12. Security

12.1 Conventions

Reserved fields and subfields are set to 0 upon transmission and are ignored upon reception.

ASCII strings are defined in 9.2.2.

12.2 Framework

12.2.1 Classes of security algorithm

This standard defines two classes of security algorithms for IEEE 802.11 networks:

— Algorithms for creating and using an RSNA, called *RSNA algorithms*
— Pre-RSNA algorithms

NOTE—This standard does not prohibit STAs from simultaneously operating pre-RSNA and RSNA algorithms.

The use of WEP for confidentiality, authentication, or access control is deprecated. The WEP algorithm is unsuitable for the purposes of this standard.

The use of TKIP is deprecated. The TKIP algorithm is unsuitable for the purposes of this standard.

12.2.2 Security methods

Pre-RSNA security comprises the following algorithms and procedures:

— WEP, described in 12.3.2
— IEEE 802.11 entity authentication, described in 12.3.3

RSNA security comprises the following algorithms and procedures:

— TKIP, described in 12.5.2
— CCMP, described in 12.5.3
— GCMP, described in 12.5.5
— BIP, described in 12.5.4
— RSNA establishment and termination procedures, including use of IEEE 802.1X authentication, described in 12.6 and SAE authentication described in 12.4
— Key management procedures, described in 12.7

12.2.3 RSNA STA capabilities

When dot11RSNAActivated is true, an RSNA STA shall include the RSNE in Beacon, Probe Response, Information Response, and (Re)Association Request frames and in message 2 and message 3 of the 4-way handshake; shall set the DMG Privacy subfield to 1 within transmitted DMG Beacons; and may include the RSNE in DMG Beacon and Announce frames. A pre-RSNA STA is not capable of creating RSNAs.

All STAs implementing RSNA shall support the RSNE described in 9.4.2.25.

12.2.4 RSNA establishment

An SME establishes an RSNA in one of six ways:

a) If an RSNA uses authentication negotiated over IEEE Std 802.1X in an infrastructure BSS, an RSNA-capable STA's SME establishes an RSNA as follows:
   1) It identifies the AP as RSNA-capable from the AP's Beacon, DMG Beacon, Announce, Information Response, or Probe Response frames.
   2) It shall invoke Open System authentication if the STA is a non-DMG STA.
   3) It negotiates cipher suites during the association process, as described in 12.6.2 and 12.6.3.
   4) It uses IEEE Std 802.1X-2010 to authenticate, as described in 12.6.10 and 12.6.11.
   5) It establishes one or more temporal keys by executing a key management algorithm, using the protocol defined by 12.7 or 13.
   6) It protects the data link by programming the negotiated cipher suites and the established temporal key into the MAC and then invoking protection. See, for example, 12.5.3 for a description of the RSNA data protection mechanisms.
   7) If the STAs negotiate management frame protection, the SME programs the TK and pairwise cipher suite into the MAC for protection of individually addressed robust Management frames. It also installs the IGTK and IPN for protection of group addressed robust Management frames.

b) If an RSNA is based on a PSK or password in an infrastructure BSS, the SME establishes an RSNA as follows:
   1) It identifies the AP as RSNA-capable from the AP's Beacon, DMG Beacon, Announce, Information Response, or Probe Response frames.
   2) If the RSNA-capable AP advertises support for SAE authentication in its Beacon or Probe Response frames, and the STA has a group defined in the dot11RSNAConfigDLCGroupTable and a password for the AP in the dot11RSNAConfigPasswordValueTable, the STA shall invoke SAE authentication to establish a PMK. If the RSNA-capable AP does not advertise support for SAE authentication in its Beacon and Probe Response frames but advertises support for the alternate form of PSK authentication (see 4.10.3.4), and the STA also supports the alternate form of PSK authentication, the non-DMG STA may invoke Open System authentication and use the PSK as the PMK with the key management algorithm in step 4) below.
   3) It negotiates cipher suites during the association process, as described in 12.6.2 and 12.6.3.
   4) It establishes one or more temporal keys by executing a key management algorithm, using the protocol defined by 12.7.
   5) It protects the data link by programming the negotiated cipher suites and the established temporal key into the MAC and then invoking protection.
   6) If the STAs negotiate management frame protection, the STA programs the TK and pairwise cipher suite into the MAC for protection of individually addressed robust Management frames. It also installs the IGTK and IPN for protection of group addressed robust Management frames.

c) If an RSNA is based on a PSK or password in an IBSS the SME executes the following sequence of procedures:
   1) It identifies the peer as RSNA-capable from the peer's Beacon, DMG Beacon, Announce, Information Response, or Probe Response frames.

NOTE—STAs might respond to a Data frame from an unrecognized STA by sending a Probe Request frame to find out whether the unrecognized STA is RSNA-capable.

   2) If the RSNA-capable peer advertises support for SAE authentication in its Beacon and Probe Response frames and the STA has a group defined in the dot11RSNAConfigDLCGroupTable and a password for the peer in the dot11RSNAConfigPasswordValueTable, the STA shall invoke SAE authentication and establish a PMK. If the RSNA-capable peer does not advertise support for SAE authentication but advertises support for the alternate form of PSK authentication (see 4.10.3.4), and the STA also supports the alternate form of PSK

authentication the STA may invoke Open System authentication if the STA is a non-DMG STA and use a PSK as the PMK with the alternate form of PSK authentication.

3) Each STA uses the procedures in 12.7, to establish one or more temporal keys and to negotiate cipher suites. Note that two peer STAs might follow this procedure simultaneously. See 12.6.15.

4) It protects the data link by programming the negotiated cipher suites and the established temporal key and then invoking protection.

d) If an RSNA uses authentication negotiated over IEEE Std 802.1X in an IBSS, an RSNA-capable STA's SME establishes an RSNA as follows:

1) It identifies the peer as RSNA-capable from the peer's Beacon, DMG Beacon, Announce, Information Response, or Probe Response frames.

NOTE—STAs might respond to a Data frame from an unrecognized STA by sending a Probe Request frame to find out whether the unrecognized STA is RSNA-capable.

2) It may invoke Open System authentication if the STA is a non-DMG STA.

3) Each STA uses IEEE Std 802.1X-2010 to authenticate with the AS associated with the other STA's Authenticator, as described in 12.6.10 and 12.6.11.

NOTE—Two peer STAs might follow this procedure simultaneously.

4) Each SME establishes one or more temporal keys by executing a key management algorithm, using the protocol defined in 12.7. Hence two such key management algorithms are happening in parallel between any two STA's Supplicants and Authenticators.

5) Both STAs use the agreed-upon temporal key portion of the PTK and pairwise cipher suite from one of the exchanges to protect the link. Each STA uses the GTK established by the exchange it initiated to protect the group addressed frames it transmits.

e) In order to associate with a PCP in a PBSS, an RSNA-capable STA's SME establishes an RSNA with the PCP following the RSNA establishment steps in an infrastructure BSS in accordance with method a) and b) above, as appropriate, with the PCP taking the role of the AP.

f) When an RSNA-capable STA chooses not to associate with a peer in a PBSS, its SME establishes an RSNA with the peer following the RSNA establishment steps in an IBSS in accordance with method c) or d) above, as appropriate, with the caveat that the RSNA authentication and key management algorithm is executed only once between the peers.

The time a security association takes to set up shall be less than dot11RSNAConfigSATimeout. The security association setup starts when initiated by the SME and completes when the MLME-SETPROTECTION.request primitive is invoked. The action the STA takes on the timeout is a policy decision. Some options include retrying the security association setup or trying another STA. This timeout allows recovery when one of the STAs setting up a security association fails to respond correctly to setting up the security association. It also allows recovery in IBSS when one of the two security associations fails because of a security association timeout.

Only Authentication frames with the authentication algorithm equal to Open System authentication or FT authentication may be used within an RSNA. An RSNA STA shall not associate if Shared Key authentication was invoked prior to RSN association.

12.2.5 RSNA PeerKey Support

The STSL mechanism is obsolete. Consequently, the PeerKey protocol components that do not support the AP PeerKey protocol might be removed in a later revision of the standard.

The PeerKey protocol provides mutual authentication, session identification, and data confidentiality for a station-to-station connection. A PeerKey association, composed of an SMKSA and an STKSA, shall be allowed only within the context of an existing RSNA by both peers with a common AP, or between peer APs that implement the AP PeerKey protocol. An initiator STA or peer STA shall not initiate an STSL master key (SMK) handshake and STSL transient key (STK) handshake if dot11RSNAActivated is false. An initiator STA or peer STA shall not establish a security association if dot11RSNAActivated is false.

A STSL session may chose to allow unprotected communication between STAs. In this case, the PeerKey protocol is not used.

12.2.6 RSNA assumptions and constraints

An RSNA assumes the following:

a) Each STA can generate cryptographic-quality random numbers. This assumption is fundamental, as cryptographic methods require a source of randomness. See J.5 for suggested hardware and software methods to achieve randomness suitable for this purpose.

b) When IEEE 802.1X authentication is used, the specific EAP method used performs mutual authentication. This assumption is intrinsic to the design of RSN in IEEE 802.11 LANs and cannot be removed without exposing both the STAs to man-in-the-middle attacks. EAP-MD5 is an example of an EAP method that does not meet this constraint (see IETF RFC 3748 [B42]). Furthermore, the use of EAP authentication methods where server and client credentials are not differentiated reduces the security of the method to that of a PSK due to the fact that malicious insiders might masquerade as servers and establish a man-in-the-middle attack.

