10: Authentication protocol stack within the ZigBee IP network

TLS [RFC 5246] MUST be used at the highest layer of the authentication stack and carries the authentication exchange. There is one cipher suite based on pre-shared key [RFC 6655] and one cipher suite based on ECC [TLS-CCM-ECC].

EAP-TLS [RFC 5216] MUST be used at the next layer to carry the TLS records for the authentication protocol.

The Extensible Authentication Protocol [RFC 3748] MUST be used to provide the mechanisms for mutual authentication. EAP requires a way to transport EAP packets between the Joining Node and the Node on which the Authentication Server resides. These nodes are not necessarily in radio range of each other, so it is necessary to have multi-hop support in the EAP transport method. The PANA protocol [RFC 5191], [RFC 6345], which operates over UDP, MUST be used for this purpose. [RFC 3748] specifies the derivation of a session key using the EAP key hierarchy; only the EAP Master Session Key shall be derived, as [RFC 5191] specifies that it is used to set up keys for PANA authentication and encryption.

PANA (RFC5191) [RFC 5191] and PANA relay (RFC6345) [RFC 6345] MUST be used at the next layer:

- The Joining Node MUST act as the PANA Client (PaC)
- The Parent Node MUST act as a PANA relay (PRE) according to [RFC 6345], unless it is also the Authentication Server. All ZIP routers MUST be capable of functioning in the PRE role.
- The Authentication Server node MUST act as the PANA Authentication Agent (PAA). The Authentication Server MUST be able to handle packets relayed according to [RFC 6345]

This network authentication process uses link-local IPv6 addresses for transport between the new node and its parent. If the parent is not the Authentication Server, it MUST then relay packets from the Joining Node to the Authentication Server and vice-versa using PANA relay mechanism [RFC 6345]. The joining node MUST use its LL64 address as the source address for initial PANA authentication message exchanges.

6.9.2 Applicability Statements

The applicability statements describe the relationship between the various specifications.

6.9.2.1 Applicability Statement for PSK TLS

[RFC 6655] contains AEAD TLS cipher suites that are very similar to [RFC 5487] whose AEAD part is detailed in [RFC 5116]. [RFC 5487] references both [RFC 5288] and the original PSK cipher suite document [RFC 4279], which references [RFC 5246], which defines the TLS 1.2 messages.

6.9.2.2 Applicability Statement for ECC TLS

[TLS-ECC-CCM] contains AEAD TLS cipher suites that are very similar to [RFC 5289] whose AEAD part is detailed in [RFC 5116]. [RFC 5289] references the original ECC cipher suite document [TLS-ECC] (RFC4492), which references [RFC 5246] document, which defines the TLS 1.2 messages.

6.9.2.3 Applicability Statement for EAP-TLS and PANA

[RFC 5216] specifies how [RFC 3748] is used to package [RFC 5246] messages into EAP packets. [RFC 5191] provides transportation for the EAP packets and additional configuration information carried in vendor specific attribute-value pairs (AVPs) and encrypted AVPs specified in [RFC 6786] and this document. The proposed PRF and AUTH hashes based on SHA-256 are represented as in [IKEv2] (RFC5996) and detailed in [IPSEC-HMAC] (RFC4868).

6.9.3 PANA

6.9.3.1 PANA Session

[RFC 5191] specifies several phases for a PANA session; a Zigbee IP PANA session MUST always be in either the authentication or authorization phase. A ZigBee IP PANA session MUST be initiated by the PaC. A ZigBee IP PANA session between the PaC and the PAA MUST remain open for the purposes of network key update and maintenance.

6.9.3.2 PANA Security Association

[RFC 5191] specifies that the PANA security association is set up based on the authentication key derived from the EAP Master Session Key and that the authentication key is used to authenticate the final PANA messages. [RFC 6786] specifies the derivation of an encryption key, which MUST be used for encrypting transport of the Network Key, Network Key Index and ancillary data to nodes.

The PAA MUST maintain the following attributes as part of the secure association, in addition to those specified by [RFC 5191].

- The EUI-64 of the PaC. This SHOULD be derived from the LL64 address of the PaC that is associated with this secure association. This information is used to uniquely identify the PaC and prevent duplicate sessions.
- The Node Auth Counter. This is a 1 octet value that is stored on the PAA and transported to the PaC as part of the network security material.

6.9.3.3 PANA between Joining Node (PaC) and Parent Node (PRE or PAA)

PANA messages between the Joining Node and the Parent Node MUST use single-hop unicast transmission in both directions with the following header addresses:

Address	Value	Comment
MAC address	64-bit	IEEE address of the Joining Node
IP address	LL64	Stateless autoconfigured link-local address of joining Node

Table 19: PANA joining node header addresses

Address	Value	Comment
MAC address	16-bit	Short address of the Parent Node
IP address	LL16	Stateless autoconfigured link-local address of parent node

Table 20: PANA parent node header addresses

6.9.3.4 PANA between Parent Node (PRE) and Authentication Server (PAA)

If the Parent Node and the Authentication Server are not the same node, then the Parent Node MUST relay PANA messages exchanged between the Joining Node and the Authentication Server according to [RFC 6345]. The relaying is transparent to the Joining Node; as far as it is concerned it is talking directly to the Authentication Server.

Relayed PANA messages between the Parent Node and the Authentication Server MUST use standard unicast transmission in both directions. Relayed PANA messages are secured at the link layer, thus satisfying the requirements of Section 3 of [RFC 6345] and avoiding the need for alternative packet protection.

6.9.3.5 Network Security Material Transport

If the PANA authentication attempt is successful, the PAA MUST transmit the network security material to the joining node in the final PANA Authentication Request message from PAA to PaC. The network security material MUST be transported in the network key AVP (see Section 5.6.3) that is encrypted using the Encr-Encap AVP [RFC 6786]. The values of the Network Key and Index MUST contain the current active network security material. The value of the Node Auth Counter MUST be taken from the PANA secure association state for that node.

At the point of completing the PANA authentication, the PAA MUST check if it has a duplicate secure association with this node. For purpose of checking the duplicate session information, the PAA SHOULD use the EUI-64 MAC address of the node. This attribute is derived from the LL64 address that is used by the PaC during the PANA authentication and is stored as part of the session information.

If a duplicate secure association is found, the PAA MUST take the Node Auth Counter value from the duplicate secure association, increment it by one (rollover to zero if necessary) and copy it into the new secure association. Furthermore, it MUST delete the old session information. Otherwise, the PAA SHOULD use a value of zero for the Node Auth Counter attribute in the secure association.

6.9.3.6 PaC Address Update

A ZIP node uses its link local IP address during the PANA authentication process. As a result, the PAA secure association for each node contains the link local address. After authentication is completed, the node bootstrap process results in the configuration of a global unicast (GP16) IP address. [RFC 5191] requires that if a node changes the IP address it uses for PANA communications, it must update that address at the PAA.

A ZIP router MUST update its IP address at the PAA server to its GP16 address after completing its bootstrap process. This is achieved by sending any valid PANA packet to the PAA with the GP16 as the source IP address. Typically, a PANA Notification Request message is used for this purpose. After updating its IP address at the PAA, the node and PAA can communicate directly using the global unicast IP addresses.

A ZIP host SHOULD not update its IP address at the PAA server to its GP16 address. Since a ZIP host is typically a sleepy device, it is not always reachable from other nodes. Therefore, a ZIP host SHOULD continue to use is link local IP address for communications with the PAA. These communications MUST be addressed to the PANA Relay entity at its parent router which relays them to the PAA.

