IEEE P802.11
Wireless LANs

|  |
| --- |
| Resolution of Security Comments from the First Sponsor Ballot |
| Date: 2010-11-23 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | Aruba Networks | 1322 Crossman ave, Sunnyvale, CA  | +1 408 227 4500 |  |
|  |  |  |  |  |

Abstract

This document proposes resolution to CIDs: 1, 2, 3, 4, 5, 6, 94, 95, 96, 130, 131, 158, 159, 231, 250, 251, 252, 283, 284, 285, 286, 287, 288, 289, 290, 292, 296, 297, 298, 302.

Change the sixth paragraph in 5.4.3.1 as follows:

* Authentication

Either SAE authentication or ~~T~~the Open System authentication algorithm is used in RSNs based on infrastructure BSS and IBSS, although Open System authentication is optional in an RSN based on an IBSS. SAE authentication is used in an MBSS. RSNA disallows the use of Shared Key authentication.

***Change section 5.4.3.2 as indicated:***

* Deauthentication

When the deauthentication service is terminating SAE authentication any PTKSA or GTKSA related to this SAE authentication shall be destroyed. If PMK caching is not enabled, deauthentication also destroys any PMKSA created as a result of this successful SAE authentication.

***Modify section 8.2a.2 as indicated:***

* Assumptions on SAE

SAE uses various functions and data to accomplish its task and assumes certain properties about each function. These are as follows:

* H is an “extractor” function (see IETF RFC 5869) that concentrates potentially dispersed entropy from an input to create an output that is a cryptographically strong, pseudo-random key. This function takes as input a non-secret “salt” and a secret input and produces a fixed-length output.
* CN is a confirmation function which takes a secret key and data to confirm and bind to the exchange.
* A finite cyclic group is negotiated for which solving the discrete logarithm problem is computationally infeasible.

When used with AKMs 8 or 9 from Table 7-34 (AKM suite selectors), H shall be instantiated as HMAC-SHA256:

* H(salt, ikm) = HMAC-SHA256(salt, ikm)

When used with AKMs 8 or 9 from Table 7-34 (AKM suite selectors), CN shall be instantiated as a function that takes a key and a sequence of data. Each piece of data is converted to an octet string and concatenated together before being passed, along with the key, to HMAC-SHA256:

CN(key, X, Y, Z, …) = HMAC-SHA256(key, D2OG(X) || D2OG(Y) || D2OG(Z) || …)

where D2OG() represents the data to octet string conversion functions in 8.2a.7.2 (Data type conversion).

Other instantiations of functions H and CN require creation of a new AKM identifier.

***Modify section 8.2a.4.1.2 as indicated:***

* Generation of the Password Element with ECC groups

The Password Element of an ECC group (***PWE***) shall be generated in a random hunt-and-peck fashion. A counter, represented as a single octet and initially set to one (1), is used with the peer identities and the password to generate a password seed. The password seed shall then be stretched using the key derivation function (KDF)(Ed) from 8.5.1.5.2 to the bit length of the prime number from the group definition with the Label of “SAE Hunting and Pecking” and the Context being the prime.

***Modify section 8.2a.4.2.1 as indicated:***

* FFC group definition

Domain parameters for FFC groups include a generator ***G***, a prime *p*, and an order *r*. An element, *B*, in an FFC group satisfies ***B*** = ***G****i* modulo *p* for some integer *i*.This special property differentiates elements from scalars, even though both elements and scalars can be represented as numbers less than the prime modulus. The notation convention of 8.2a.4 signifies this difference between an element and a scalar in an FFC group. The identity element for an FFC group is the value one (1).

***Modify section 8.2a.5.1 as indicated***

* Message exchanges

The protocol consists of two message exchanges, a commitment exchange and a confirmation exchange. The rules for performing these exchanges are specified by the finite state machine in 8.2a.8.

When a party has sent its message in the commit exchange it is said to have *committed* and when it has sent its message in the confirmation exchange it has *confirmed*. The following rules can be ascribed to the protocol:

* A party can *commit* at any time
* A party can *confirm* after it has *committed* and its peer has *committed*
* A party can *accept* authentication after a peer has *confirmed*
* The protocol successfully *terminates* after each peer has *accepted*

***Split section 8.2a.5.2 into 2, creating a new 8.2a.5.3 and making the the existing 8.2a.5.3 into 8.2a.5.4 and similarly incrementing the existing 8.2a.5.4 and 8.2a.5.5.***

* *PWE* and Secret Generation

Prior to beginning the protocol message exchange, the secret element ***PWE*** and two secret values are generated. First, a group is selected, eitherthe most preferred group if the STA is initiating SAE to a peer, or the group from a received Commit Message if the STA is responding to a peer. The ***PWE*** shall be generated for that group (according to 8.2a.4.1.2 (Generation of the Password Element with ECC groups) or 8.2a.4.2.2 (Generation of the Password Element with FFC groups), depending on whether the group is ECC or FFC, respectively) using the identities of the two STAs and the configured password.