   In particular, the mutual authentication requirement implies an unspecified prior enrollment process (e.g., a long-lived authentication key or establishment of trust through a third party such as a certification authority), as the STA needs to be able to identify the ESS or IBSS as a trustworthy entity and vice versa. The STA shares authentication credentials with the AS utilized by the selected AP or, in the case of PSK, the selected AP. The SSID provides an unprotected indication that the selected AP's authentication entity shares credentials with the STA. Only the successful completion of the IEEE 802.1X EAP or PSK authentication, after association, validates any such indication that the AP is connected to an authorized network or service provider.

c) The mutual authentication method needs to be strong, meaning impersonation attacks are computationally infeasible when based on the information exposed by the authentication. This assumption is intrinsic to the design of RSN.

d) The AP and AS have a trustworthy channel between them that can be used to exchange cryptographic keys without exposure to any intermediate parties.

e) An IEEE 802.1X AS never exposes the common symmetric key to any party except the AP with which the STA is currently communicating. This is a very strong constraint. It implies that the AS itself is never compromised. It also implies that the IEEE 802.1X AS is embedded in the AP or that the AP is physically secure and the AS and the AP lie entirely within the same administrative domain. This assumption follows from the fact that if the AP and the AS are not collocated or do not share pairwise key encryption keys directly, then it is impossible to assure the non-AP STA that its key, which is distributed by the AS to the AP, has not been compromised prior to use.

f) Similarly, a STA never shares with a third party a common symmetric key that it shares with a peer. Doing so destroys the utility of the key for detecting MPDU replay and forgery.

g) The Supplicant and the Authenticator generate a different, fresh PTK for each session between the pair. This assumption is fundamental, as reuse of any PTK would enable compromise of all the data protected by that key.

h) The destination STA chosen by the transmitter is the correct destination. For example, Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP) are methods of determining the destination STA's MAC address that are not secure from attacks by other members

of the ESS. One of the possible solutions to this problem might be for the STA to send or receive only frames whose final DA or SA are the AP and for the AP to provide a network layer routing function. However, such solutions are outside the scope of this standard.

An HT STA shall not use either of the pairwise cipher suite selectors: "Use group cipher suite" or TKIP to communicate with another HT STA.

NOTE—Because a VHT STA is also an HT STA, the elimination of TKIP also applies to VHT STAs.

12.2.7 Requirements for the Protected Frame field

The Protected Frame field shall be set to 1 in:

— Data frames that are protected using the mechanisms specified in Clause 12.
— Individually addressed robust Management frames.
— Authentication frames with Authentication Algorithm Number field equal to 1 (Shared Key) and Authentication Transaction Sequence Number field equal to 3.

It shall be set to 0 in all other frames.

12.2.8 Requirements for robust management frame protection

The robust Management frames are Disassociation, Deauthentication, and robust Action frames.

Action frames specified with "No" in the "Robust" column of Table 9-47 are not robust Management frames and shall not be protected.

When management frame protection is negotiated, all group addressed robust Management frames shall be encapsulated using the procedures defined in 11.13.

12.2.9 Emergency service establishment in an RSN

An AP or mesh STA that supports RSNAs and has the ESR bit set to 1 and the UESA bit set to 1 in the Interworking element in Beacon and Probe Response frames supports both RSNAs and emergency services associations (see 11.3.5.2) simultaneously.

In an infrastructure BSS, the STAs with emergency services association should discard all group addressed frames they receive, as they do not possess the Group Key and therefore are not able to decrypt group addressed frames. In an RSNA enabled BSS that has one or more STAs associated with an emergency services association, an AP should avoid transmitting unprotected group addressed frames in order not to disturb the operation of STAs that are in possession of Group Key. One possible way of achieving this is to support Proxy-ARP in the AP, as defined in 11.24.14. In addition, it is recommended that an AP supporting emergency services association should also support DMS to convert group addressed frames to individually addressed frames and transmit them to STAs associated using the emergency services association. STAs using emergency services association might request DMS if needed.

12.3 Pre-RSNA security methods

12.3.1 Status of Pre-RSNA security methods

Except for Open System authentication, all pre-RSNA security mechanisms are obsolete. Support for them might be removed in a later revision of the standard.

Open System Authentication and Open System Deauthentication shall not be used between mesh STAs.

12.3.2 Wired equivalent privacy (WEP)

12.3.2.1 WEP overview

WEP-40 was defined as a means of protecting (using a 40-bit key) the confidentiality of data exchanged among authorized users of a WLAN from casual eavesdropping. Implementation of WEP is optional. The same algorithms have been widely used with a 104-bit key instead of a 40-bit key in fielded implementations; this is called *WEP-104*. The WEP cryptographic encapsulation and decapsulation mechanics are the same whether a 40-bit or a 104-bit key is used. The term WEP by itself refers to either WEP-40 or WEP-104.

12.3.2.2 WEP MPDU format

Figure 12-1 depicts the encrypted frame body as constructed by the WEP algorithm.

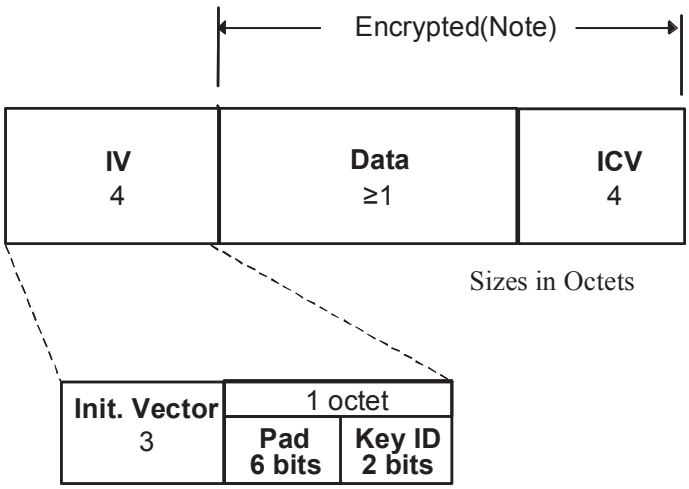

Figure 12-1—Construction of expanded WEP MPDU

The WEP ICV field shall be 32 bits in length. The expanded frame body shall start with a 32-bit IV field. This field shall contain three subfields: a 3-octet subfield that contains the IV, a 2-bit Key ID subfield, and a 6-bit Pad subfield. The ordering conventions defined in 9.2.2 apply to the IV field and its subfields and to the ICV field. The Key ID subfield contents select one of four possible secret key values for use in decrypting this frame body. When key-mapping keys are used, the Key ID field is reserved.

Interpretation of these bits is discussed further in 12.3.2.3. The contents of the Pad subfield shall be 0. The Key ID subfield occupies the 2 MSBs of the last octet of the IV field, while the Pad subfield occupies the 6 LSBs of this octet.

12.3.2.3 WEP state

WEP uses encryption keys only; it performs no data authentication. Therefore, it does not have data integrity keys. WEP uses two types of encryption keys: key-mapping keys and default keys.

A key-mapping key is an unnamed key corresponding to a distinct transmitter address-receiver address <TA,RA> pair. Implementations shall use the key-mapping key if it is configured for a <TA,RA> pair. In other words, the key-mapping key shall be used to WEP-encapsulate or -decapsulate MPDUs transmitted by TA to RA, regardless of the presence of other key types. When a key-mapping key for an address pair is present, the WEP Key ID subfield in the MPDU is reserved.

A default key is an item in a four-element MIB array called *dot11WEPDefaultKeys*, named by the value of a related array index called *dot11WEPDefaultKeyID*. If a key-mapping key is not configured for a WEP MPDU's <TA,RA> pair, WEP shall use a default key to encapsulate or decapsulate the MPDU. On transmit, the key selected is the element of the dot11DefaultKeys array given by the index dot11WEPDefaultKeyID—a value of 0, 1, 2, or 3—corresponding to the first, second, third, or fourth element, respectively, of dot11WEPDefaultKeys. The value the transmitter encodes in the WEP Key ID subfield of the transmitted MPDU shall be the dot11WEPDefaultKeyID value. The receiver shall use the Key ID subfield of the MPDU to index into dot11WEPDefaultKeys to obtain the correct default key. All WEP implementations shall support default keys.

NOTE—Many implementations also support 104-bit WEP keys. These are used exactly like 40-bit WEP keys: a 24-bit WEP IV is prepended to the 104-bit key to construct a 128-bit WEP seed, as explained in 12.3.2.4.3. The resulting 128-bit WEP seed is then consumed by the ARC4 stream cipher. This construction based on 104-bit keys affords no more assurance than the 40-bit construction, and its implementation and use are in no way condoned by this standard. Rather, the 104-bit construction is noted only to document de facto practice.

The default value for all WEP keys shall be null. WEP implementations shall discard the MSDU and generate an MA-UNITDATA-STATUS.indication primitive with transmission status indicating that a frame shall not be encapsulated with a null key in response to any request to encapsulate an MPDU with a null key.

12.3.2.4 WEP procedures

12.3.2.4.1 WEP ICV algorithm

The WEP ICV shall be computed using the CRC-32, as defined in 9.2.4.7, calculated over the plaintext MPDU Data (PDU) field.

12.3.2.4.2 WEP encryption algorithm

A WEP implementation shall use the ARC4 stream cipher from RSA Security, Inc., as its encryption and decryption algorithm. ARC4 uses a pseudorandom number generator (PRNG) to generate a key stream that it exclusive-ORs (XORs) with a plaintext data stream to produce cipher text or to recover plaintext from a cipher text.

12.3.2.4.3 WEP seed construction

A WEP implementation shall construct a per-MPDU key, called a *seed*, by concatenating an encryption key to an IV.

For WEP-40, bits 0–39 of the WEP key correspond to bits 24–63 of the seed, and bits 0–23 of the IV correspond to bits 0–23 of the seed, respectively. The bit numbering conventions in 9.2.2 apply to the seed. The seed shall be the input to ARC4, in order to encrypt or decrypt the WEP Data and ICV fields.

NOTE—For WEP-104, bits 0–103 of the WEP key correspond to bits 24–127 of the seed, and bit 0–23 of the IV correspond to bits 0–23 of the seed, respectively.

The WEP implementation encapsulating an MPDU's plaintext data should select a new IV for every MPDU it WEP-protects. The IV selection algorithm is unspecified. The algorithm used to select the encryption key used to construct the seed is also unspecified.

The WEP implementation decapsulating an MPDU shall use the IV from the received MPDU's Init Vector subfield. See 12.3.2.4.5 for the specification of how the decapsulator selects the key to use to construct the per-MPDU key.

12.3.2.4.4 WEP MPDU cryptographic encapsulation

WEP shall apply three transformations to the plaintext MPDU to effect the WEP cryptographic encapsulation. WEP computes the ICV over the plaintext data and appends this after the MPDU data. WEP encrypts the MPDU plaintext data and ICV using ARC4 with a seed constructed as specified in 12.3.2.4.3. WEP encodes the IV and key identifier into the IV field, prepended to the encrypted Data field.

Figure 12-2 depicts the WEP cryptographic encapsulation process. The ICV shall be computed and appended to the plaintext data prior to encryption, but the IV encoding step may occur in any order convenient for the implementation.