6.9.4 Enforcement Point Processing

Every ZIP Node MUST implement an Enforcement Point (EP) function. The EP acts by policing all traffic entering a node at all layers up to layer 4, thus effectively firewalling communication from all outside nodes. The EP has filtering rules which are dependent on configuration and packet properties. The filtering rules are described below. The net effect of these rules is that all incoming MAC data packets that are not secured at the MAC layer are discarded unless it contains an IPv6 packet with a destination address that belongs to the node and sent using UDP protocol to the assigned PANA port number (716) or to the assigned MLE port number.

6.9.4.1 Layer 2 (MAC) Filtering

- If the packet is protected by L2 security (network key), the EP MUST tag the packet as ‘L2 secure’ and bypass any further layer filtering, allowing the packet through for further processing.
- If the packet is unprotected by L2 security (network key), the EP MUST tag the packet as ‘L2 unsecure’ and pass the packet for Layer 3 filtering.

6.9.4.2 Layer 3 (IP) Filtering

- If the packet is tagged as ‘L2 unsecure’ and the packet is a UDP message destined to this node (the destination IP address is a link-local address assigned to this node, including multicast addresses with link-local scope), the EP MUST pass the packet for Layer 4 filtering.
- Otherwise the EP MUST silently discard the packet.

6.9.4.3 Layer 4 (Transport) Filtering

- If the packet is tagged as ‘L2 unsecure’, and the packet is either a PANA message from a Joining Node (characterized as a UDP datagram with the destination port set to the assigned PANA port number and using link-local source and destination addresses) or an MLE packet (characterized as a UDP datagram with the destination port set to the assigned MLE port number), the EP MUST pass the packet to the respective application layer.

  In the case of MLE messages, the rules for handling of “L2 unsecured” messages are further described in 5.11.4. In case of PANA messages, no additional rules are necessary as the protocol does not rely on lower layer security.

- Otherwise the EP MUST silently discard the packet.

6.10 Network Key Update

The network key can be updated by the Authentication server at any time. The frequency and timing of such updates is implementation-specific. However, it MUST NOT initiate a network key update until the previous key update and activation is complete.

Typically, the Authentication server would update the network security material for one of the following reasons

- Periodically update security material used for the MAC frame security as part of a standard operating procedures
- Revoke network access to a node that possesses the current network security material.
- Update security material in anticipation of the Node Auth Counter reaching its maximum value for any ZIP node.

The updated network security material is delivered to the authorized nodes via the PANA protocol. It can be delivered via either a “push” or “pull” mechanism. The PAA “pushes” the updated network security material to all ZIP routers. The ZIP hosts are expected to “pull” the updated network security material from the PAA.

It is RECOMMENDED that the Authentication server update the network security material periodically with duration between 1 day and 1 month. The reason to update network security material at least once a month is to ensure that the node frame counter does not reach the maximum value. However, if security material is updated too frequently, that will add control overhead on the network. Also, sleepy Host nodes can potentially miss the key updates and lose network connectivity. Therefore, it is RECOMMENDED that key update is not performed more often than once a day.

An example network key update process is illustrated in Figure 11

6.10.1 PAA Network Security Update Procedure

The network security update is triggered by the management entity on the Authentication server.

A new network security material (see Section 5.6.2) is created by generating a new 128-bit Network Key. The sequence number for this key SHOULD be set to the sequence number of the current active security material, incremented by one. If the current sequence number value has a value of 255, the new sequence number SHOULD roll over to a value of 1. The Node Auth counter MUST be reset to a value of 0 for all nodes.

In addition to the new security material, the management entity MAY also provide a list of nodes, identified by their EUI-64 MAC addresses, which are currently on the network but SHOULD not receive any further network security material.

Upon obtaining the new network security material, the PAA server performs the following actions:

1. The PAA deletes the PANA sessions corresponding to the nodes that are not eligible to receive further network security material.
2. The PAA "pushes" new network security material to each node for which it has a secure association and also possesses the global unicast IP address.
3. The "push" involves sending a PANA Notification Request message. The PAA MUST include the updated network security material in a network key AVP (see Section 5.6.3) that is encrypted using the Encr-Encap AVP [RFC 6786].

After the PAA has completed the above, the management entity MAY activate the new security material.

During the time between the start of the key update process and completion of the activation, the PAA is in possession of two network security materials. Note that this includes two copies of the Node Auth counter for each node.

6.10.2 Network Key Pull

A ZIP node MUST initiate a network key pull when it detects usage of new security material by another node. This happens when the node receives a packet that is secured at the MAC or MLE layers using a key index that is greater (taking rollover into account) than what it currently possesses.

6.10.2.1 Request

The network key pull is initiated by sending a PANA Notification Request message to the PAA. The node SHOULD use the IP address that is has previously registered with the PAA as the source address when sending this message (see Section 6.9.3.6). This is the link local address in the case of a ZIP Host and the GP16 address for a ZIP Router.

A ZIP host MUST use its link local IP address as the source address for this packet. It MUST send the packet to its parent router. The PANA Relay entity on the parent router will transparently relay this request and the response between the Host and the PAA.

A ZIP router MUST use the global unicast IP address that it has previously registered with the PAA as the source IP address and send the packet directly to the PAA.

If the ZIP node supports the Key Request AVP, it MUST include it in the PANA Notification Request packet. The `nwk_key_req_flags` SHOULD be set of value of 1. The `nwk_key_idx` field SHOULD be populated with value of the current active key index.

6.10.2.2 Response

The PANA Notification Answer message is sent from the PAA to the ZIP node in response to the above request.

If the incoming PANA Notification Request message does not include the Key request AVP or if the PAA does not support the Key request AVP, then the PAA MUST transport the new network security material if a key update is currently in progress or transport the current network security material otherwise.

If the incoming PANA Notification Request message includes the Key request AVP and the PAA supports this AVP, the PAA responds as follows:

- If the least significant bit of the `nwk_key_req_flags` field has a value of 1:

   - If the nwk_key_idx field is equal to the active key index, then the PAA MUST transport the new network security material if a key update is in progress and MUST send an empty response otherwise.
   - If the nwk_key_idx field is not equal to the active key index, the PAA MUST transport the active network security material.
- If the least significant bit of the nwk_key_req_flags field has a value of 0:
   - If the nwk_key_idx field is equal to the active key index, then the PAA MUST transport the active network security material.
   - Otherwise, the PAA MUST send an empty response

The PAA MUST transport the current or new network security material in a network key AVP (see Section 5.6.3) that is encrypted using the Encr-Encap AVP [RFC 6786]. The Node Auth counter MUST be set to value of zero if the new security material is being transported. Otherwise, the auth counter attribute from the PANA secure association corresponding to the ZIP node MUST be incremented by one and that value MUST be used in the network key AVP.

Note that if the PAA is transporting the network security material to a new node that is joining the network (i.e., in the final PANA Authentication Request message from PAA to PaC), it MUST always transport the current active network security material to the node.

A ZIP host MAY also periodically perform the network key pull procedure to check if there is updated security material at the PAA before that material is activated. However, this SHOULD be done judiciously if either the PaC or the PAA does not support the key request AVP as each network key pull results in an increment of the node auth counter value until the next network key update resets it to zero. If the auth counter reaches the maximum value for a node, then the node frame counters could reach their maximum limit and the node would be unable to communicate securely in the network.