After generation of the ***PWE***, each STA shall generate a secret value, *rand*, and a temporary secret value, *mask*, each of which shall be chosen randomly such that 1 < *rand*, *mask* < *r*, the order of the group.

**8.2a.5.3 Construction of a Commit Message**

A Commit Message consists of a scalar and an element that shall be produced using the ***PWE*** and secrets generated in 8.2a.5.2, as follows:

 *commit*-*scalar* = (*rand* + *mask*) modulo *r*

 ***COMMIT-ELEMENT*** (Ed)= inverse(scalar-op(*mask*,***PWE***))

This message shall be transmitted to the peer as described in 8.2a.7 (Framing of SAE). The temporary secret *mask* may be destroyed at this point.

***Modify sections 8.2a.5.4 and 8.2a.5.5 as indicated:***

* Construction of a Confirm Message

A peer generates a Confirm Message by passing the KCK, the current value of the *send-confirm* counter (see 7.3.1.35 (Send-Confirm field)), the scalar and element from the sent Commit Message, and the scalar and element from the received Commit Message to the confirmation function CN.

 *confirm* = CN(KCK*,* *send-confirm, commit*-*scalar,* ***COMMIT-ELEMENT,***(Ed) *peer*-*commit*-*scalar,*

***PEER-COMMIT-ELEMENT***(Ed))

The message shall be transmitted to the peer as described in 8.2a.7 (Framing of SAE).

* Processing of a peer’s Confirm Message

Upon receipt of a peer’s Confirm Message a *verifier* is computed, which is the expected value of the peer’s confirmation, *peer*-*confirm*, extracted from the received Confirm Message. The *verifier* is computed by passing the KCK, the peer’s send-confirm counter from the received Confirm Message (see 7.3.1.35 (Send-Confirm field)), the scalar and element from the received Commit Message, and scalar and element from the sent Commit Message to the confirmation function CN.

 *verifier* = CN(KCK*, peer-send*-*confirm,* *peer*-*commit*-*scalar,*

 ***PEER-COMMIT-ELEMENT,***(Ed)*commit*-*scalar,****COMMIT-ELEMENT***(Ed))

If the *verifier* equals *peer-confirm,* the STA shall accept the peer’s authentication and set the lifetime of the PMK to the minimum of the lifetime of the password used to generate ***PWE*** and the value dot11RSNAConfigPMKLifetime. If the *verifier* differs from the *peer*-*confirm,* the STA shall reject the peer’s authentication and destroy the PMK.

***Modify section 8.2a.8.1 as indicated:***

* General

The protocol is instantiated by the finite state machine in Figure s8-3a (SAE finite state machine). Each instance of the protocol is identified by a tuple consisting of the local MAC address and the peer MAC address. The model in which SAE is defined consists of a parent process, managed by the SME, which receives messages, and dispatches them to the appropriate protocol instance, also managed by the SME. The parent process manages a database of protocol instances indexed by the peer identity. Protocol instances maintain state, receive events from the parent process, send events to itself, and output data.

***Modify section 8.2a.8.6.2b as indicated:***

* Nothing state

In *Nothing* state a protocol instance has just been allocated.

Upon receipt of an *Init* event, the protocol instance shall zero its *Sync* variable, *Rc*, and *Sc* variables, select a group from local configuration and generate the ***PWE*** and the secret values according to 8.2a5.2, generate a Commit Message (see 8.2a.5.2 (Construction of a Commit Message)), and set its *t0* (retransmission) timer. The protocol instance transitions into *Committed* state.