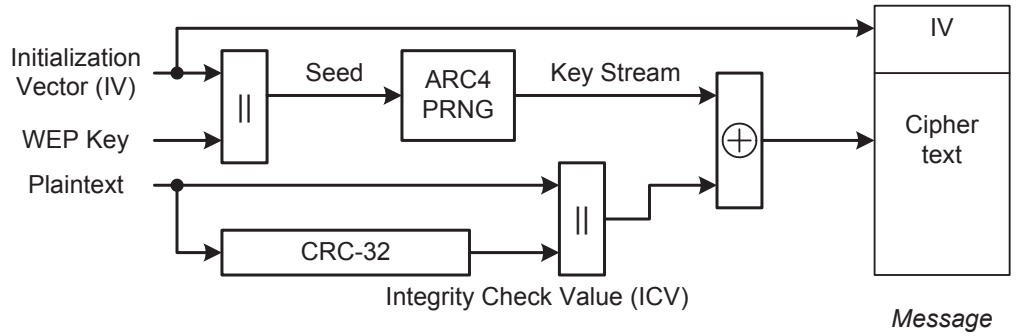

Figure 12-2—WEP encapsulation block diagram

12.3.2.4.5 WEP MPDU decapsulation

WEP shall apply three transformations to the WEP MPDU to decapsulate its payload. WEP extracts the IV and key identifier from the received MPDU. If a key-mapping key is present for the <TA,RA> pair, then this shall be used as the WEP key. Otherwise, the key identifier is extracted from the Key ID subfield of the WEP IV field in the received MPDU, identifying the default key to use.

WEP uses the constructed seed to decrypt the Data field of the WEP MPDU; this produces plaintext data and an ICV. Finally WEP recomputes the ICV and bitwise compares it with the decrypted ICV from the MPDU. If the two are bitwise identical, then WEP removes the IV and ICV from the MPDU, which is accepted as valid. If they differ in any bit position, WEP generates an error indication to MAC management. MSDUs with erroneous MPDUs (due to inability to decrypt) shall not be passed to LLC.

Figure 12-3 depicts a block diagram for WEP decapsulation. Unlike cryptographic encapsulation, the decapsulation steps shall be in the indicated order.

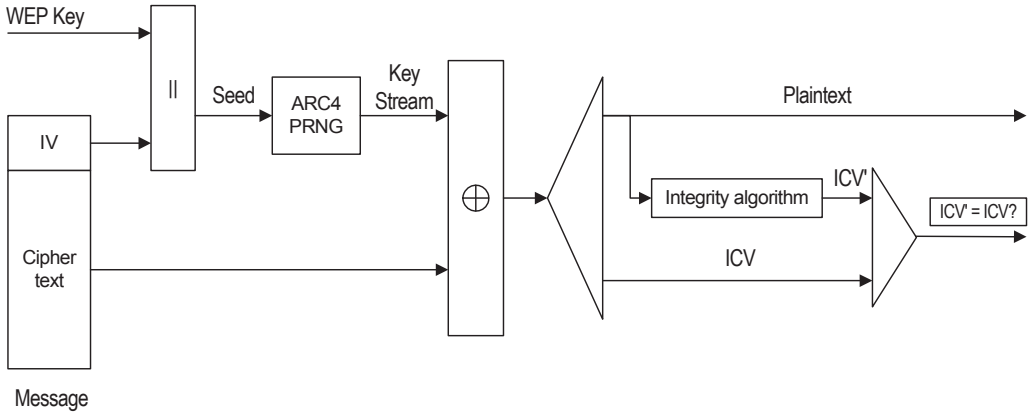

Figure 12-3—WEP decapsulation block diagram

12.3.3 Pre-RSNA authentication

12.3.3.1 Overview

In an infrastructure BSS, a non-DMG STA shall complete an IEEE 802.11 authentication exchange prior to association. A DMG STA not in an IBSS shall complete an IEEE 802.11 authentication exchange prior to association when an authentication algorithm other than the Open System authentication algorithm is requested. A DMG STA shall not perform an IEEE 802.11 authentication exchange using the Open System authentication algorithm. An IEEE 802.11 authentication exchange is optional in an IBSS.

All Authentication frames shall be individually addressed, as IEEE 802.11 authentication is performed between pairs of STAs, i.e., group addressed authentication is not allowed. Deauthentication frames are advisory and may be sent as group addressed frames.

Shared Key authentication is deprecated and should not be implemented except for backward compatibility with pre-RSNA STAs.

12.3.3.2 Open System authentication

12.3.3.2.1 General

Open System authentication is a null authentication algorithm.

Any non-DMG STA requesting Open System authentication can be authenticated if dot11AuthenticationAlgorithmsTable at the peer STA includes an entry with dot11AuthenticationAlgorithm equal to openSystem and dot11AuthenticationAlgorithmActivated equal to true.

A STA may decline to authenticate with another requesting STA. Open System authentication is the default authentication algorithm for a pre-RSNA STA.

Open System authentication utilizes a two-message authentication transaction sequence. The first message asserts identity and requests authentication. The second message returns the authentication result. If the result is “successful,” the STAs shall be declared mutually authenticated.

In the description in 12.3.3.2.2 and 12.3.3.2.3, the STA initiating the authentication exchange is referred to as the *requester*, and the STA to which the initial frame in the exchange is addressed is referred to as the *responder*. The specific items in each of the messages described in the following subclauses are defined in 9.3.3.12, Table 9-35, and Table 9-36.

12.3.3.2.2 Open System authentication (first frame)

Upon receipt of an Open System MLME-AUTHENTICATE.request primitive, the requester shall construct an Open System authentication request carried in an Authentication frame and transmit it to the responder.

12.3.3.2.3 Open System authentication (final frame)

Upon receipt of an Authentication frame requesting Open System authentication, the responder may authenticate the requester using the following procedure:

a) Issue an MLME-AUTHENTICATE.indication primitive to inform the SME of the authentication request.
b) Construct and transmit a response carried in an Authentication frame with the fields as defined in 9.3.3.12 and the status field as defined in 9.4.1.9.

If dot11AuthenticationAlgorithmTable does not include an entry with dot11AuthenticationAlgorithm equal to openSystem and dot11AuthenticationAlgorithmActivated equal to true, the result code shall not take the value "successful."

12.3.3.3 Shared Key authentication

12.3.3.3.1 General

Shared Key authentication seeks to authenticate STAs as either a member of those who know a shared secret key or a member of those who do not.

Shared Key authentication may be used if WEP has been selected and shall not be used otherwise.

This mechanism uses a shared key delivered to participating STAs via a secure channel that is independent of IEEE Std 802.11. This shared key is set in a write-only MIB attribute with the intent to keep the key value internal to the STA.

A STA shall not initiate a Shared Key authentication exchange unless dot11PrivacyOptionImplemented is true.

In the description in 12.3.3.3.2 to 12.3.3.3.6, the STA initiating the authentication exchange is referred to as the *requester*, and the STA to which the initial frame in the exchange is addressed is referred to as the *responder*. The specific items in each of the messages described in the following subclauses are defined in 9.3.3.12, Table 9-35, and Table 9-36.

12.3.3.3.2 Shared Key authentication (first frame)

Upon receipt of a Shared Key MLME-AUTHENTICATE.request primitive, the requester shall construct a Shared Key authentication request carried in an Authentication frame and transmit it to the responder.

12.3.3.3.3 Shared Key authentication (second frame)

Upon receipt of an Authentication frame requesting Shared Key authentication, the responder may authenticate the requester using the procedure here and in the following two frames:

a) Issue an MLME-AUTHENTICATE.indication primitive to inform the SME of the authentication request.
b) Before sending the second frame in the Shared Key authentication sequence, the responder shall use WEP to generate a string of octets to be used as the authentication challenge text.
c) Construct and transmit to the requester a response carried in an Authentication frame with the fields as defined in 9.3.3.12 and the status field as defined in 9.4.1.9.

If the status code is not SUCCESS, this shall be the last frame of the transaction sequence; and the content of the challenge text field is unspecified.

If the status code is SUCCESS, the following additional information items shall have valid contents:

— Authentication algorithm dependent information = The challenge text
— This authentication result shall be of fixed length of 128 octets. The field shall be filled with octets generated by the WEP PRNG. The actual value of the challenge field is unimportant, but the value shall not be a static value.

12.3.3.3.4 Shared Key authentication (third frame)

The requester shall copy the challenge text from the second frame into a third Authentication frame. The third frame shall be transmitted to the responder after cryptographic encapsulation by WEP, as defined in 12.3.2, using the shared key.

12.3.3.3.5 Shared Key authentication (final frame)

The responder shall WEP-decapsulate the third frame as described in 12.3.2. If the WEP ICV check is successful, the responder shall compare the decrypted contents of the Challenge Text field with the challenge text sent in second frame. If they are the same, then the responder shall transmit an Authentication frame to the requester with a successful status code in the final frame of the sequence. If the WEP ICV check fails or challenge text comparison fails, the responder shall respond with an unsuccessful status code in final frame.