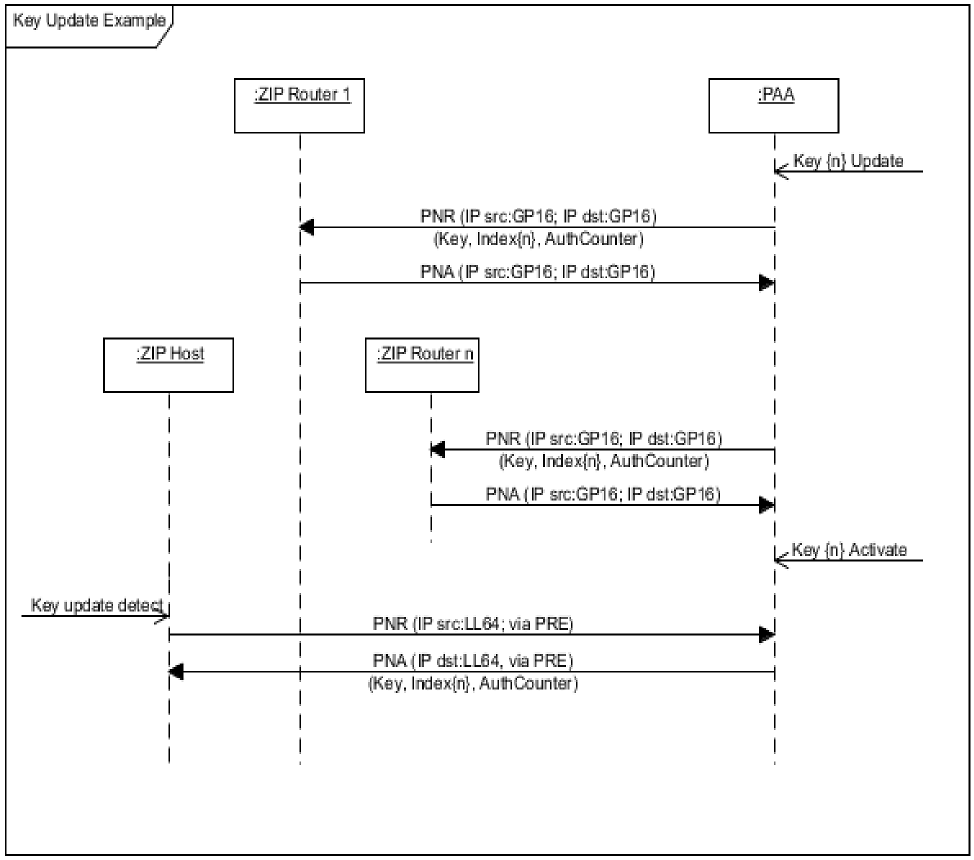

Figure 11: Network key update

6.10.3 Network Key Activation

The management entity on the Authentication server is responsible for activating the new network security material.

It is RECOMMENDED that this action is taken a short time after the new security material has been propagated to all the non-sleepy nodes in the network. The additional delay allows sleepy nodes to pull the new security material from the PAA before it is activated.

The activation of the new network security material results in an update to the active MAC key and active MLE key as they are derived from the network security material.

On the PAA, the node simply activates the MAC and MLE security material whose key index matches the new network key sequence number. This will cause outgoing MAC frames and MLE messages from the PAA to be secured with the new key material.

When a ZIP node receives an incoming MLE message that is secured with a higher key index (adjusting for index rollover) than its current active MLE key index, and that higher key index is equal to the alternate MLE key index, the node MUST swap the active and alternate MLE security materials.

When a ZIP node receives an incoming MAC message that is secured with a higher key index (adjusting for index rollover) than the nodes current active MAC key index, and the node possesses a MAC KeyDescriptor with that higher key index, the node updates the value of its active MAC key index to the higher key index.

When a ZIP node updates the active security material for either the MAC or MLE layer, the node management entity SHOULD also update the active security material for the other layer at the same time.

6.11 Node Diagnostics

The ZIP stack makes available node management and diagnostic functionality for the 802.15.4 layer, 6LoWPAN layer and the network layer. For each of these layers the following information SHOULD be available. The node management functions shall always be available however the collection of diagnostics and statistics MAY be turned on and off.

The IEEE 802.15.4 layer MUST make the following attributes available to the node management application:

- IEEE EUI 64 address
- IEEE short address
- CapabilityInfo
- Device PANID

The IEEE 802.15.4 layer SHOULD make the following information available:

- Packets sent and received
- Octets sent and received
- Packets dropped on transmit and receive
- Security errors on receive
- Packet transmit failures due to no acknowledgement
- Packet transmit failure due to CSMA (channel access) failure
- Number of MAC retries

The 6LoWPAN layer SHOULD make the following information available:

- Packets sent and received
- Octets sent and received
- Fragmentation errors on receive

The network layer SHOULD make the following parameters available:

- IPv6 address list: The list of IPv6 addresses that are assigned to the ZigBee IP interface on the node
- RPL instance list: The list of RPL instances to which the node belongs
- RPL source routes list: The list of RPL source routes, for each RPL Instance, that are available on the node.
- RPL parent list: The set of RPL parents, for each RPL Instance, on this node.

The management layer SHOULD make the following parameters available:

- NetworkID: The identifier of the ZigBee IP network to which this node belongs.
- MLE neighbor table: The list of neighbor node addresses and the associated link quality information.

6.12 Persistent Data

Devices operating in the field may be reset either 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. Devices which are reset need to have the ability to restart network operation without user intervention.

ZIP Routers and Hosts SHOULD store the NetworkID value in non-volatile storage. This is so that the node can recover from an unscheduled reset without user intervention. Additionally, ZIP Routers and Hosts SHOULD store the PANA security session information in non-volatile storage to make the rejoin process more efficient. A node that is restoring previous configuration after a reset SHOULD not reuse its previous GP16 IPv6 address (or the MAC short address) without checking for uniqueness again.

ZIP Coordinator MUST store in persistent storage all the information that is necessary to restore the ZIP network configuration after a reset. This includes

- The value of ZIP NetworkID field
- The PANA security session information for each of the authenticated nodes.
- The network security key material
- The information necessary to recreate information in the Router advertisement packet. This includes the ABRO version, prefix and context information
- The information necessary to recreate the DIO packets. This includes the RPL Instance id and DODAG version.

The method by which this data is made to persist is outside the scope of this specification.

7 Constants and Attributes

This section specifies the constants and attributes required by the ZigBee IP protocol suite.

7.1 Attributes

A ZIP node MUST configure the following attribute values.

Attribute	**Description**	**Value**
MIN_6LP_CID_COUNT	The minimum number of 6LoWPAN header compression context identifiers that are supported by a node	4
MIN_6LP_PREFIX	The minimum number of 6LoWPAN prefixes that are supported by a node.	2
MIN_RPL_INSTANCE_COUNT	The minimum number of RPL Instances that a ZIP Router is capable of participating in.	2
MLE_ADV_INTERVAL	The time interval between transmission of successive MLE advertisement packets by a ZIP Router.	16 seconds
MLE_ADV_TIMEOUT	The time interval after which a ZIP router SHOULD remove a node from it MAC device table if it has not received MLE advertisements from that neighbor node containing this node as a neighbor.	54 seconds
MLE_MAX_ALLOW_JOIN_TIME	The maximum amount of time a ZIP router SHOULD keep the Allow Join flag enabled without additional commands.	30 minutes
RPL_INSTANCE_LOST_TIMEOUT	The amount of time a ZIP Router can lose connectivity to a RPL Instance before removing itself from that Instance.	1200 seconds
RPL_MIN_DAO_PARENT	The number of DAO parents that a RPL router SHOULD be able to support.	2
RPL_MAX_RIO	The maximum number of route information options that SHOULD be included in a DIO packet.	3
RPL_MTU_EXTENSION	The additional number of bytes added to the link layer MTU for IP packets sent over the RPL tunnel interface.	100 bytes
RPL_MAX_PIO	The maximum number of prefix information options that can be included in a DIO packet.	1
EAP_TLS_MTU	The maximum size of TLS data in the EAP payload when using EAP-TLS	512 octets

	fragmentation.	
MAC_MIN_INDIRECT_TIMEOUT	The minimum amount of time a ZIP router buffers an IPv6 packet for indirect transmission at the MAC layer.	1 second
MAC_MIN_INDIRECT_BUFFER	The minimum number of IPv6 packets that a ZIP router can buffer for indirect transmission at the MAC layer.	1
MAC_MAX_FAST_POLL_TIME	The maximum duration between consecutive MAC polls when a sleepy host node is in fast poll state.	500 ms
MAC_ MAX_POLL_TIME	The value for maximum duration of inactivity from a sleepy host after which a ZIP router can remove the entry from its MAC device table.	1 day
MAC_MAX_NWK_KEYS	The number of MAC keys that are stored by a node.	2
MAC_MIN_DEV_TBL	The minimum number of entries a ZIP router SHOULD support in the MAC device table.	6
MCAST_MIN_TBL_SIZE	The minimum number of trickle multicast sequence values that can be stored in a ZIP router.	8

Table 21: Node attributes

8 Informative Appendix

This section contains informative clarifications which are used to aid implementation of the specification. The clarifications are there to clarify explicit or implicit normative requirements.