Upon receipt of a *Com* event, the protocol instance shall check the Status of the Authentication frame. If the Status code is non-zero, the frame shall be silently discarded and a *Del* event shall be sent to the parent process.Otherwise, the frame shall be processed by first checking the finite cyclic group field to see if the requested group is supported. If not, *BadGrp* shall be set and the protocol instance shall construct and transmit an Authentication frame with Status code 77 indicating rejection with the finite cyclic group field set to the rejected group, and shall send the parent process a *Del* event. If the group is supported, the protocol instance shall zero the *Sc* and *Rc* counters and it shall generate the ***PWE*** and the secret values according to 8.2a.5.2. It shall thenprocess the received Commit Message (see 8.2a.5.3 (Processing of a peer’s Commit Message)). If validation of the received Commit Message fails, the protocol instance shall send a Del event to the parent process, otherwise it shall construct and transmit a Commit Message (see 8.2a.5.2 (Construction of a Commit Message)) followed by a Confirm Message (see 8.2a.5.4 (Construction of a Confirm Message)). The *Sync* counter shall be set to zero and the t0 (retransmission) timer shall be set. The protocol instance transitions to *Confirmed*.

NOTE—A protocol instance in Nothing state will never receive a Confirm Message due to state machine behavior of the parent process.

***Modify section 8.2a.8.6.2c as indicated:***

* Committed state

In *Committed* state, a protocol instance has sent its peer a Commit Message but has yet to receive (and accept) anything.

Upon receipt of a *Com* event, the t0 (retransmission) timer shall be cancelled. Then the following is performed:

* The protocol instance shall check the Status code of the Authentication frame. If the Status code is 76, a new Commit Message shall be constructed with the Anti-Clogging Token from the received Authentication frame, and the *commit-scalar* and ***COMMIT-ELEMENT***(Ed) previously sent. The new Commit Message shall be transmitted to the peer, *Sync* shall be zeroed, and the t0 (retransmission) timer shall be set.
* If the Status code is 77, the protocol instance shall check the finite cyclic group field being rejected. If the rejected group does not match the last offered group the protocol instance shall silently discard the message and set the t0 (retransmission) timer. If the rejected group matches the last offered group, the protocol instance shall choose a different group; it then generates and transmits a new Commit Message to the peer, zeros *Sync*, sets the t0 (retransmission) timer, and remains in *Committed* state. If there are no other groups to choose, the protocol instance shall send a *Del* event to the parent process and transitions back to *Nothing*.
* If the Status is some other non-zero value, the frame shall be silently discarded and the t0 (retransmission) timer shall be set.
* If the Status is zero, the finite cyclic group field is checked. If the group is not supported, *BadGrp* shall be set and the value of *Sync* shall be checked.
* If *Sync* is greater than dot11RSNASAESync, the protocol instance shall send a *Del* event to the parent process and transitions back to Nothing.
* If *Sync* is not greater than dot11RSNASAESync, *Sync* shall be incremented, a Commit Message with Status code equal to 77 indicating rejection, and the Algorithm identifier set to the rejected algorithm, shall be sent to the peer, the t0 (retransmission) timer shall be set and the protocol instance shall remain in *Committed* state.
* If the group is supported but does not match that used when the protocol instance constructed its Commit Message, *DiffGrp* shall be set and the local identity and peer identity shall be checked.
* The mesh STA, with the numerically greater of the two MAC addresses, drops the received Commit Message, retransmits its last Commit Message, and shall set the t0 (retransmission) timer and remain in *Committed* state.
* The mesh STA, with the numerically lesser of the two MAC addresses, zeros *Sync*, shall increment *Sc*, choose the group from the received Commit Message, generate a new ***PWE*** and new secret values according to 8.2a.5.2, process the received Commit Message according to 8.2a.5.3 (Processing of a peer’s Commit Message), generate a new Commit Message and Confirm Message, and shall transmit the new Commit and Confirm to the peer. It shall then transition to *Confirmed* state.

***Modify section 8.2a.8.6.2e as indicated:***

* Accepted state

In *Accepted* state a protocol instance has sent a Commit Message and a Confirm Message to its peer and received a Commit Message and Confirm Message from the peer. Unfortunately, there is no guarantee that the final Confirm Message sent by the STA was received by the peer.

***Modify sections 10.3.4.1.2 and 10.3.4.1.3 as indicated:***

* Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.request(

PeerSTAAddress,

AuthenticationType,

AuthenticateFailureTimeout,

Content of SAE Authentication Frame,

Content of FT Authentication (#1684)Elements,(11r)