12.3.3.3.6 Shared key MIB attributes

To transmit a Management frame of subtype Authentication, with an Authentication Transaction Sequence Number field value of 2, the MAC shall operate according to the following decision tree:

if dot11PrivacyOptionImplemented is false **then**
    the MMPDU is transmitted with a sequence of 0 octets in the Challenge Text field and a status code value of UNSUPPORTED_AUTH_ALGORITHM
else
    the MMPDU is transmitted with a sequence of 128 octets generated using the WEP PRNG and a key whose value is unspecified and beyond the scope of this standard and a randomly chosen IV value (note that this is typically selected by the same mechanism for choosing IV values for transmitted Data frames) in the Challenge Text field and a status code value of SUCCESS (the IV used is immaterial and is not transmitted). Note that there are cryptographic issues involved in the choice of key/IV for this process as the challenge text is sent unencrypted and, therefore, provides a known output sequence from the PRNG.
endif

To receive a Management frame of subtype Authentication, with an Authentication Transaction Sequence Number field value of 2, the MAC shall operate according to the following decision tree:

if the Protected Frame subfield of the Frame Control field is 1 **then**
    respond with a status code value of CHALLENGE_FAILURE
else
    if dot11PrivacyOptionImplemented is true **then**
        if there is a mapping in dot11WEPKeyMappings matching the MSDU's TA **then**
            if that key is null **then**
                respond with a frame whose Authentication Transaction Sequence Number field is 3 that contains the appropriate authentication algorithm number, a status code value of CHALLENGE_FAILURE, and no Challenge Text field, without encrypting the contents of the frame
            else
                respond with a frame whose Authentication Transaction Sequence Number field is 3 that contains the appropriate authentication algorithm number, a status code value of SUCCESS, and the identical Challenge Text field, encrypted using that key, and setting the Key ID subfield in the IV field to 0
            endif
        else
            if dot11WEPDefaultKeys[dot11WEPDefaultKeyID] is null **then**

respond with a frame whose Authentication Transaction Sequence Number field is 3 that contains the appropriate authentication algorithm number, a status code value of CHALLENGE_FAILURE, and no Challenge Text field, without encrypting the contents of the frame
else
respond with a frame whose Authentication Transaction Sequence Number field is 3 that contains the appropriate authentication algorithm number, a status code value of SUCCESS, and the identical Challenge Text field, WEP-encapsulating the frame under the key dot11WEPDefaultKeys[dot11WEPDefaultKeyID], and setting the Key ID subfield in the IV field to dot11WEPDefaultKeyID
endif
endif
else
respond with a frame whose Authentication Transaction Sequence Number field is 3 that contains the appropriate authentication algorithm number, a status code value of UNSUPPORTED_AUTH_ALGORITHM, and no Challenge Text field, without encrypting the contents of the frame
endif
endif

When receiving a Management frame of subtype Authentication, with an Authentication Transaction Sequence Number field value of 3, the MAC shall operate according to the following decision tree:

if the Protected Frame subfield of the Frame Control field is 0 **then**
respond with a status code value of CHALLENGE_FAILURE
else
if dot11PrivacyOptionImplemented is true **then**
if there is a mapping in dot11WEPKeyMappings matching the MSDU's TA **then**
if that key is null **then**
respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate authentication algorithm number and a status code value of CHALLENGE_FAILURE without encrypting the contents of the frame
else
WEP-decapsulate with that key, incrementing dot11WEPICVErrorCount and responding with a status code value of CHALLENGE_FAILURE if the ICV check fails
endif
else
if dot11WEPDefaultKeys[dot11WEPDefaultKeyID] is null **then**
respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate authentication algorithm number and a status code value of CHALLENGE_FAILURE without encrypting the contents of the frame
else
WEP-decapsulate with dot11WEPDefaultKeys[dot11WEPDefaultKeyID], incrementing dot11WEPICVErrorCount and responding with a status code value of CHALLENGE_FAILURE if the ICV check fails
endif
endif
else
respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate authentication algorithm number and a status code value of CHALLENGE_FAILURE

endif
endif

dot11PrivacyInvoked shall not take the value of true if dot11PrivacyOptionImplemented is false. Setting dot11WEPKeyMappings to a value that includes more than dot11WEPKeyMappingLengthImplemented entries is illegal and shall have an implementation-specific effect on the operation of the data confidentiality service. Note that dot11WEPKeyMappings may contain from zero to dot11WEPKeyMappingLengthImplemented entries, inclusive.

The values of the attributes in the aPrivacygrp should not be changed during the authentication sequence, as unintended operation may result.

12.4 Authentication using a password

12.4.1 SAE overview

STAs, both AP STAs and non-AP STAs, may authenticate each other by proving possession of a password. Authentication protocols that employ passwords need to be resistant to off-line dictionary attacks.

Simultaneous authentication of equals (SAE) is a variant of *Dragonfly*, a password-authenticated key exchange based on a zero-knowledge proof. SAE is used by STAs to authenticate with a password; it has the following security properties:

— The successful termination of the protocol results in a PMK shared between the two STAs.
— An attacker is unable to determine either the password or the resulting PMK by passively observing an exchange or by interposing itself into the exchange by faithfully relaying messages between the two STAs.
— An attacker is unable to determine either the password or the resulting shared key by modifying, forging, or replaying frames to an honest, uncorrupted STA.
— An attacker is unable to make more than one guess at the password per attack. This implies that the attacker cannot make one attack and then go offline and make repeated guesses at the password until successful. In other words, SAE is resistant to dictionary attack.
— Compromise of a PMK from a previous run of the protocol does not provide any advantage to an adversary attempting to determine the password or the shared key from any other instance.
— Compromise of the password does not provide any advantage to an adversary in attempting to determine the PMK from the previous instance.

Unlike other authentication protocols SAE does not have a notion of an “Initiator” and “Responder” or of a “Supplicant” and “Authenticator.” The parties to the exchange are equals, with each side being able to initiate the protocol. Each side may initiate the protocol simultaneously such that each side views itself as the “initiator” for a particular run of the protocol. Such a peer-to-peer protocol may be used in a traditional client-server (or Supplicant/Authenticator) fashion but the converse does not hold. This requirement is necessary to address the unique nature of MBSSs.

The parties involved are called *STA-A* and *STA-B*. They are identified by their MAC addresses, STA-A-MAC and STA-B-MAC, respectively. STAs begin the protocol when they discover a peer by receiving Beacon or Probe Response frame(s), or when they receive an Authentication frame indicating SAE authentication from a peer.

SAE is an RSNA authentication protocol and is selected according to 12.6.2.

SAE shall be implemented on all mesh STAs to facilitate and promote interoperability.

12.4.2 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, pseudorandom key. This function takes as input a non-secret "salt" and a secret input keying material "ikm" and produces a fixed-length output.

— CN is a confirmation function that 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 00-0F-AC:8 or 00-0F-AC:9 from Table 9-133, H is instantiated as HMAC-SHA-256:

H(salt, ikm) = HMAC-SHA-256(salt, ikm)

When used with AKMs 00-0F-AC:8 or 00-0F-AC:9 from Table 9-133, CN is instantiated as a function that takes a key, a counter, and a sequence of data. Each piece of data is converted to an octet string and concatenated together before being concatenated to the counter and passed, along with the key, to HMAC-SHA-256:

CN(key, counter, X, Y, Z, …) = HMAC-SHA-256(key, counter || D2OS(X) || D2OS(Y) || D2OS(Z) || …)

where D2OS() represents the data to octet string conversion functions in 12.4.7.2. Each invocation of CN() specifies the format of the counter.

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

12.4.3 Representation of a password

Passwords are used in SAE to deterministically compute a secret element in the negotiated group, called a *password element*. The input to this process needs to be in the form of a binary string. For the protocol to successfully terminate, it is necessary for each side to produce identical binary strings for a given password, even if that password is in character format. There is no canonical binary representation of a character and ambiguity exists when the password is a character string. To eliminate this ambiguity, a STA shall represent a character-based password as an ASCII string. Representation of a character-based password in another character set or use of a password preprocessing technique (to map a character string to a binary string) may be agreed upon, in an out-of-band fashion, prior to beginning SAE. If the password is already in binary form (e.g., it is a binary preshared key) no character set representation is assumed. The binary representation of the password, after being transformed from a character representation or directly if it is already in binary form, is stored in the dot11RSNConfigPasswordValueTable. When a "password" is called for in the description of SAE that follows the credential from the dot11RSNConfigPasswordValueTable is used.

12.4.4 Finite cyclic groups

12.4.4.1 General

SAE uses discrete logarithm cryptography to achieve authentication and key agreement. Each party to the exchange derives ephemeral public and private keys with respect to a particular set of domain parameters that define a finite cyclic group. Groups may be based on either Finite Field Cryptography (FFC) or on Elliptic Curve Cryptography (ECC). Each component of a group is referred to as an *element*. Groups are negotiated using an identifying number from a repository maintained by IANA as "Group Description"

attributes for IETF RFC 2409 (IKE) [B17][B31]. The repository maps an identifying number to a complete set of domain parameters for the particular group. For the purpose of interoperability, a STA shall support group 19, an ECC group defined over a 256-bit prime order field.

More than one group may be configured on a STA for use with SAE by using the dot11RSNAConfigDLCGroup table. Configured groups are prioritized in ascending order of preference. If only one group is configured, it is, by definition, the most preferred group.

NOTE—The preference of one group over another is a local policy issue.

SAE uses three arithmetic operators defined for both FFC and ECC groups, an operation that takes two elements to produce a third element (called the *element operation*), an operation that takes an integer (called *scalar*) and an element to produce a second element (called the *scalar operation*), and an operation that takes an element to produce a second element (called the *inverse operation*). The convention used here is to represent group elements in uppercase bold italic and scalar values in lowercase italic. The element operation takes two elements, $\boldsymbol{X}$ and $\boldsymbol{Y}$, to produce a third element, $\boldsymbol{Z}$, and is denoted $\boldsymbol{Z}$ = elem-op($\boldsymbol{X}$,$\boldsymbol{Y}$); the scalar operation takes a scalar, x, and an element, $\boldsymbol{Y}$, to produce a second element $\boldsymbol{Z}$ and is denoted $\boldsymbol{Z}$ = scalar-op(x,$\boldsymbol{Y}$); the inverse operation takes an element, $\boldsymbol{X}$, to produce a second element, $\boldsymbol{Z}$, and is denoted $\boldsymbol{Z}$ = inverse-op($\boldsymbol{X}$).

scalar-op(x,$\boldsymbol{Y}$) is defined as successive iterations of elem-op($\boldsymbol{Y}$, $\boldsymbol{Y}$). That is, it is possible to define scalar-op(1, $\boldsymbol{Y}$) = $\boldsymbol{Y}$ and for $x > 1$, scalar-op(x, $\boldsymbol{Y}$) = elem-op(scalar-op(x-1, $\boldsymbol{Y}$), $\boldsymbol{Y}$). The specific definition of elem-op($\boldsymbol{X}$, $\boldsymbol{Y}$) depends on the type of group, either ECC or FFC.

12.4.4.2 Elliptic curve cryptography (ECC) groups

12.4.4.2.1 ECC group definition

ECC groups used by SAE are defined by the sextuple (p, a, b, $\boldsymbol{G}$, r, h) where p is a prime number, a and b specify the elliptic curve defined by the equation, $y^2 = x^3 + ax + b \bmod p$, $\boldsymbol{G}$ is a generator (a base point on the elliptic curve), r is the prime order of $\boldsymbol{G}$, and h is the co-factor. Elements in ECC groups are the points on the elliptic curve defined by their coordinates—(x, y)—that satisfy the equation for the curve and the identity element, the so-called "point at infinity."