All normative requirements are contained in the normative sections of this document and the specifications referenced in this document.

8.1 PANA

8.1.1 Packets

PANA packets SHOULD be a multiple of 4 octets in size.

8.1.2 AVPs

PANA AVPs can appear in any order, except for the AUTH AVP, which must be the final AVP. Octet string AVPs (Auth, EAP-Payload, Nonce) must be aligned to 4 octets, without the padding being included in the length field; other AVPs are automatically aligned.

8.1.3 Transactions

PANA packet transactions form the basis of transportation of EAP packets. PANA transactions occur between a PANA client (PaC) and a PANA Authentication Agent (PAA) and can be relayed via a PANA Relay Entity (PRE). A relayed session essentially carries the same EAP and TLS information but the PANA session is carried between three entities.

An EAP Response SHOULD be piggy-backed on the PANA answer. However an implementation SHOULD assume that an EAP Response may alternatively be carried in a separate PAR initiated by the PaC followed by a PAN from the PAA.

8.1.4 PANA Key Generation

[RFC 5191] and [RFC 6786] specify how the PANA_AUTH_KEY and PANA_ENCR_KEY are generated. This section provides additional guidance.

```
PANA_AUTH_KEY = prf+(MSK, "IETF PANA",
|I_PAR|I_PAN|PaC_nonce|PAA_nonce|Key_ID);
PANA_ENCR_KEY = prf+(MSK, "IETF PANA Encryption Key",
|I_PAR|I_PAN|PaC_nonce|PAA_nonce|Key_ID);
```

The PRF function only needs to be iterated once as the PANA_AUTH_KEY and PANA_ENCR_KEY lengths are the same as the underlying hash, i.e. 32 bytes. Therefore, the TLS PRF function can be used simply by concatenating 0x01 to the string:

```
prf+(K, S) = P_hash(K, S | 0x01)
```

The string “IETF PANA” is not null-terminated, i.e. has a length of 9 octets and the string “IETF PANA Encryption Key” is not null-terminated, i.e. has a length of 24 octets.

8.1.5 IKEv2 prf+ Function used in PANA

All PANA transactions use the prf+ function specified in [IKEv2] (RFC5996). In the following, | indicates concatenation.

prf+ is defined as:

```
prf+ (K,S) = T1 | T2 | T3 | T4 | ...
```

where:

```
 T1 = prf (K, S | 0x01)
 T2 = prf (K, T1 | S | 0x02)
```

```
T3 = prf (K, T2 | S | 0x03)
T4 = prf (K, T3 | S | 0x04)
...
```

This continues until all the material needed to compute all required keys has been output from prf+.

The PRF used is the IPsec PRF function PRF-HMAC-SHA-256 specified in [IPSEC-HMAC].

Note that the HMAC key size (section 2.1.1) specifies that the HMAC key size must be the size of the underlying hash. So in this case, the PANA_AUTH_KEY size is 32 bytes (the output from SHA-256).

Note also that if the output is always the size of the underlying hash or less, the prf+ function only has to be iterated once. In this case, the TLS PRF function can be used simply by concatenating 0x01 to the string:

```
prf+(K, S) ≡ P_hash(K, S | 0x01)
```

8.2 TLS

8.2.1 TLS PSK

8.2.1.1 Premaster Secret

[RFC 4279] states: "if the PSK is N octets long, concatenate a uint16 with the value N, N zero octets (plain PSK case), a second uint16 with the value N, and the PSK itself"

Premaster Secret = 00 10 || 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 || 00 10 || CF CE CD CC CB CA C9 C8 C7 C6 C5 C4 C3 C2 C1 C0

where || is the concatenation operator.

Note that the concatenation of the length with the data represents a TLS variable length vector <0..2^16-1>

8.2.1.2 PSK Key Exchange

The TLS PSK key exchange is shown below. The optional elements are not shown.

```
Client                                               Server
------                                               ------

ClientHello                 -------->
                                                ServerHello
                            <--------       ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished                    -------->
                                           ChangeCipherSpec
                            <--------              Finished
Application Data            <------->      Application Data
```

8.2.1.3 PSK Verify Data

In the following diagram:

- ‘+’ indicates concatenation
- ‘[]’ indicates recipient of data as opposed to originator of data or in the case of `verify_data`, reconstructed data
- ‘=>’ indicates calculation
- The final `Finished` message included in the concatenation of messages is used as cleartext

- Validation can be performed on the server at `SVAL` and at the client at `CVAL`
- `verify_data = PRF(master_secret, finished_label, Hash(handshake_messages))`
- `verify_data_length` is 12 (bytes)
- For `Finished` messages sent by the client, the finished_label is the string "client finished"
- For `Finished` messages sent by the server, the finished_label is the string "server finished"

Verify data is calculated over the accumulated handshake messages as follows:

```
      Client                                                  Server
      ------                                                  ------

      C:ClientHello              -------->             [C:ClientHello]
      +                                                              +
      [S:ServerHello]                                    S:ServerHello
      +                                                              +
      [S:ServerHelloDone]        <--------           S:ServerHelloDone
      +                                                              +
      C:ClientKeyExchange                          [C:ClientKeyExchange]
      => C:verify_data                                => [C:verify_data]
      +                                                              +
      C:Finished(C:verify_data)  -------->  [C:Finished(C:verify_data)]
SVAL
      => [S:verify_data]                               =>  S:verify_data
CVAL  [S:Finished(S:verify_data)]  <--------    S:Finished(S:verify_data)
```

8.2.2 TLS ECC

8.2.2.1 ECC Key Exchange

The TLS ECC key exchange is shown below. The optional elements are not shown. Since authentication is mutual, if this cipher suite is used, the TLS server must require client authentication, i.e. it must request the client's certificate

```
      Client                                               Server
      ------                                               ------

      ClientHello                 -------->
                                                      ServerHello
                                                      Certificate
                                                ServerKeyExchange
                                               CertificateRequest
                                  <--------       ServerHelloDone
      Certificate
      ClientKeyExchange
      CertificateVerify
      ChangeCipherSpec
      Finished                    -------->
                                                 ChangeCipherSpec
                                  <--------              Finished
      Application Data            <------->      Application Data
```

8.2.2.2 ECC Verify Data

In the following diagram:

- '+' indicates concatenation
- '[]' indicates recipient of data as opposed to originator of data or in the case of `verify_data`, reconstructed data

- '=>' indicates calculation
- The final `Finished` message included in the concatenation of messages is used as cleartext
- Validation can be performed on the server at `SVAL` and at the client at `CVAL`
- `verify_data = PRF(master_secret, finished_label, Hash(handshake_messages))`
- `verify_data_length` is 12 (bytes)
- For `Finished` messages sent by the client, the finished_label is the string "client finished"
- For `Finished` messages sent by the server, the finished_label is the string "server finished"