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| PeerSTAAddress | MACAddress | Any valid individual MAC address | Specifies the address of the peer MAC entity with which to perform the authentication process. |
| AuthenticationType | Enumeration | OPEN\_SYSTEM,SHARED\_KEY,FAST\_BSS\_TRANSITION, SAE (11r) | Specifies the type of authentication algorithm to use during the authentication process. |
| AuthenticationFailureTimeout | Integer | 1 | Specifies a time limit (in TU) after which the authentication procedure will be terminated. |
| Content of SAE Authentication Frame | Sequence of octets | As defined in 7.3.1.35 (Send-Confirm Field), 7.3.1.36 (Anti-Clogging Token field), 7.3.1.37 (Scalar Field), 7.3.1.38 ( Element Field), 7.3.1.39 (Confirm Field), and 7.3.1.40 (Finite Cyclic Group Field) | The contents of the SAE Commit Message or SAE Confirm Message. Present only if AuthenticationType indicates SAE authentication. |
| Content of FT Authentication (#1684)Elements(11r) | Sequence of (#1684)elements | As defined in 11A.8 (FT authentication sequence) | The set of (#1684)elements to be included in the first message of the FT authentication sequence, as described in 11A.8.2 (FT authentication sequence: contents of first message). Present only if(#29) dot11FastBSSTransitionActivated(#1005) is(#1217) true(#1535). |
| VendorSpecificInfo | A set of (#1684)elements | As defined in 7.3.2.26 (Vendor Specific element) | Zero or more (#1684)elements. |

* When generated

This primitive is generated by the SME for a STA to establish authentication with a specified peer MAC entity in order to permit Class 2 frames, or Mesh Peering Management frames, to be exchanged between the two STAs. During the authentication procedure, the SME can generate additional MLME-AUTHENTICATE.request primitives.

***Modify section 10.3.4.2.2 as indicated***

* Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.confirm(

PeerSTAAddress,

AuthenticationType,

ResultCode,

Content of SAE Authentication Frame,

Content of FT Authentication (#1684)Elements,(11r)

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| PeerSTAAddress | MACAddress | Any valid individual MAC address | Specifies the address of the peer MAC entity with which the authentication process was attempted. This value must match the peerSTAAddress parameter specified in the corresponding MLME-AUTHENTICATE.request primitive(#2172). |
| AuthenticationType | Enumeration | OPEN\_SYSTEM,SHARED\_KEYFAST\_BSS\_TRANSITIONSAE (11r) | Specifies the type of authentication algorithm that was used during the authentication process. This value must match the authenticationType parameter specified in the corresponding MLME-AUTHENTICATE.request primitive(#2172). |
| ResultCode | Enumeration | SUCCESS, INVALID\_PARAMETERS, TIMEOUT, TOO\_MANY\_SIMULTANEOUS\_ REQUESTS, REFUSED, ANTI-CLOGGING TOKEN REQUIRED, FINITE CYCLIC GROUP NOT SUPPORTED, AUTHENTICATION REJECTED | Indicates the result of the MLME-AUTHENTICATE.request primitive(#2172). |
| Content of SAE Authentication Frame | Sequence of octets | As defined in 7.3.1.35 (Send-Confirm Field), 7.3.1.36 (Anti-Clogging Token field), 7.3.1.37 (Scalar Field), 7.3.1.38 ( Element Field), 7.3.1.39 (Confirm Field), and 7.3.1.40 (Finite Cyclic Group Field) | The contents of the SAE Commit Message or SAE Confirm Message. Present only if AuthenticationType indicates SAE authentication. |
| Content of FT Authentication (#1684)Elements (11r) | Sequence of (#1684)elements | As defined in 11A.8 (FT authentication sequence) | The set of (#1684)elements included in the second message of the FT authentication sequence, as described in 11A.8.3 (FT authentication sequence: contents of second message). Present only if(#29) dot11FastBSSTransitionActivated(#1005) is(#1217) true(#1535). |
| VendorSpecificInfo | A set of (#1684)elements | As defined in 7.3.2.26 (Vendor Specific element) | Zero or more (#1684)elements. |

***Modify section 10.3.4.3.2 as indicated***

* Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.indication(

PeerSTAAddress,

AuthenticationType,

Content of SAE Authentication Frame,

Content of FT Authentication (#1684)Elements,(11r)

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| PeerSTAAddress | MACAddress | Any valid individual MAC address | Specifies the address of the peer MAC entity with which the authentication relationship was established. |
| AuthenticationType | Enumeration | OPEN\_SYSTEM,SHARED\_KEY,FAST\_BSS\_TRANSITION, SAE(11r) | Specifies the type of authentication algorithm that was used during the authentication process. |
| Content of SAE Authentication Frame | Sequence of octets | As defined in 7.3.1.35 (Send-Confirm Field), 7.3.1.36 (Anti-Clogging Token field), 7.3.1.37 (Scalar Field), 7.3.1.38 ( Element Field), 7.3.1.39 (Confirm Field), and 7.3.1.40 (Finite Cyclic Group Field) | The contents of the SAE Commit Message or SAE Confirm Message. Present only if AuthenticationType indicates SAE authentication. |
| Content of FT Authentication (#1684)elements (11r) | Sequence of (#1684)elements | As defined in 11A.8 (FT authentication sequence) | The set of (#1684)elements included in the first message of the FT authentication sequence, as described in 11A.8.2 (FT authentication sequence: contents of first message). Present only if(#29) dot11FastBSSTransitionActivated(#1005) is(#1217) true(#1535). |
| VendorSpecificInfo | A set of (#1684)elements | As defined in 7.3.2.26 (Vendor Specific element) | Zero or more (#1684)elements. |