The IANA registry used to map negotiated numbers to group domain parameters includes some ECC groups defined over a characteristic 2 finite field and may include some ECC groups with a co-factor greater than 1. These groups shall not be used with SAE. Only ECC groups defined over an odd prime finite field with a co-factor equal to 1 shall be used with SAE.

The element operation in an ECC group is addition of two points on the curve resulting in a third point on the curve. For example, the point $\boldsymbol{X}$ is added to the point $\boldsymbol{Y}$ to produce the point $\boldsymbol{Z}$:

$$\boldsymbol{Z} = \boldsymbol{X} + \boldsymbol{Y} = \text{elem-op}(\boldsymbol{X},\boldsymbol{Y})$$

The scalar operation in an ECC group is multiplication of a point on the curve by a scalar resulting in a second point on the curve. For example, the point $\boldsymbol{Y}$ is multiplied by the scalar x to produce the point $\boldsymbol{Z}$:

$$\boldsymbol{Z} = x\boldsymbol{Y} = \text{scalar-op}(x,\boldsymbol{Y})$$

The inverse operation in an ECC group is inversion of a point on a curve resulting in a second point on the curve. A point on an elliptic curve is the inverse of a different point if their sum is the "point at infinity." In other words:

$$\text{elem-op}(\boldsymbol{X}, \text{inverse}(\boldsymbol{X})) = \text{"point at infinity"}$$

ECC groups make use of a mapping function, F, that maps a point (x, y) that satisfies the curve equation to its x-coordinate—i.e., if $\boldsymbol{P} = (x, y)$ then $F(\boldsymbol{P}) = x$. Function F is not defined with the identity element as input.

NOTE—SAE protocol operations preclude function F from ever being called with the identity element, i.e., the "point at infinity."

12.4.4.2.2 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. The password and a counter, represented as a single octet and initially set to 1, are used with the peer identities to generate a password seed. The password seed shall then be stretched using the key derivation function (KDF) from 12.7.1.7.2 to a length equal to the bit length of the prime number, p, from the elliptic curve domain parameters with the Label being the string "SAE Hunting and Pecking" and with the Context being the prime number. If the resulting password value is greater than or equal to the prime number, the counter shall be incremented, a new password seed shall be derived and the hunting-and-pecking shall continue. Otherwise, it shall be used as the x-coordinate of a candidate point (x, y) on the curve satisfying the curve equation, if such a point exists. If no solution exists, the counter shall be incremented, a new password-seed shall be derived and the hunting-and-pecking shall continue. Otherwise, there are two possible solutions: (x, y) and (x, $p - y$). The password seed shall be used to determine which one to use: if the least-significant bit (LSB) of the password seed is equal to that of y, the ***PWE*** shall be set to (x, y); otherwise, it shall be set to (x, $p - y$).

In order to minimize the possibility of side-channel attacks that attempt to determine the number of interactions of the "hunting-and-pecking" loop required for a given <password, STA-A-MAC, STA-B-MAC> tuple, implementations should perform at least k iterations regardless of whether ***PWE*** is discovered or not. The value k may be set to any non-negative value and should be set to a sufficiently large number to effectively guarantee the discovery of ***PWE*** in less than k iterations. If ***PWE*** is discovered in less than k iterations a random "password" can be used in subsequent iterations to further obfuscate the true cost of discovering ***PWE***.

NOTE—The probability that one requires more than n iterations of the "hunting and pecking" loop to find ***PWE*** is roughly $(r/2p)^n$, which rapidly approaches 0 as n increases.

Algorithmically this process is described as follows:

```
found = 0;
counter = 1
Length = len(p)
base = password
do {
        pwd-seed = H(MAX(STA-A-MAC, STA-B-MAC) || MIN(STA-A-MAC, STA-B-MAC),
                base || counter)
        pwd-value = KDF-Hash-Length(pwd-seed, "SAE Hunting and Pecking", p)
        if (pwd-value < p)
        then
                if (pwd-value^3 + a × pwd-value + b) is a quadratic residue modulo p
                then
                        if (found==0)
                        then
                                x = pwd-value
                                save = pwd-seed
                                found = 1
                                base = a new random number
                        fi
                fi
        fi
```

```
        counter = counter + 1
} while ((counter <= k) or (found==0))
y = sqrt(x^3 + ax + b) mod p
if (LSB(save) == LSB(y))
then
        PWE = (x, y)
else
        PWE = (x, p – y)
fi
```

where

KDF-Hash-*Length*	is the key derivation function defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133)
len()	returns the length of its argument in bits

Checking whether a value is a quadratic residue modulo a prime can leak information that can be used in launching a side-channel attack. Therefore, a STA should use this blinding technique in determining a quadratic residue to address the possibility of a side-channel attack.

The blinding technique involves multiplication of the value with a random number so the value being checked for quadratic residue modulo a prime can take on all numbers between 1 and p–1 with equal probability. The blinded value is multiplied by a quadratic residue or quadratic non-residue depending on the value of a coin flip and the result is checked whether the result is a quadratic residue or quadratic non-residue, respectively.

This technique involves creation of a quadratic residue, *qr*, and quadratic non-residue, *qnr*, prior to beginning of the hunting-and-pecking loop. These values can be chosen at random by checking their legendre symbol:

```
do {
        qr = random() mod p
} while ( LGR(qr | p) is not equal to –1)

do {
        qnr = random() mod p
} while ( LGR(qnr | p) is not equal to 1)
```

The blinding technique of determining whether a value, v, is a quadratic residue modulo a prime, p, is then:

```
r = (random() mod (p – 1)) + 1
num = (v × r × r) mod p
if (LSB(r) == 1)
then
        num = (num × qr) mod p
        if (LGR(num | p) == 1)
        then
                v is a quadratic residue modulo p
        fi
else
        num = (num × qnr) mod p
        if (LGR(num | p) == –1)
        then
                v is a quadratic residue modulo p
```

fi
fi
v is a quadratic non-residue modulo *p*

The values *qr* and *qnr* may be used for all loops in the hunting-and-pecking process but a new value for *r* shall be generated each time a quadratic residue is checked.

12.4.4.3 Finite field cryptography (FFC) groups

12.4.4.3.1 FFC group definition

FFC groups used by SAE are defined by the triple (p, $\boldsymbol{G}$, r), where p is a prime number, $\boldsymbol{G}$ is a generator, and r is the prime order of $\boldsymbol{G}$ mod p. An element, $\boldsymbol{B}$, in an FFC group satisfies $\boldsymbol{B} = \boldsymbol{G}^i \bmod p$ for some integer i. This special property differentiates elements from scalars, even though both elements and scalars can be represented as non-negative integers less than the prime modulus p. The notation convention of 12.4.4 signifies this difference between an element and a scalar in an FFC group. The identity element for an FFC group is the value 1 mod p.

The element operation in an FFC group is modular multiplication of two elements of this group resulting in a third element of this group. For example, the element $\boldsymbol{X}$ is multiplied by the element $\boldsymbol{Y}$ to product the element $\boldsymbol{Z}$:

$$\boldsymbol{Z} = (\boldsymbol{XY}) \bmod p = \text{elem-op}(\boldsymbol{X},\boldsymbol{Y})$$

The scalar operation in an FFC group is modular exponentiation of an element of this group by a scalar resulting in a second element of this group. For example, the point $\boldsymbol{Y}$ is raised to the power x to produce the element $\boldsymbol{Z}$:

$$\boldsymbol{Z} = \boldsymbol{Y}^x \bmod p = \text{scalar-op}(x,\boldsymbol{Y})$$

Some FFC groups in the IANA repository are based on *safe primes*, i.e., a prime, p, of the form $p = 2q + 1$, where q is also a prime number. For these FFC groups, the group generated by $\boldsymbol{G}$ always has order $r = (p - 1)/2$ and thus is uniquely derived from context. For other FFC groups, the parameter r shall be explicitly stated as part of the domain parameters.

The inverse operation in an FFC group is modular inversion of an element of this group producing a second element in this group. An element $\boldsymbol{Z}$ is the inverse of a second element $\boldsymbol{X}$ of this group if their modular product is the identity element of the FFC group. In other words:

$$\text{elem-op}(\boldsymbol{X}, \text{inverse}(\boldsymbol{X})) = 1 \bmod p$$

In contrast to ECC groups, FFC groups do not need a mapping function that maps an element of the FFC group to an integer (since those elements are already non-negative integers less than the prime number, p). However, for sake of uniform protocol definition, function F with FFC groups is defined as the identity function—i.e., if x is an element of the FFC group then $F(x) = x$.

12.4.4.3.2 Generation of the password element with FFC groups

The password element of an FFC group (***PWE***) shall be generated in a random hunt-and-peck fashion similar to the technique for an ECC group. The password and a counter, represented as a single octet and initially set to 1, are used with the two peer identities to generate a password seed. The password seed shall then be stretched using the key derivation function (KDF) from 12.7.1.7.2 to a length equal to the bit length

of the prime number, *p*, from the group domain parameters with the Label being the string "SAE Hunting and Pecking" and the Content being the prime number. If the resulting password value is greater than or equal to the prime number, the counter shall be incremented, a new password seed shall be derived, and the hunting-and-pecking shall continue. Otherwise, it shall be raised to the power $(p - 1) / r$ (where *p* is the prime number and *r* is the order) modulo the prime number to produce a candidate ***PWE***. If the candidate ***PWE*** is greater than 1, the candidate ***PWE*** becomes the ***PWE***; otherwise, the counter shall be incremented, a new password seed shall be derived, and the hunting-and-pecking shall continue.

Algorithmically this process is described as follows:

```
found = 0;
counter = 1
Length = len(p)
do {
        pwd-seed = H(MAX(STA-A-MAC, STA-B-MAC) || MIN(STA-A-MAC, STA-B-MAC),
                password || counter)
        pwd-value = KDF-Hash-Length(pwd-seed, "SAE Hunting and Pecking", p)
        if (pwd-value < p)
        then
                PWE = pwd-value^((p-1)/r) mod p
                if (PWE > 1)
                then
                        found = 1
                fi
        fi
        counter = counter + 1
} while (found==0)
```

where

KDF-Hash-*Length*	is the key derivation function defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133)
len()	returns the length of its argument in bits

12.4.5 SAE protocol

12.4.5.1 Message exchanges

The protocol consists of two message exchanges, a commitment exchange and a confirmation exchange. The commitment exchange is used to force each party to the exchange to commit to a single guess of the password. The confirmation exchange is used to prove that the password guess was correct. Authentication frames are used to perform these exchanges (see 9.3.3.12 and 12.4.7.3). The rules for performing these exchanges are specified by the finite state machine in 12.4.8.