Verify data is calculated over the accumulated handshake messages as follows:

```
      Client                                                   Server
      ------                                                   ------

      C:ClientHello             -------->            [C:ClientHello]
      +                                                             +
      [S:ServerHello]                                   S:ServerHello
      +                                                             +
      [S:Certificate]                                   S:Certificate
      +                                                             +
      [S:ServerKeyExchange]                       S:ServerKeyExchange
      +                                                             +
      [S:CertificateRequest]                     S:CertificateRequest
      +                                                             +
      [S:ServerHelloDone]       <--------             S:ServerHelloDone
      +                                                             +
      C:Certificate                                   [C:Certificate]
      +                                                             +
      C:ClientKeyExchange                       [C:ClientKeyExchange]
      +                                                             +
      C:CertificateVerify                      [C: CertificateVerify]
      => C:verify_data                             => [C:verify_data]
      +                                                             +
      C:Finished(C:verify_data)   -------->  [C:Finished(C:verify_data)]
SVAL
      => [S:verify_data]                            =>  S:verify_data
CVAL  [S:Finished(S:verify_data)]  <--------    S:Finished(S:verify_data)
```

8.2.3 TLS ECC Additional Information

8.2.3.1 ClientHello Extension

`ClientHello` has extensions, which can be identified as additional data being present after the `compression_methods` field.

The extensions from section 5.1 of [TLS-ECC] are as follows:

- `elliptic_curves` (10), size 4:
  - `EllipticCurveList` length: 2
  - One `NamedCurve`: `secp256r1` (0x0017)
- `ec_point_formats` (11), size 2
  - `ECPointFormatList` length: 1
  - One `ECPointFormat`: `uncompressed` (0x00)

The extensions from [RFC 5246] are as follows:

- `signature_algorithms` (13), size 4:
  - `SignatureAndHashAlgorithm` length: 2
  - hash `sha256` (0x04)
  - signature `ecdsa` (0x03)

8.2.3.2 ServerHello Extension

`ServerHello` has extensions, which can be identified as additional data being present after the `compression_method` field.

The extensions from section 5.2 of [TLS-ECC] are as follows:

- `ec_point_formats` (11), size 2:
  - `ECPointFormatList` length: 1
  - One `ECPointFormat`: `uncompressed` (0x00)

8.2.4 TLS CCM Parameters

The following parameters are used for the CCM AEAD cipher in the TLS-PSK and TLS-ECC cipher suites, as specified in [RFC 5116]:

Parameter	Value	Description
M	8	MIC length
L	3	Length length

8.3 Example Transactions

The transactions are generally layered:

- TLS Records
- EAP Packets
- PANA Packets

The PANA session wraps the EAP session, which wraps the TLS handshake transactions.

8.3.1 Syntax

The syntax used is similar to C structure syntax. All fields are clearly sized and where the field value is fixed for the packet, the value is stated.

8.3.2 TLS

TLS Records are typically concatenated as described in the handshake transactions. Each record contains plaintext data for the TLS Handshake and TLS Change Cipher Spec records and ciphertext data for TLS Handshake records.

8.3.3 EAP

EAP packets carry the request and the responses between the EAP entities, i.e. Peer and Authenticator. The EAP protocol allows packets to be fragmented and reassembled. EAP-TLS is the specific EAP method used which encapsulates TLS records into the EAP protocol and defines key derivation.

8.3.4 PANA

The PANA packet transactions form the basis of transportation of the higher layer packets. PANA transactions can occur between PANA client (PaC) and PANA Authentication Agent (PAA) and can be relayed via a PANA Relay Entity (PRE).

The PANA session for a PaC to a PAA is shown below. A relayed session essentially carries the same EAP and TLS information but the PANA session is between three entities.

The sequence shown assumes that the EAP Response can be piggy-backed on the PANA answer. This may not always be the case and the implementation SHOULD assume that an EAP Response may alternatively be carried in a separate PAR initiated by the PaC followed by a PAA from the PAA.