***Modify 10.3.4.4.2 as indicated***

* Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.response(

PeerSTAAddress,
ResultCode,

Content of SAE Authentication Frame,
Content of FT Authentication (#1684)Elements, (11r)
VendorSpecificInfo
)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| PeerSTAAddress | MACAddress | Any valid individual MAC address | Specifies the address of the peer MAC entity from which the authentication request was received. |
| ResultCode | Enumeration | SUCCESS, REFUSED, ANTI-CLOGGING TOKEN REQUIRED, FINITE CYCLIC GROUP NOT SUPPORTED, AUTHENTICATION REJECTED | Indicates the result response to the authentication request from the peer MAC entity. |
| Content of SAE Authentication Frame | Sequence of octets | As defined in 7.3.1.35 (Send-Confirm Field), 7.3.1.36 (Anti-Clogging Token field), 7.3.1.37 (Scalar Field), 7.3.1.38 ( Element Field), 7.3.1.39 (Confirm Field), and 7.3.1.40 (Finite Cyclic Group Field) | The contents of the SAE Commit Message or SAE Confirm Message. Present only if the AuthenticationType of the MLME-AUTHENTICATE.indication primitive that generated this response indicated SAE authentication. |
| Content of FT Authentication (#1684)elements (11r) | Sequence of (#1684)elements | As defined in 11A.8 (FT authentication sequence) | The set of (#1684)elements to be included in the second message of the FT authentication sequence, as described in 11A.8.3 (FT authentication sequence: contents of second message). Present only if(#29) dot11FastBSSTransitionActivated(#1005) is(#1217) true(#1535). |
| VendorSpecificInfo | A set of (#1684)elements | As defined in 7.3.2.26 (Vendor Specific element) | Zero or more (#1684)elements. |

***Modify section 11.3.5.1.3 as indicated:***

* When generated

This primitive is generated by the SME for a STA to invalidate authentication with a specified peer MAC entity in order to prevent the exchange of Class 2 frames, or Mesh Peering Management frames, between the two STAs. During the deauthentication procedure, the SME can generate additional MLME-DEAUTHENTICATE.request primitives

***Modify section 10.3.73.2.2 and 10.3.73.2.3 as indicated:***

* **Semantics of the service primitive**

The primitive parameters are as follows:

MLME-MeshPeeringManagement.confirm(

 peerMAC,

 ResultCode,

 MeshPeeringMgmtFrameContent

 )

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Valid range** | **Description** |
| peerMAC | MAC Address | Valid individual MAC address | Specifies the address of the peer MAC entity to which the Mesh Peering Management frame was sent. |
| ResultCode | Enumeration | SUCCESS, INVALID\_PARAMETERS, or UNSPECIFIED\_FAILURE | Reports the outcome of the request to send a Mesh Peering Management frame. |
| MeshPeeringMgmtFrameContent(CID303) | Sequence of octets | As defined in 7.4.14.2 (Mesh Peering Open frame format), 7.4.14.3 (Mesh Peering Confirm frame format), or 7.4.14.4 (Mesh Peering Close frame format). | The contents of the Action field of the Mesh Peering Open, Mesh Peering Confirm, or Mesh Peering Close frame to send to the peer MAC entity. |

* **When generated**

This primitive is generated as a result of an MLME-MeshPeeringManagement.request with a specified MAC peer.

***Add new section 10.3.73.4:***

* + - 1. **MLME-MeshPeeringManagement.response**
				1. **Function**

This primitive is used to send a response to a Mesh Peering Management frame to the specified peer MAC entity.