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 are ascribed to the protocol:

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

12.4.5.2 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, either the most preferred group if the STA is initiating SAE to a peer, or the group from a received SAE Commit message if the STA is responding to a peer. The ***PWE*** shall be generated for that group (according to 12.4.4.2.2 or 12.4.4.3.2, 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 $1 < rand < r$ and $1 < mask < r$ and $(rand + mask)$ mod r is greater than 1, where r is the (prime) order of the group. If their sum modulo r is not greater than 1, they shall both be irretrievably deleted and new values shall be randomly generated. The values *rand* and *mask* shall be random numbers produced from a quality random number drawn from a uniform distribution generator. These values shall never be reused on distinct protocol runs.

12.4.5.3 Construction of an SAE Commit message

An SAE Commit message consists of a scalar and an element that shall be produced using the ***PWE*** and secrets generated in 12.4.5.2, as follows:

commit-scalar = (*rand* + *mask*) mod *r*
COMMIT-ELEMENT = inverse(scalar-op(*mask*, ***PWE***))

This message shall be transmitted to the peer as described in 12.4.7. The temporary secret *mask* may be deleted at this point.

12.4.5.4 Processing of a peer's SAE Commit message

Upon receipt of a peer's SAE Commit message both the scalar and element shall be verified.

If the scalar value is greater than 0 and less than the order, r, of the negotiated group, scalar validation succeeds; otherwise, it fails. Element validation depends on the type of group. For FFC groups, the element shall be an integer greater than 1 and less than the prime number p minus 1, $(p - 1)$, and the scalar operation of the element and the order of the group, r, shall equal 1 modulo the prime number p. If either of these conditions does not hold, element validation fails; otherwise, it succeeds. For ECC groups, both the x- and y-coordinates of the element shall be non-negative integers less than the prime number p, and the two coordinates shall produce a valid point on the curve satisfying the group's curve definition, not being equal to the "point at the infinity." If either of those conditions does not hold, element validation fails; otherwise, element validation succeeds.

If either scalar validation or element validation fails, the STA shall reject the peer's authentication. If both the scalar and element from the peer's SAE Commit message are successfully validated, a shared secret element, K, shall be derived using the scalar and element (*peer-commit-scalar* and ***PEER-COMMIT-ELEMENT***, respectively) from the peer's SAE Commit message and the STA's secret value.

K = scalar-op(*rand*, (elem-op(scalar-op(*peer-commit-scalar*, ***PWE***),
PEER-COMMIT-ELEMENT)))

If the shared secret element, ***K***, is the identity element for the negotiated group (the value one for an FFC group or the point-at-infinity for an ECC group) the STA shall reject the peer's authentication. Otherwise, a secret value, k, shall be computed as:

k = F(***K***)

The entropy of *k* shall then be extracted using H to produce *keyseed*. The key derivation function from 12.7.1.7.2 shall then be used with the hash algorithm identified by the AKM suite selector (see Table 9-133) to derive a key confirmation key, KCK, and a pairwise master key, PMK, from *keyseed*. When used with AKMs 8 or 9, the salt shall consist of thirty-two (32) octets of the value 0 (indicated below as <0>32) and both the KCK and PMK shall be 256 bits in length, and therefore the length of keying material derived is 512. Use of other AKMs require definition of the lengths of the salt, the KCK, and the PMK.

keyseed = H(<0>32, *k*)
kck_and_pmk = KDF-Hash-512(*keyseed*, "SAE KCK and PMK",
(*commit-scalar* + *peer-commit-scalar*) mod *r*)
KCK = L(*kck_and_pmk*, 0, 256)
PMK = L(*kck_and_pmk*, 256, 256)

where

KDF-Hash-512 is the key derivation function defined in 12.7.1.7.2 using the hash algorithm defined by the negotiated AKM in Table 9-133.

The PMK identifier is defined as follows:

PMKID = L((*commit-scalar* + *peer-commit-scalar*) mod *r*, 0, 128)

12.4.5.5 Construction of an SAE Confirm message

A peer generates an SAE Confirm message by passing the KCK, the current value of the *send-confirm* counter (see 9.4.1.38), the scalar and element from the sent SAE Commit message, and the scalar and element from the received SAE Commit message to the confirmation function CN.

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

The *send-confirm* counter shall be in the format specified in subclause 9.2.2 in the order in which it is transmitted over the air. The message shall be transmitted to the peer as described in 12.4.7.

12.4.5.6 Processing of a peer's SAE Confirm message

Upon receipt of a peer's SAE Confirm message a *verifier* is computed, which is the expected value of the peer's confirmation, *peer-confirm*, extracted from the received an SAE Confirm message. The *verifier* is computed by passing the KCK, the peer's send-confirm counter from the received an SAE Confirm message (see 9.4.1.38), the scalar and element from the received SAE Commit message, and scalar and element from the sent SAE Commit message to the confirmation function CN.

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

The p*eer-send-confirm* shall be in the format in subclause 9.2.2, as extracted out of the received frame. If the *verifier* equals *peer-confirm,* the STA shall accept the peer's authentication and set the lifetime of the PMK to the value dot11RSNAConfigPMKLifetime. If the *verifier* differs from the *peer-confirm,* the STA shall reject the peer's authentication and delete the PMK.

12.4.6 Anti-clogging tokens

A STA is required to do a considerable amount of work upon receipt of an SAE Commit message. This opens up the possibility of a distributed denial-of-service attack by flooding a STA with bogus SAE Commit

messages from forged MAC addresses. To prevent this from happening, a STA shall maintain an *Open* counter in its SAE state machine indicating the number of open and unfinished protocol instances (see 12.4.5.1). When that counter hits or exceeds dot11RSNASAEAntiCloggingThreshold, the STA shall respond to each SAE Commit message with a rejection that includes an Anti-Clogging Token statelessly bound to the sender of the SAE Commit message. The sender of the SAE Commit message shall then include this Anti-Clogging Token in a subsequent SAE Commit message.

The Anti-Clogging Token is a variable-length value that statelessly binds the MAC address of the sender of an SAE Commit message. The length of the Anti-Clogging Token needs not be specified because the generation and processing of the Anti-Clogging Token is solely up to one peer. To the other peer in the SAE protocol, the Anti-Clogging Token is merely an opaque blob whose length is insignificant. It is suggested that an Anti-Clogging Token not exceed 256 octets.

NOTE—A suggested method for producing Anti-Clogging Tokens is to generate a random secret value each time the state machine variable hits dot11RSNASAEAntiCloggingThreshold and pass that secret and the MAC address of the sender of the SAE Commit message to the random function H to generate the token.

As long as the state machine variable *Open* is greater than or equal to dot11RSNASAEAntiCloggingThreshold all SAE Commit messages that do not include a valid Anti-Clogging Token shall be rejected with a request to repeat the SAE Commit message and include the token (see 12.4.5.1).

Since the Anti-Clogging Token is of fixed size and the size of the *peer-commit-scalar* and ***PEER-COMMIT-ELEMENT*** are inferred from the finite cyclic group being used, it is straightforward to determine whether a received SAE Commit message includes an Anti-Clogging Token or not.

Encoding of the Anti-Clogging Token and its placement with respect to the *peer-commit-scalar* and ***PEER-COMMIT-ELEMENT*** is described in 12.4.7.4.

12.4.7 Framing of SAE

12.4.7.1 General

SAE Commit messages and SAE Confirm messages are sent and received by a SAE protocol using Authentication frames.

12.4.7.2 Data type conversion

12.4.7.2.1 General

This protocol requires elements in finite cyclic groups to be converted to octet strings prior to transmission and back again upon receipt. To convert an element into an octet string, the first step is to represent the element in integer format and then employ an integer-to-octet string conversion prior to transmission. To convert an octet string into an element requires an octet string to integer conversion and then representing the integer(s) as an element.

12.4.7.2.2 Integer to octet string conversion

An integer, x, shall be converted into an octet string of length m such that $2^{8m} > x$ by first representing x in its binary form and then converting the result to an octet-string.

Given x, m, represent x as a sequence of x_{m-i} base 2^8:

$$x = x_{m-1} \times 2^{8(m-1)} + x_{m-2} \times 2^{8(m-2)} + \ldots + x_1 \times 2^8 + x_0$$

then let the octet M_i have the value x_i for $0 \le i \le m-1$, and the octet string shall be $M_{m-1} \| M_{m-2} \| ... \| M_1 \| M_0$.

12.4.7.2.3 Octet string to integer conversion

An octet string shall be converted into an integer by viewing the octet string as the base 2^8 representation of the integer.

$$x = \sum_{i=1}^{m} 2^{8(m-i)} \times M_{m-i}$$

12.4.7.2.4 Element to octet string conversion

For ECC groups, each element, except the "point at infinity," is a point on the elliptic curve satisfying the curve equation and consists of two components: an x-coordinate and a y-coordinate. To convert this point to an octet string, each component shall be treated as an integer and converted into an octet string whose length is the smallest integer m such that $2^{8m} > p$, where p is the prime number specified by the elliptic curve domain parameters, according to 12.4.7.2.2. The point shall be represented as the concatenation of the x-coordinate and the y-coordinate, each represented as an octet string of length m octets, and is $2m$ octets long.

For FFC groups each element is a non-negative integer less than the prime number p specified by the FFC domain parameters. To convert this element into an octet string, it shall be treated directly as an integer and converted into an octet string whose length is the smallest integer m such that $2^{8m} > p$, where p is the prime number specified by the domain parameters, according to 12.4.7.2.2.

12.4.7.2.5 Octet string to element conversion

To convert an octet string into a point on an elliptic curve it is necessary to divide it into two octet strings of equal length m. If the length of the octet string does not evenly divide by two, conversion shall fail. Each octet string of length m shall be converted to an integer according to 12.4.7.2.3. The first octet string conversion produces an integer that becomes the x-coordinate of the point and the second octet string conversion produces an integer that becomes the y-coordinate of the point. If either integer equals 0 or is greater than or equal to p, the prime from the elliptic curve domain parameters, conversion shall fail. If the resulting (x, y) point does not satisfy the equation of the curve, or produces the "point at infinity," conversion shall fail.