PANA packets SHOULD be a multiple of 4 bytes in size

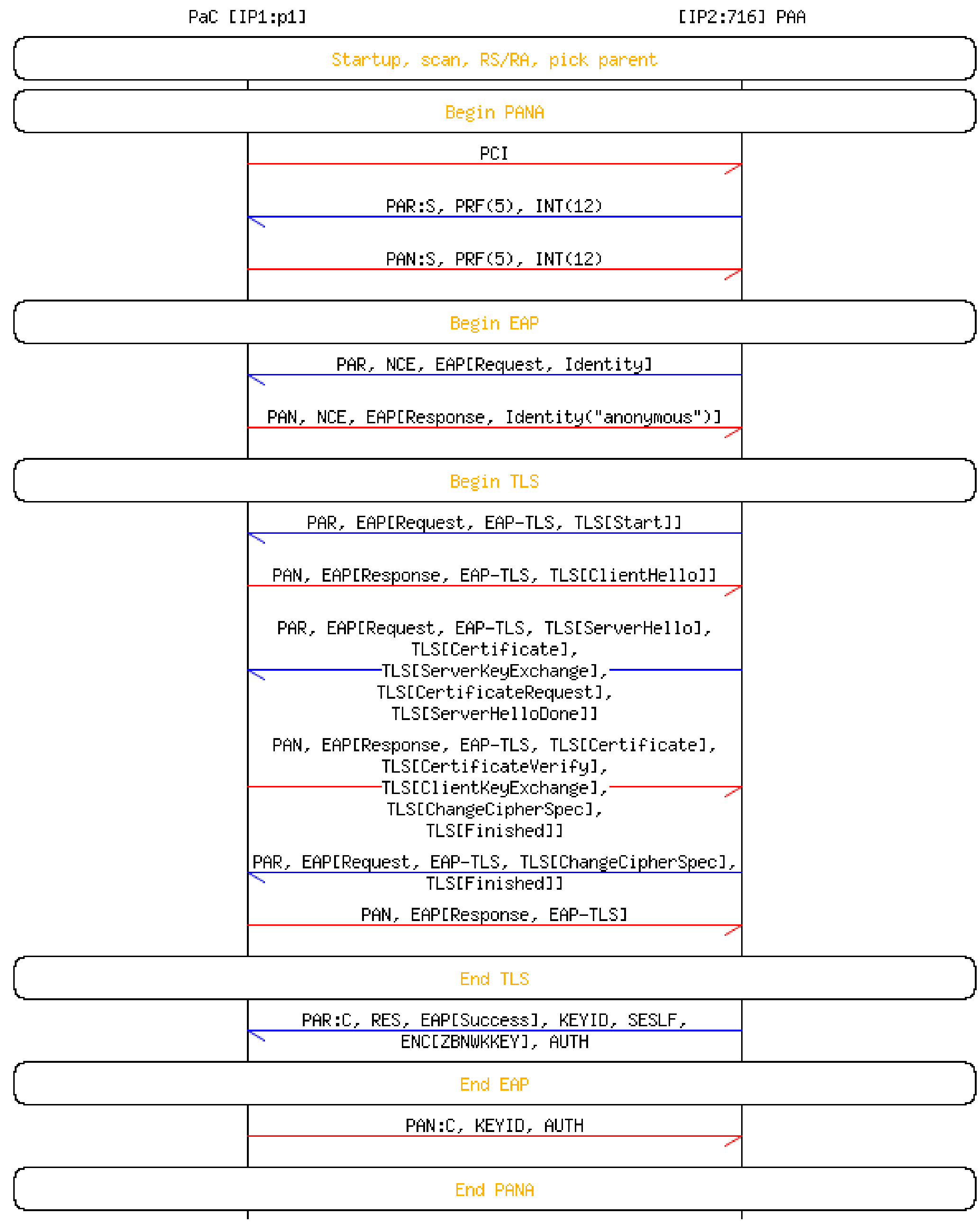

Figure 12: ECC PANA exchange

8.3.5 PCI from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 16; /* 16H */
   uint16 flags = 0x0000;
   uint16 type = 1; /* PCI */
   uint32 session_id = 0;
   uint32 seq_no = 0;
};
```

8.3.6 PANA start from PAA to PaC

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 52; /* 16H + (8H + 4P) + (8H + 4P) + (8H + 4P) */
   uint16 flags = 0xC000; /* Request, start */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id; /* Chosen by PAA */
   uint32 seq_no = paa_seq_no; /* Random number chosen by PAA */
   /* If PRF_HMAC_SHA2_256 is the only PRF, the following AVP may be
optional */
   struct PANAAVP {
      uint16 code = 6; /* PRF algorithm */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 prf_algorithm = 5;
   }
   /* If AUTH_HMAC_SHA2_256_128 is the only integrity algorithm, the
following AVP may be optional */
   struct PANAAVP {
      uint16 code = 3; /* Integrity algorithm */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 integrity_algorithm = 12;
   }
   /* If AES-CTR is the only encryption, the following AVP may be optional
*/
   struct PANAAVP {
      uint16 code = 12; /* Encryption algorithm */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 encryption_algorithm = 1;
   }
};
```

8.3.7 PANA Start from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 52; /* 16H + (8H + 4P) + (8H + 4P) + (8H + 4P) */
   uint16 flags = 0x4000; /* Answer, Start */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id; /* Returned by PaC */
   uint32 seq_no = paa_seq_no; /* Returned by PaC */
   /* If PRF_HMAC_SHA2_256 is the only PRF, the following AVP may be
optional */
   struct PANAAVP {
```



```
        uint16 code = 6; /* PRF algorithm */
        uint16 flags = 0;
        uint16 length = 4;
        uint16 rsvd = 0;
        uint32 prf_algorithm = 5;
    }
    /* If AUTH_HMAC_SHA2_256_128 is the only integrity algorithm, the
following AVP may be optional */
    struct PANAAVP {
        uint16 code = 3; /* Integrity algorithm */
        uint16 flags = 0;
        uint16 length = 4;
        uint16 rsvd = 0;
        uint32 integrity_algorithm = 12;
    }
    /* If AES-CTR is the only encryption, the following AVP may be optional
*/
    struct PANAAVP {
        uint16 code = 12; /* Encryption algorithm */
        uint16 flags = 0;
        uint16 length = 4;
        uint16 rsvd = 0;
        uint32 encryption_algorithm = 1;
    }
};
```

8.3.8 EAP Identity Request from PAA to PaC

```
struct PANA {
    uint16 rsvd = 0;
    uint16 length = 56; /* 16 + (8H + 16P) + (8H + 5P + 3Pd) */
    uint16 flags = 0x8000; /* Request */
    uint16 type = 2; /* PA */
    uint32 session_id = paa_session_id;
    uint32 seq_no = paa_seq_no + 1; /* Increment sequence number */
    struct PANAAVP {
        uint16 code = 5; /* Nonce */
        uint16 flags = 0;
        uint16 length = 16;
        uint16 rsvd = 0;
        uint8 nonce[16];
    }
    /* The following AVP may be optional */
    struct PANAAVP {
        uint16 code = 2; /* EAP Payload */
        uint16 flags = 0;
        uint16 length = 5; /* 5P */
        uint16 rsvd = 0;
        struct EAPReqUnfrag {
            uint8 code = 1; /* EAPReq */
            uint8 identifier = idseq;
            uint16 length = 5; /* inc. 5H + 0P */
            uint8 type = 1; /* EAP-Identity */
        };
        struct AVPPad {
        uint8 bytes[3];
     };
    };
};
```

8.3.9 EAP Identity Response from PaC

```
struct PANA {
    uint16 rsvd = 0;
    uint16 length = 64; /* 16H + (8H + 16P) + (8H + 14P + 2Pd) */
    uint16 flags = 0x0000; /* Answer */
    uint16 type = 2; /* PA */
    uint32 session_id = paa_session_id; /* Returned by PaC */
    uint32 seq_no = paa_seq_no + 1; /* Returned by PaC */
    struct PANAAVP {
        uint16 code = 5; /* Nonce */
        uint16 flags = 0;
        uint16 length = 16;
        uint16 rsvd = 0;
        uint8 nonce[16];
    }
    /* The following AVP may be optional */
    struct PANAAVP {
        uint16 code = 2; /* EAP Payload */
        uint16 flags = 0;
        uint16 length = 14;
        uint16 rsvd = 0;
        struct EAPRspUnfrag {
            uint8 code = 2; /* EAPRsp */
            uint8 identifier = idseq; /* Corresponds to request */
            uint16 length = 14; /* inc. 5H + 9P */
            uint8 type = 1; /* EAP-Identity */
            /* Anonymous NAI */
            uint8 identity[] = "anonymous";
        };
        struct AVPPad {
        uint8 bytes[2];
     };
    };
};
```

8.3.10 TLS Start from PAA to PaC

```
struct PANA {
    uint16 rsvd = 0;
    uint16 length = 32; /* 16H + (8H + 6P + 2Pd) */
    uint16 flags = 0x8000; /* Request */
    uint16 type = 2; /* PA */
    uint32 session_id = paa_session_id;
    uint32 seq_no = paa_seq_no + 2; /* Increment sequence number */
    struct PANAAVP {
        uint16 code = 2; /* EAP Payload */
        uint16 flags = 0;
        uint16 length = 6;
        uint16 rsvd = 0;
        struct EAPReqUnfrag {
            uint8 code = 1;
            uint8 identifier = idseq + 1;
            uint16 length = 6; /* inc. 6H + 0P */
            uint8 type = 13; /* EAP-TLS */
            uint8 flags = 0x20; /* Start */
        };
        struct AVPPad {
        uint8 bytes[2];
     };
    };
```

```
};
```

8.3.11 PSK TLS ClientHello from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 80; /* 16H + (8H + 56P) */
   uint16 flags = 0x0000; /* Answer */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id; /* Returned by PaC */
   uint32 seq_no = paa_seq_no + 2; /* Returned by PaC */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 56;
      uint16 rsvd = 0;
      struct EAPRspUnfrag {
         uint8 code = 2;
         uint8 identifier = idseq + 1; /* Corresponds to request */
         uint16 length = 56; /* inc. 6H + (5H + 45P) */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
         struct TLSPlaintext {
            uint8 type = 22; /* Handshake */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 45; /* 4H + 41P */
            struct Handshake {
               uint8 msg_type = 1; /* ClientHello */
               uint24 length = 41; /* 2P + 32P + 1P + 4P + 2P */
               struct ClientHello {
                  struct ProtocolVersion {
                     uint8 major = 0x03;
                     uint8 minor = 0x03; /* TLS 1.2? */
                  } client_version;
                  struct Random {
                     uint32 gmt_unix_time;
                     uint8 random_bytes[28];
                  } random;
                  struct SessionID<0..32> {
                     uint8 length = 0; /* NULL */
                  } session_id;
                  struct <2..2^16-2> {
                     uint16 length = 2;
                     struct CipherSuite {
                        uint8 bytes[2] = {0x00, 0xC6};
                     } cipher_suites[1];
                  };
                  struct <1..2^8-2> {
                     uint8 length = 1;
                     uint8 compression_methods[1] = {0};
                  }
                  /* NOTE: extensions will be needed for public key cipher
suite */
                  struct { }; /* No extensions */
               };
            };
         };
      };
   };
};
```