* + - * 1. **Semantics of the service primitive**

The primitive parameters are as follows:

MLME-MeshPeeringManagement.response(

 peerMAC,

 ResultCode,

 MeshPeeringMgmtFrameContent(CID303)

 )

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Valid range** | **Description** |
| peerMAC | MAC Address | Valid individual MAC address | Specifies the address of the peer MAC entity to which the Mesh Peering Management frame is to be sent. |
| ResultCode | Enumeration | SUCCESS, INVALID\_PARAMETERS, or UNSPECIFIED\_FAILURE | Reports the result response to the Mesh Peering Management frame from the peer MAC entity. |
| MeshPeeringMgmtFrameContent(CID303) | Sequence of octets | As defined in 7.4.14.2 (Mesh Peering Open frame format), 7.4.14.3 (Mesh Peering Confirm frame format), or 7.4.14.4 (Mesh Peering Close frame format). | The contents of the Action field of the Mesh Peering Open, Mesh Peering Confirm, or Mesh Peering Close frame to send to the peer MAC entity. |

* + - * 1. **When generated**

This primitive is generated by the SME as a response to an MLME-MeshPeeringManagememt.indication primitive.

* + - * 1. **Effect of receipt**

This primitive indicates secheduling for transmission of a Mesh Peering management frame containing the indicated response.

***Delete section 10.3.74***

***Modify section 11C.3.1 as indicated:***

* General

The Mesh Peering Management framework supports all functions to establish, manage, and tear down mesh peerings. When dot11MeshSecurityActivated is true, the mesh STA shall manage mesh peerings and Mesh TKSAs for each peer mesh STA.

MBSS peering management functions shall be invoked after a candidate peer mesh STA has been discovered via the candidate peer mesh STA discovery procedure described in 11C.2.7 (Candidate peer mesh STA). Mesh STAs shall not transmit frames other than the ones used for candidate peer mesh STA discovery, Mesh Peering Management, and SAE to a neighboring mesh STA until a mesh peering has been established with the mesh STA. Upon successful completion of a mesh peering, mesh STAs may transmit other frames, such as Mesh Action frames, to maintain the integrity of the mesh BSS.

Depending on the setting of dot11MeshSecurityActiviated, one of the following protocols shall be invoked to establish a mesh peering with a candidate peer mesh STA:

* When dot11MeshSecurityActivated is false, the Mesh Peering Management (MPM) protocol is used to establish and manage the mesh peering with the candidate peer mesh STAs. See 11C.4 (Mesh Peering Management) for MPM protocol details.
* When dot11MeshSecurityActivated is true, the peers must establish an authenticated mesh peering using the Authenticated Mesh Peering Exchange (AMPE) protocol . The AMPE protocol requires an existing Mesh PMKSA. If a Mesh PMKSA with the candidate peer mesh STA exists it shall be used directly with AMPE. If no Mesh PMKSA exists the peers must first authenticate with SAE to establish a Mesh PMKSA. See 11C.5 (Authenticated Mesh Peering Exchange for AMPE protocol details.

The Authenticated Mesh Peering Exchange includes the Mesh Peering Management protocol but differs in that it has additional requirements on creation and processing of frames. The successful completion of AMPE establishes the mesh peering and Mesh TKSA with the peer mesh STA, and the mesh TK and MGTKs are installed. Upon failure of AMPE, the mesh STA shall terminate the mesh peering establishment procedure with the current candidate peer mesh STA.

Mesh STAs shall support SAE authentication (see 8.2a (Authentication using a password)) using a pre-shared secret with the candidate peer mesh STA.

***Modify section 11C.3.2.2 as indicated:***

* Creating mesh peering instance and Mesh TKSA for a peer mesh STA

The mesh peering instance controller shall create a new mesh peering instance after successful candidate peer mesh STA discovery identified by the mesh peering instance identifier. It shall generate a new protocol finite state machine for this mesh peering instance and activate the new finite state machine to initiate the mesh peering establishment.

A mesh STA may create multiple mesh peering instances to establish a peering with the same candidate peer mesh STA. However, once a mesh peering is established successfully, all other mesh peering instances with the same peer mesh STA shall be closed. A new mesh peering instance may be started when the mesh STA already maintains a valid mesh peering with the same peer mesh STA, due to the change of some mesh peering parameter. Once the new mesh peering is established successfully, the previous valid mesh peering shall be closed.

***Modify section 11C.4.1 as indicated:***

* General

The Mesh Peering Management protocol is used to establish, maintain, and close mesh peerings between mesh STAs when security is not required.

A mesh STA shall assign a unique AID to every peer mesh STA during the mesh peering establishment procedure. AID is used to encode TIM element in the Beacon frame (see 7.3.2.6 (TIM element)). AID 0 (zero) is reserved to indicate the presence of buffered groupcast MSDUs and MMPDUs.