To convert an octet string into an element in a prime modulus group the octet string shall be converted into an integer according to 12.4.7.2.3 and the integer shall be used directly as the group element.

12.4.7.3 Authentication transaction sequence number for SAE

An SAE Commit message shall use authentication transaction sequence number 1. an SAE Confirm message shall use authentication transaction sequence number 2.

12.4.7.4 Encoding and decoding of SAE Commit messages

An SAE Commit message shall be encoded as an Authentication frame with an Authentication Algorithm Number field set to 3, a Transaction Sequence Number of 1 and a Status Code of SUCCESS Status codes not equal to SUCCESS indicate a rejection of a peer's SAE Commit message and are described in 12.4.7.6.

An SAE Commit message shall consist of a Finite Cyclic Group field (9.4.1.43) indicating a group, a Scalar field (9.4.1.40) containing the scalar, and an FFE field containing the element (9.4.1.41). If the SAE Commit

message is in response to an Anti-Clogging Token request (see 12.4.7.6), the Anti-Clogging Token is present (see 9.4.1.39).

When transmitting an SAE Commit message, the scalar and element shall be converted to octet strings and placed in the Scalar field and FFE field, respectively. The scalar shall be treated as an integer and converted into an octet string of length m such that $2^{8m} > r$, where r is the order of the group, according to 12.4.7.2.2, and the element shall be converted into (an) octet string(s) according to 12.4.7.2.4. When receiving an SAE Commit message the component octet strings in the Scalar field and Element field shall be converted into a scalar and element, respectively, according to 12.4.7.2.3 and 12.4.7.2.5, respectively.

12.4.7.5 Encoding and decoding of SAE Confirm messages

An SAE Confirm message shall be encoded as an Authentication frame with an Authentication Algorithm Number field set to 3, a Transaction Sequence Number of 2 and a Status Code of SUCCESS. Status codes not equal to SUCCESS indicate rejection of a peer's SAE Confirm message and are described in 12.4.7.6.

An SAE Confirm message shall consist of a Send-Confirm field (9.4.1.38) and a Confirm field (9.4.1.42) containing the output of the random function as described in 12.4.5.5. When transmitting an SAE Confirm message the output of the random function shall be treated as an integer and converted into an octet string of length m, where m is the block size of the random function, according to 12.4.7.2.2 and placed in the Confirm field. When receiving an SAE Confirm message, the octet string in the Confirm field shall be converted into an integer representing the peer's Confirm according to 12.4.7.2.3.

12.4.7.6 Status codes

An SAE Commit message with a status code not equal to SUCCESS shall indicate that a peer rejects a previously sent SAE Commit message. An unsupported finite cyclic group is indicated with a status code of UNSUPPORTED_FINITE_CYCLIC_GROUP, "Authentication is rejected because the offered finite cyclic group is not supported." An Anti-Clogging Token is requested by transmitting an SAE Commit message with a status code of ANTI_CLOGGING_TOKEN_REQUIRED, "Anti-Clogging Token Requested," with the Anti-Clogging Token occupying the Token field of the Authentication frame.

An SAE Confirm message, with a status code not equal to SUCCESS, shall indicate that a peer rejects a previously sent SAE Confirm message. An SAE Confirm message that was not successfully verified is indicated with a status code of CHALLENGE_FAILURE.

12.4.8 SAE finite state machine

12.4.8.1 General

The protocol is instantiated by the finite state machine in Figure 12-4. 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.

NOTE—Figure 12-4 does not show all state machine transitions. A full description of the SAE finite state machine is in 12.4.8.6.2 to 12.4.8.6.6.

The parent process instantiates protocol instances upon receipt of SAE messages and initiation by SME. The parent process also maintains a counter of the number of protocol instances created.

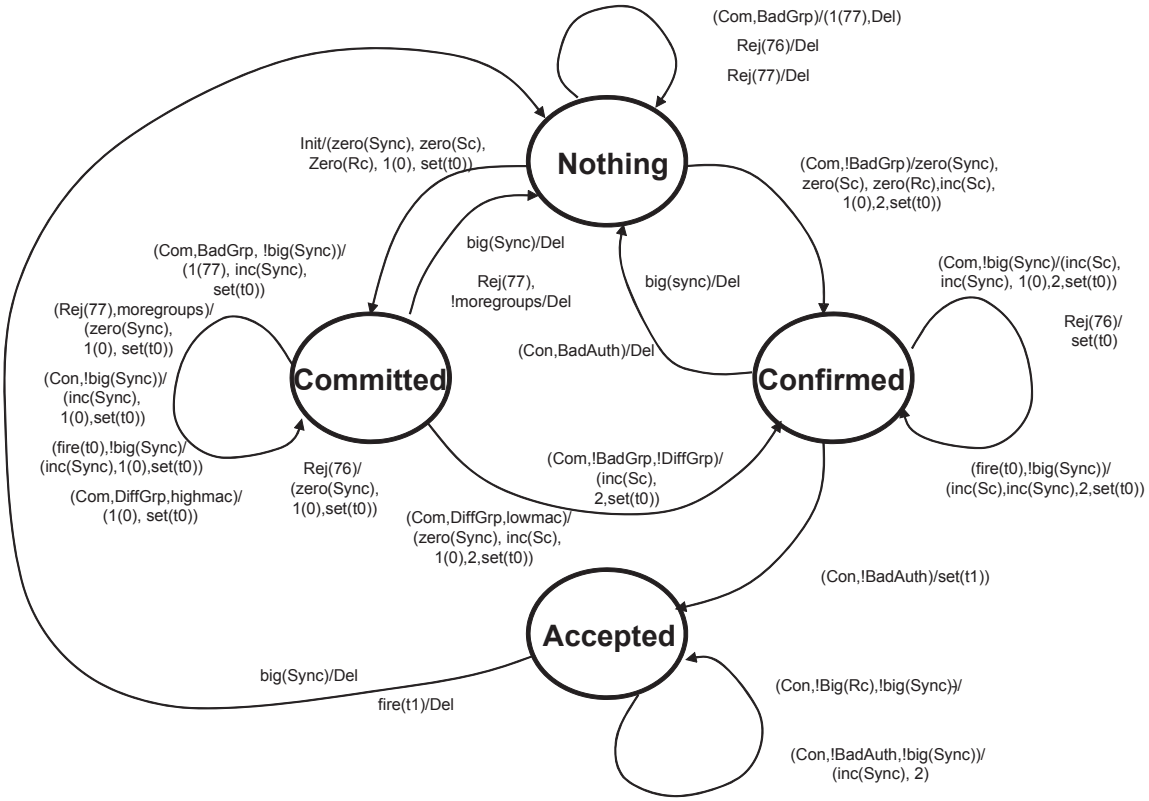

Figure 12-4—SAE finite state machine

12.4.8.2 States

12.4.8.2.1 Parent process states

The parent process is in a continuous quiescent state.

12.4.8.2.2 Protocol instance states

Each protocol instance is in one of the following four states:

— *Nothing*—The *Nothing* state represents the initial state of a freshly allocated protocol instance or the terminal state of a soon-to-be deallocated protocol instance. Freshly created protocol instances will immediately transition out of *Nothing* state depending on the reason for their creation. Protocol instances that transition into *Nothing* state shall immediately be irretrievably deleted.

— *Committed*—In the *Committed* state, the finite state machine has sent an SAE Commit message and is awaiting an SAE Commit message and an SAE Confirm message from the peer.

— *Confirmed*—In the *Confirmed* state, the finite state machine has sent both an SAE Commit message and an SAE Confirm message and received an SAE Commit message. It awaits an SAE Confirm message.

— *Accepted*—In the *Accepted* state, the protocol instance has both sent and received an SAE Commit message and an SAE Confirm message and the protocol instance has finished.

12.4.8.3 Events and output

12.4.8.3.1 Parent process events and output

The parent process receives events from three sources: the SME, protocol instances, and received frames.

The SME signals the following events to the parent SAE process:

— *Initiate*—An *Initiate* event is used to instantiate a protocol instance to begin SAE with a designated peer.

— *Kill*—A *Kill* event is used to remove a protocol instance with a designated peer.

Protocol instances send the following events to the SAE parent process:

— *Fail*—The peer failed to be authenticated.

— *Auth*—The peer was successfully authenticated.

— *Del*—The protocol instance has had a fatal event.

Receipt of frames containing SAE messages signals the following events to the SAE parent process:

— *Authentication frame with Transaction Sequence number 1*—This event indicates that an SAE Commit message has been received from a peer STA.

— *Authentication frame with Transaction Sequence number 2*—This event indicates that an SAE Confirm message has been received from a peer STA.

The parent process generates Authentication frames with Authentication transaction sequence 1 and a Status of 76 indicating rejection of an Authentication attempt because an Anti-Clogging Token is required.

12.4.8.3.2 Protocol instance events and output

The protocol instance receives events from the parent SAE process.

— *Com*—Indicates receipt of an SAE Commit message (authentication transaction sequence number 1) with a status of 0.

— *Con*—Indicates receipt of an SAE Confirm message (authentication transaction sequence number 2) with a status of 0.

— *Init*—Indicates that the protocol instance should begin negotiation with a specified peer.

— *Rej(N)*—Indicates receipt of a rejected SAE Commit message with status *N*.

In addition, protocol instances receive *fire(X)* events indicating the expiration of timer *X*. Upon expiration of a timer and generation of a *fire()* event, the expired timer is not reset.

The protocol instance generates output from the following events:

— *1(N)*—Indicates generation of an SAE Commit message (authentication transaction sequence number 1) with status *N*.

— *2*—Indicates generation of an SAE Confirm message (authentication transaction sequence number 2).

12.4.8.4 Timers

The parent SAE process does not use timers. Each protocol instance can set timers that result in *fire()* events to be sent to itself. The following timers can be set:

— t0—A retransmission timer.

— t1—A PMK expiration timer.

Timers are set by the protocol instance issuing a *set()* for the particular timer.

12.4.8.5 Variables

12.4.8.5.1 Parent process variables

The parent SAE process maintains a counter, *Open*, which indicates the number of protocol instances in either *Committed* or *Confirmed* state. When the parent SAE process starts up, *Open* is set to 0.