8.3.12 ECC TLS ClientHello from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 108; /* 16H + (8H + 82P + 2Pd) */
   uint16 flags = 0x0000; /* Answer */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id; /* Returned by PaC */
   uint32 seq_no = paa_seq_no + 2; /* Returned by PaC */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 82;
      uint16 rsvd = 0;
      struct EAPRspUnfrag {
         uint8 code = 2;
         uint8 identifier = idseq + 1; /* Corresponds to request */
         uint16 length = 82; /* inc. 6H + (5H + 77P) */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
         struct TLSPlaintext {
            uint8 type = 22; /* Handshake */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 71; /* 4H + 67P */
            struct Handshake {
               uint8 msg_type = 1; /* ClientHello */
               uint24 length = 67; /* 2P + 32P + 1P + 8P + 2P + 22P */
               struct ClientHello {
                  struct ProtocolVersion {
                     uint8 major = 0x03;
                     uint8 minor = 0x03; /* TLS 1.2? */
                  } client_version;
                  struct Random {
                     uint32 gmt_unix_time;
                     uint8 random_bytes[28];
                  } random;
                  struct SessionID<0..32> {
                     uint8 length = 0; /* NULL */
                  } session_id;
                  struct <2..2^16-2> {
                     uint16 length = 4;
                     struct CipherSuite {
                        uint8 bytes[2] = {0xC0, 0xC6};
                     } cipher_suites[1];
                     struct CipherSuite {
                        uint8 bytes[2] = {0x00, 0xC6};
                     } cipher_suites[1];
                  };
                  struct <1..2^8-2> {
                     uint8 length = 1;
                     uint8 compression_methods[1] = {0};
                  }
                  struct { /* ECC extensions */
                     uint16 length = 22;
                     struct EllipticCurvesExtension {
                        uint16 type = 10; /* elliptic_curves */
                        uint16 length = 4;
                        uint16 eclength = 2;
                        uint16 ec = 23; /* secp256r1 */
                     };
                     struct ECPointFormatsExtension {
```

```
                        uint16 type = 11; /* ec_point_formats */
                        uint16 length = 2;
                        uint8 pflength = 1;
                        uint8 pf = 0; /* uncompressed */
                     };
                     struct SignatureAlgorithmsExtension {
                        uint16 type = 13; /* signature_algorithms */
                        uint16 length = 4; /* 2? */
                        struct <2..2^16-2> {
                           uint16 length = 2;
                           struct SignatureAndHashAlgorithm {
                              uint8 hash = 0x04; /* sha256 */
                              uint8 signature = 0x03; /* ecdsa */
                           } signature_and_hash_algorithm[1];
                        };
                     };
             };
               };
            };
         };
      };
      struct AVPPad {
      uint8 bytes[2];
    };
   };
};
```

8.3.13 PSK TLS ServerHello and ServerHelloDone from PAA to PaC

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 88; /* 16H + (8H + 61P + 3Pd) */
   uint16 flags = 0x8000; /* Request */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id;
   uint32 seq_no = paa_seq_no + 3; /* Increment sequence number */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 61;
      uint16 rsvd = 0;
      struct EAPReqUnfrag {
         uint8 code = 1;
         uint8 identifier = idseq + 2;
         uint16 length = 61; /* inc. 6H + (5H + 50P) */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
         struct TLSPlaintext {
            uint8 type = 22; /* Handshake */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 50; /* (4H + 42P) + (4H + 0P) */
            struct Handshake {
               uint8 msg_type = 2; /* ServerHello */
               uint24 length = 42; /* 2P + 32P + 5P + 2P + 1P */
               struct ServerHello {
                  struct ProtocolVersion {
                     uint8 major = 0x03;
                     uint8 minor = 0x03; /* TLS 1.2? */
                  } server_version;
                  struct Random {
```

```
                     uint32 gmt_unix_time;
                     uint8 random_bytes[28];
                  } random;
                  struct SessionID<0..32> {
                     uint8 length = 4; /* Arbitrary for now */
                     uint8 bytes[4];
                  } session_id;
                  struct CipherSuite {
                     uint8 bytes[2] = {0x00, 0xC6};
                  } cipher_suite;
                  uint8 compression_method = {0};
                  /* NOTE: extensions will be needed for public key cipher
suite */
                  struct { }; /* No extensions */
               };
            };
            struct Handshake {
               uint8 msg_type = 14; /* ServerHelloDone */
               uint24 length = 0;
               struct ServerHelloDone { }; /* Empty */
            };
         };
      };
      struct AVPPad {
      uint8 bytes[3];
      };
   };
};
```

8.3.14 ECC TLS ServerHello, Certificate, ServerKeyExchange, CertificateRequest and ServerHelloDone from PAA to PaC

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 844; /* 16H + (8H + 61P + 3Pd) */
   uint16 flags = 0x8000; /* Request */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id;
   uint32 seq_no = paa_seq_no + 3; /* Increment sequence number */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 820;
      uint16 rsvd = 0;
      struct EAPReqUnfrag {
         uint8 code = 1;
         uint8 identifier = idseq + 2;
         uint16 length = 820; /* inc. 6H + (5H + 50P) */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
         struct TLSPlaintext {
            uint8 type = 22; /* Handshake */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 50; /* (4H + 42P) + (4H + 0P) */
            struct Handshake {
               uint8 msg_type = 2; /* ServerHello */
               uint24 length = 78; /* 2P + 32P + 5P + 2P + 1P */
               struct ServerHello {
                  struct ProtocolVersion {
                     uint8 major = 0x03;
```



```
         uint8 minor = 0x03; /* TLS 1.2? */
      } server_version;
      struct Random {
         uint32 gmt_unix_time;
         uint8 random_bytes[28];
      } random;
      struct SessionID<0..32> {
         uint8 length = 32; /* Arbitrary for now */
         uint8 bytes[32];
      } session_id;
      struct CipherSuite {
         uint8 bytes[2] = {0xC0, 0xC6};
      } cipher_suite;
      uint8 compression_method = {0};
      struct { /* ECC extensions */
         uint16 length = 6;
         struct ECPointFormatsExtension {
            uint16 type = 11; /* ec_point_formats */
            uint16 length = 2;
            uint8 pflength = 1;
            uint8 pf = 0; /* uncompressed */
         };
      };
   };
};
struct Handshake {
   uint8 msg_type = 11; /* Certificate */
   uint24 length = 559;
   uint24 certificates_length = 556;
   uint24 certificate length = 553;
   uint8 certificate[0][553]; /* Single certificate */
};
struct Handshake {
   uint8 msg_type = 12; /* ServerKeyExchange */
   uint24 length = 144;
   uint8 server_key_exchange[144]; /* Single certificate */
   struct ServerHelloDone { }; /* Empty */
};
struct Handshake {
   uint8 msg_type = 13; /* CertificateRequest */
   uint24 length = 10;
   struct <2..2^8-1> {
      uint8 length = 1;
      uint8 certificate_types = 0x40; /* ecdsa_sign */
   };
   struct <2..2^16-2> {
      uint16 length = 2;
      struct SignatureAndHashAlgorithm {
         uint8 hash = 0x04; /* sha256 */
         uint8 signature = 0x03; /* ecdsa */
      } signature_and_hash_algorithm[1];
   };
   struct <2..2^16-1> {
      uint16 length = 0;
   };
};
struct Handshake {
   uint8 msg_type = 14; /* ServerHelloDone */
   uint24 length = 0;
   struct ServerHelloDone { }; /* Empty */
```