The mesh STA shall start the Mesh Peering Management protocol in either of the following two events:

* A receipt of a Mesh Peering Open frame from a candidate peer mesh STA
* A receipt of an MLME-MeshPeeringManagement.request, in order to establish a mesh peering with a candidate peer mesh STA

A mesh peering instance ends when the mesh peering is closed. The reasons for closing a mesh peering are outside the scope of this specification.

***Modify section 11C.4.2 as indicated:***

* Pre-processing Mesh Peering Management frames

If the Mesh Peering Open frame is not discarded, the mesh peering instance controller shall generate a new protocol finite state machine and actively reject or accept the mesh peering open request. If dot11MeshAcceptingAdditionalPeerings is set to zero the Mesh Peering Open request shall be rejected with the reason of MESH-MAX-PEERS. A unique local link ID shall be generated for the mesh peering instance. If the mesh peering instance is to be established by Authenticated Mesh Peering Exchange, a random local nonce shall be generated for identifying the mesh peering instance as well. The Mesh Peering Open frame shall be passed to the newly generated mesh peering instance finite state machine for further processing.

The mesh peering open request may be rejected due to an internal reason with a reason code of MESH-PEERING-CANCELED.

If the Mesh Peering Open request is rejected, the REQ\_RJCT event shall be passed with the specified reason code to the newly generated protocol finite state machine to actively reject the mesh peering open request.

NOTE—Example internal reasons to reject new mesh peering request could be the mesh STA has reached its capacity to set up more mesh peering, the mesh STA is configured to reject mesh peering request from another specific peer mesh STA.

If the mesh peering instance controller does not find a matching mesh peering instance, or if the frame was dropped and if the frame is a Mesh Peering Confirm frame or Mesh Peering Close frame, it shall be silently discarded.

***Modify section 11C.4.3.2.2 as indicated:***

* Processing Mesh Peering Open frames

The mesh STA shall first check that the Mesh ID element and Mesh Configuration element of the Mesh Peering Open frame is identical to its own mesh STA configuration as specified in 11C.2.3 (Mesh Profile) and 11C.2.4 (Mesh STA configuration). If a mismatch is found the frame shall be rejected with a reason code of MESH-CONFIGURATION-POLICY-VIOLATION and the mesh peering establishment attempt shall be terminated.

Otherwise, the mesh STA shall accept the Mesh Peering Open frame. The mesh peering instance state shall be updated to include the mesh peering instance identifier and other information from Mesh Configuration element. The mesh STA may also update the mesh peering state based on other parameters in the Mesh Peering Open frame. The Mesh Peering Open frames shall be passed to the corresponding Mesh Peering Management finite state machine for further processing.

***Modify section 11C.4.3.3.1 as indicated:***

* Mesh Peering Confirm frame contents

The Mesh Peering Confirm frame shall contain the following:

* Mesh ID element and Mesh Configuration element shall be set to the same value as in the corresponding Mesh Peering Open frame.
* The Local Link ID field shall be set to the localLinkID. The Peer link ID field shall be set to thepeerLinkID.
* Other elements as defined in 7.4.14.3 (Mesh Peering Confirm frame format) may be present and set to the value according to mesh STA’s parameter configuration.

***Modify section 11C.4.3.3.2 as indicated:***

* Processing Mesh Peering Confirm frames

The mesh STA shall check that the Mesh ID element and Mesh Configuration element of the Mesh Peering Confirm frame match its own mesh STA configuration as specified in 11C.2.3 (Mesh Profile) and 11C.2.4 (Mesh STA configuration). If a mismatch is found, the frame shall be rejected with the reason code of MESH-INCONSISTENT-PARAMETERS and mesh peering establishment shall be terminated.

Otherwise, the mesh STA shall accept the Mesh Peering Confirm frame and shall perform the actions described in 11C.4.4 (the Mesh Peering Management finite state machine). The mesh STA may update the mesh peering state based on other parameters in the Mesh Peering Confirm frame, if present-- see 7.4.14.3.2 (Mesh Peering Confirm frame details).

***Modify section 11C.4.3.4.1 as indicated***

* Mesh Peering Close frame contents

The Mesh Peering Close frame shall contain the following:

* The Mesh Peering Protocol Identifier in the Mesh Configuration element shall be set to 0 “Mesh Peering Management Protocol.”
* Mesh ID shall be set to the value of the Mesh ID in the mesh STA’s mesh profile.
* Mesh Peering Management element shall contain the Local Link ID field, set to the localLinkID. If the mesh STA instance has a peerLinkID, the mesh STA shall include the Peer link ID field in the Mesh Peering Management element and set it to peerLinkID.
* Reason code shall be set to the value that specifies the reason to close the mesh peering instance. See 7.3.1.7 (Reason Code field).