```
                };
            };
        };
        struct AVPPad {
        uint8 bytes[3];
        };
    };
};
```

8.3.15 TLS ClientKeyExchange and ChangeCipherSpec and Finished from PaC to PAA

```
struct PANA {
    uint16 rsvd = 0;
    uint16 length = 88; /* 16H + (8H + 62P + 2Pd) */
    uint16 flags = 0x0000; /* Answer */
    uint16 type = 2; /* PA */
    uint32 session_id = paa_session_id; /* Returned by PaC */
    uint32 seq_no = paa_seq_no + 3; /* Returned by PaC */
    struct PANAAVP {
        uint16 code = 2; /* EAP Payload */
        uint16 flags = 0;
        uint16 length = 62;
        uint16 rsvd = 0;
        struct EAPRspUnfrag {
            uint8 code = 2;
            uint8 identifier = idseq + 2; /* Corresponds to request */
            uint16 length = 62; /* inc. 6H + (5H + (4H + 4P)) + (5H + 1P) +
(5H + 32P) */
            uint8 type = 13; /* EAP-TLS */
            uint8 flags = 0x00;
            struct TLSPlaintext{
                uint8 type = 22; /* Handshake */
                uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
                uint16 length = 4;
                struct Handshake {
                    uint8 msg_type = 16; /* ClientKeyExchange */
                    uint24 length = 4;
                    struct ClientKeyExchange {
                        struct <0..2^16-1> {
                            uint16 length = 2;
                            uint8 bytes[1] = {0x30, 0x00};
                        } psk_identity;
                    } ;
                } ;
            } ;
            struct TLSPlaintext{
                uint8 type = 20; /* ChangeCipherSpec */
                uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
                uint16 length = 1;
                struct ChangeCipherSpec{
                    uint8 type = 1;    /* ChangeCipherSpec */
                };
            };
            struct TLSCiphertext {
                uint8 type = 22; /* Handshake */
                uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
                uint16 length = 32;
                struct GenericAEADCipher {
                    struct CCMNonceExplicit {
```

```
                  uint64 seq_num;
               };
               struct CCMCipherText { /* inferred from draft-mcgrew-tls-
aes-ccm */
                  struct Handshake { /* Encrypted */
                     uint8 msg_type = 20; /* Finished */
                     uint24 length = 12;
                     struct Finished {
                        uint8 verify_data[12];
                     ;}
                  };
                  uint8 MAC[8]; /* Using AES_CCM_8 */
               };
            };
         };
      };
      struct AVPPad {
      uint8 bytes[2];
    };
   };
};
```

8.3.16 TLS ChangeCipherSpec and TLS Finished from PAA to PaC

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 134; /* 16H + (8H + 49P + 0Pd) */
   uint16 flags = 0x8000; /* Request */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id;
   uint32 seq_no = paa_seq_no + 4; /* Increment sequence number */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 49;
      uint16 rsvd = 0;
      struct EAPReqUnfrag {
         uint8 code = 1;
         uint8 identifier = idseq + 3;
         uint16 length = 49; /* inc. 6H + (5H + 1P) + (5H + 32P) */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
         struct TLSPlaintext{
            uint8 type = 20; /* ChangeCipherSpec */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 1;
            struct ChangeCipherSpec{
               uint8 type = 1;    /* ChangeCipherSpec */
            };
         };
         struct TLSCiphertext {
            uint8 type = 22; /* Handshake */
            uint8 version[2] = {0x03, 0x03}; /* TLS 1.2 */
            uint16 length = 32;
            struct GenericAEADCipher {
               struct CCMNonceExplicit {
                  uint64 seq_num;
               };
               struct CCMCipherText { /* inferred from draft-mcgrew-tls-
aes-ccm */
```

```
                      struct Handshake { /* Encrypted */
                         uint8 msg_type = 20; /* Finished */
                         uint24 length = 12;
                         struct Finished {
                            uint8 verify_data[12];
                         };
                      };
                      uint8 MAC[8]; /* Using AES_CCM_8 */
                   };
                };
             };
          };
       };
    };
```

8.3.17 Final EAP response from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 30; /* 16H + (8H + 6P + 2Pd) */
   uint16 flags = 0x0000; /* Answer */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id; /* Returned by PaC */
   uint32 seq_no = paa_seq_no + 4; /* Returned by PaC */
   struct PANAAVP {
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 6;
      uint16 rsvd = 0;
      struct EAPRspUnfrag {
         uint8 code = 2;
         uint8 identifier = idseq + 3; /* Corresponds to request */
         uint16 length = 6; /* inc. 6H + 0P */
         uint8 type = 13; /* EAP-TLS */
         uint8 flags = 0x00;
      };
      struct AVPPad {
      uint8 bytes[2];
    };
   };
};
```

8.3.18 PANA Complete and EAP Success from PAA to PaC

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 128; /* 16H + (8H + 4P) + (8H + 4P) + (8H + 4P) + (8H +
4P) + (8H + (12H + 18P + 2Pd) + (8H + 16P) */
   uint16 flags = 0xA000; /* Request, Complete */
   uint16 type = 2; /* PA */
   uint32 session_id = paa_session_id;
   uint32 seq_no = paa_seq_no + 5; /* Increment sequence number */
   struct PANAAVP {
      uint16 code = 7; /* Result code */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 result_code = 0; /* PANA_SUCCESS */
   };
   struct PANAAVP {
```

```
      uint16 code = 2; /* EAP Payload */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      struct EAPSuccess {
          uint8 code = 3;
          uint8 identifier = idseq + 4;
          uint16 length = 4; /* inc. 4H + 0P */
      };
   };
   struct PANAAVP {
      uint16 code = 4; /* Key ID */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 key_id = 0; /* Initial MSK */
   };
   struct PANAAVP {
      uint16 code = 8; /* Session Lifetime */
      uint16 flags = 0;
      uint16 length = 4;
      uint16 rsvd = 0;
      uint32 sess_life = 0xFFFFFFFF; /* -1 = forever (136 years) */
   };
   struct PANAAVP {
      uint16 code = 13; /* Encrypted Encapsulation */
      uint16 flags = 0;
      uint16 length = 32;
      uint16 rsvd = 0;
      struct PANAAVP {
        uint16 code = 1; /* ZigBee Network Key */
        uint16 flags = 1; /* Vendor specific */
        uint16 length = 18;
        uint16 rsvd = 0;
        uint32 vendor_id = 37244; /* ZigBee Vendor ID */
       struct ZBNWKKEY {
          uint8 nwk_key[16];
          uint8 nwk_key_idx;
          uint8 auth_cntr;
       };
        struct AVPPad {
        uint8 bytes[2];
      };
     };
   };
   struct PANAAVP {
      uint16 code = 1; /* Auth */
      uint16 flags = 0;
      uint16 length = 16;
      uint16 rsvd = 0;
      uint8 auth[16]; /* Hash */
   };
};
```

8.3.19 PANA Complete from PaC to PAA

```
struct PANA {
   uint16 rsvd = 0;
   uint16 length = 54; /* 16H + (8H + 4P) + (8H + 16P) */
   uint16 flags = 0x2000; /* Answer, Complete */
```



```
    uint16 type = 2; /* PA */
    uint32 session_id = paa_session_id; /* Returned by PaC */
    uint32 seq_no = paa_seq_no + 5; /* Returned by PaC */
    struct PANAAVP {
        uint16 code = 4; /* Key ID */
        uint16 flags = 0;
        uint16 length = 4;
        uint16 rsvd = 0;
        uint32 key_id = 0; /* Initial MSK */
    };
    struct PANAAVP {
        uint16 code = 1; /* Auth */
        uint16 flags = 0;
        uint16 length = 16;
        uint16 rsvd = 0;
        uint8 auth[16]; /* Hash */
    };
};
```