***Modify section 11C.4.4.3 as indicated:***

* Events and actions

The events generated by frame processing are as follows:

* OPN\_ACPT—PeeringOpen\_Accept(peerMAC, peerLinkID) event indicates that a Mesh Peering Open frame meeting the correctness criteria of 11C.4.3.2 (Mesh Peering Open frames) has been received from peerMAC for the mesh peering instance identified by peerLinkID.
* OPN\_RJCT—PeeringOpen\_Reject(peerMAC, peerLinkID, Configuration, reasonCode) event indicates that a Mesh Peering Open frame from peerMAC for the mesh peering instance identified by peerLinkID is rejected due to incomplete or erroneous configuration or other internal reasons. CNF\_ACPT—PeeringConfirm\_Accept(peerMAC, localLinkID, peerLinkID) event indicates that a Mesh Peering Confirm frame meeting the correctness criteria of 11C.4.3.3 (Mesh Peering Confirm frames) has been received from peerMAC for the mesh peering instance identified by localLinkID and peerLinkID.
* CNF\_RJCT—PeeringConfirm\_Reject(peerMAC, localLinkID, peerLinkID, reasonCode) event indicates that a Mesh Peering Confirm frame from peerMAC for the mesh peering instance identified by localLinkID and peerLinkID is rejected due to incomplete or erroneous configuration. CLS\_ACPT—PeeringClose\_Accept(peerMAC, localLinkID, peerLinkID, reasonCode) event indicates that a Mesh Peering Close frame meeting the correctness criteria of 11C.4.3.4 (Mesh Peering Close frames) has been received from peerMAC for the mesh peering instance identified by localLinkID and peerLinkID. The reasonCode specifies the reason that caused the generation of the Mesh Peering Close frame.
* REQ\_RJCT—PeeringRequest\_Reject(peerMAC, peerLinkID, reasonCode) event indicates a special incidence that the mesh STA rejects the incoming Mesh Peering Open frame requesting to set up a new mesh peering for some specified reason. The incoming request is identified by the peerMAC and peerLinkID as received from the Mesh Peering Open frame.

***Modify section 11C.4.4.5 as indicated***

* IDLE state

IDLE is a quiescent state the finite state machine enters prior to establishing a new mesh peering.

When ACTOPN event occurs, the mesh STA shall initiate the retryCounter to zero, and send a Mesh Peering Open frame to the candidate peer mesh STA whose address is peerMAC. The retryTimer shall be set according to retryTimeout. The finite state machine shall transition to OPN\_SNT state.

When an OPN\_ACPT event occurs, the mesh STA shall send the corresponding Mesh Peering Confirm frame to respond to the Mesh Peering Open frame. It shall also send a Mesh Peering Open frame to request a Mesh Peering Confirm frame from the candidate peer mesh STA. The retryTimer is set according to dot11MeshRetryTimeout value. The finite state machine shall transition to OPN\_RCVD state.

When an REQ\_RJCT event occurs, a Mesh Peering Close frame shall be sent to reject the mesh peering open request. The reason code in the Mesh Peering Close frame shall be set to the reason code in the REQ\_RJCT event. The finite state machine shall stay in the IDLE state.

***Modify section 11C.5.1 as indicated***

* Overview

The Authenticated Mesh Peering Exchange establishes an authenticated mesh peering between the mesh STAs, under the assumption that Mesh PMKSA has already been established before the initiation of the protocol via the active authentication protocol. An authenticated mesh peering includes a mesh peering, corresponding Mesh TKSA, and the two mesh STAs mesh GTKSAs.

The Authenticated Mesh Peering Exchange uses Mesh Peering Management frames. Parameters are exchanged via RSN element, Authenticated Mesh Peering Exchange element, and MIC element.

The major functions provided by AMPE are security sapabilities selection, key confirmation, and key management.

* The security capabilities selection function (specified in 11C.5.2 (Security capabilities selection)) is performed by agreeing on the security parameters used for the protocol instance.
* Key confirmation using the shared Mesh PMK is performed by verifying that the protection on the Mesh Peering Management frames is correct.
* Key management (specified in 8.8.1 (Keys and Key Derivation Algorithm)) is performed by the derivation of the temporal key in the Mesh TKSA and the exchange of each mesh STAs’ MGTK.

**References:**