The parent process maintains a database of protocol instances.

NOTE—Depending on how Anti-Clogging Tokens (see 12.4.6) are constructed, the parent SAE process might also maintain a random secret used for token creation.

12.4.8.5.2 Protocol instance variables

Each protocol instance maintains the following three variables:

— *Sync*—The number of state resynchronizations that have occurred.
— *Sc*—The number of SAE Confirm messages that have been sent. This is the send-confirm counter used in the construction of SAE Confirm messages (see 12.4.5.5).
— *Rc*—The received value of the *send-confirm* counter in the last received SAE Confirm message. In other words, this is the value of the peer's send-confirm counter.

Function *zero(X)* assigns the value 0 to the variable *X*, *inc(X)* increments the variable *X*, and *big(X)* indicates that the variable *X* has exceeded a maximum value.

In addition, protocol instances maintain the following six indicators that are not maintained as state variables but, instead, indicate the cause of certain behavior.

— *BadGrp*—The group specified in an SAE Commit message is not supported.
— *DiffGrp*—The group specified in an SAE Commit message is supported but differs from the one offered.
— *BadConf*—The contents of a confirm frame were incorrect.
— *highmac*—The peer identity is numerically less than the local identity.
— *lowmac*—The peer identity is numerically greater than the local identity.
— *moregroups*—There are finite cyclic groups in the configuration that have not been offered to the peer.

A negative indication is shown with an exclamation point (!)—e.g., "the group specified in an SAE Commit message is supported" would be !BadGrp, which is read as "not BadGrp."

12.4.8.6 Behavior of state machine

12.4.8.6.1 Parent process behavior

For any given peer identity, there shall be only one protocol instance in *Committed* or *Confirmed* state. Similarly, for any given peer identity, there shall be only one protocol instance in *Accepted* state.

The parent process creates protocol instances based upon different actions. Creating a protocol instance entails allocation of state necessary to maintain the protocol instance state machine, putting the protocol instance in *Nothing* state, incrementing the *Open* counter, and inserting the protocol instance into its database indexed by the MAC address of the peer with whom the protocol instance will communicate.

The parent process shall delete protocol instances irretrievably.

Upon receipt of an *Initiate* event, the parent process shall check whether there exists a protocol instance for the peer MAC address (from the *Init* event) in either *Committed* or *Confirmed* state. If there is, the *Initiate* event shall be ignored. Otherwise, a protocol instance shall be created, and an *Init* event shall be sent to the protocol instance.

Upon receipt of a *Kill* event, the parent process shall delete all protocol instances indexed by the peer MAC address (from the *Kill* event) in its database. For each protocol instance in *Committed* or *Confirmed* state, the *Open* counter shall be decremented.

Upon receipt of a *Sync, Del,* or *Fail* event from a protocol instance, the parent process shall decrement the *Open* counter and deletes the protocol instance.

Upon receipt of an *Auth* event from a protocol instance, the parent process shall decrement the *Open* counter. If another protocol instance exists in the database indexed by the same peer identity as the protocol instance that sent the *Auth* event, the other protocol instance shall be deleted.

Upon receipt of an SAE Commit message, the parent process checks whether a protocol instance for the peer MAC address exists in the database. If one does, and it is in either *Committed* state or *Confirmed* state the frame shall be passed to the protocol instance. If one does and it is in *Accepted* state, the scalar in the received frame is checked against the *peer-scalar* used in authentication of the existing protocol instance (in *Accepted* state). If it is identical, the frame shall be dropped. If not, the parent process checks the value of *Open*. If *Open* is greater than dot11RSNASAEAntiCloggingThreshold, the parent process shall check for the presence of an Anti-Clogging Token. If an Anti-Clogging Token exists and is correct, the parent process shall create a protocol instance. If the Anti-Clogging Token is incorrect, the frame shall be silently discarded. If Open is greater than or equal to dot11RSNASAEAntiCloggingThreshold and there is no Anti-Clogging Token in the received frame, the parent process shall construct a response as an Authentication frame with the Authentication Transaction Sequence Number field set to 1, the Status Code field set to ANTI_CLOGGING_TOKEN_REQUIRED, and the body of the frame consisting of an Anti-Clogging Token (see 12.4.6). If *Open* is less than dot11RSNASAEAntiCloggingThreshold, the parent process shall create a protocol instance and the frame shall be sent to the protocol instance as a *Com* event.

Upon receipt of an SAE Confirm message, the parent process checks whether a protocol instance for the peer MAC address (as indicated by the SA in the received frame) exists in the database. If there is a single protocol instance, the frame shall be passed to it as a *Con* event. If there are two protocol instances indexed by that peer MAC address, the frame shall be passed, as a *Con* event, to the protocol instance that is not in *Accepted* state. If there are no protocol instances indexed by that peer MAC address, the frame shall be dropped.

12.4.8.6.2 Protocol instance behavior—General

State machine behavior is illustrated in Figure 12-4. The protocol instance receives events from the parent process and from itself. It generates SAE messages that are transmitted to a peer and sends events to itself and the parent process.

The semantics of the state diagram are “occurrence/behavior” where “occurrence” is a comma-separated list of events and/or indicators, or the special symbol “—” indicating no occurrence; and, “behavior” is a comma-separated list of outputs and/or functions, or the special symbol “—” indicating no behavior.

When the state machine calls for the t0 (retransmission) timer to be set, it shall be set to dot11RSNASAERetransPeriod. When the state machine calls for the t1 (key expiration) timer to be set, it shall be set to dot11RSNAConfigPMKLifetime.

12.4.8.6.3 Protocol instance behavior - 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 12.4.5.2, generate an SAE Commit message (see 12.4.5.3), 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 not SUCCESS, 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 a an Authentication frame with Status code UNSUPPORTED_FINITE_CYCLIC_GROUP 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 12.4.5.2. It shall then process the received SAE Commit message (see 12.4.5.4). If validation of the received SAE Commit message fails, the protocol instance shall send a Del event to the parent process; otherwise, it shall construct and transmit an SAE Commit message (see 12.4.5.3) followed by an SAE Confirm message (see 12.4.5.5). The *Sync* counter shall be set to 0 and the t0 (retransmission) timer shall be set. The protocol instance transitions to *Confirmed* state.

NOTE—A protocol instance in *Nothing* state will never receive an SAE Confirm message due to state machine behavior of the parent process.

12.4.8.6.4 Protocol instance behavior - Committed state

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

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

- The protocol instance shall check the Status code of the Authentication frame. If the Status code is ANTI_CLOGGING_TOKEN_REQUIRED, a new SAE Commit message shall be constructed with the Anti-Clogging Token from the received Authentication frame, and the *commit-scalar* and ***COMMIT-ELEMENT*** previously sent. The new SAE 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 UNSUPPORTED_FINITE_CYCLIC_GROUP, 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 and generate the ***PWE*** and the secret values according to 12.4.5.2; it then generates and transmits a new SAE 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* state.
- If the Status is some other nonzero 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* state.
  - If *Sync* is not greater than dot11RSNASAESync, *Sync* shall be incremented, an SAE Commit message with Status code equal to UNSUPPORTED_FINITE_CYCLIC_GROUP 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 SAE 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 SAE Commit message, retransmits its last SAE 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 SAE Commit message, generate new ***PWE*** and new secret values according to 12.4.5.2, process the received SAE Commit message according to 12.4.5.4, generate a new SAE Commit message and SAE Confirm message, and shall transmit the new Commit and Confirm to the peer. It shall then transition to *Confirmed* state.
— If the group is supported and matches that used when the protocol instance constructed its SAE Commit message, the protocol instance checks the *peer-commit-scalar* and ***PEER-COMMIT-ELEMENT*** from the message. If they match those sent as part of the protocol instance's own SAE Commit message, the frame shall be silently discarded (because it is evidence of a reflection attack) and the t0 (retransmission) timer shall be set. If the received element and scalar differ from the element and scalar offered, the received SAE Commit message shall be processed according to 12.4.5.4, the *Sc* counter shall be incremented (thereby setting its value to one), the protocol instance shall then construct an SAE Confirm message, transmit it to the peer, and set the t0 (retransmission) timer. It shall then transition to *Confirmed* state.

If the t0 (retransmission) timer fires, the value of the *Sync* counter is checked. If *Sync* is greater than dot11RSNASAESync, the protocol instance shall send a *Del* event to the parent process and transition back to *Nothing* state. If *Sync* is not greater than dot11RSNASAESync, the *Sync* counter shall be incremented, the last message sent shall be sent again, and the t0 (retransmission) timer shall be set.

Upon receipt of a *Con* event, the t0 (retransmission) timer shall be canceled. Then the protocol instance checks the value of *Sync*. If it is greater than dot11RSNASAESync, the protocol instance shall send a *Del* event to the parent process and transition back to *Nothing* state. If *Sync* is not greater than dot11RSNASAESync, the protocol instance shall increment *Sync*, transmit the last SAE Commit message sent to the peer, and set the t0 (retransmission) timer.

12.4.8.6.5 Protocol instance behavior - Confirmed state

In *Confirmed* state, a protocol instance has sent its peer an SAE Commit message and SAE Confirm message. It has received an SAE Commit message from its peer.

Rejection frames received in Confirmed state shall be silently discarded.

Upon receipt of a *Com* event, the t0 (retransmission) timer shall be canceled. If the Status is nonzero, the frame shall be silently discarded, the t0 (retransmission) timer set, and the protocol instance shall remain in the *Confirmed* state. If *Sync* is greater than dot11RSNASAESync, the protocol instance shall send the parent process a *Del* event and transitions back to *Nothing* state. If *Sync* is not greater than dot11RSNASAESync, the protocol instance shall verify that the finite cyclic group is the same as the previously received Commit frame. If not, the frame shall be silently discarded. If so, the protocol instance shall increment *Sync*, increment *Sc*, and transmit its Commit and Confirm (with the new *Sc* value) messages. It then shall set the t0 (retransmission) timer.

Upon receipt of a *Con* event, the t0 (retransmission) timer shall be canceled and the SAE Confirm message shall be processed according to 12.4.5.6. If processing is successful and the SAE Confirm message has been

verified, the *Rc* variable shall be set to the send-confirm portion of the frame, *Sc* shall be set to the value $2^{16} - 1$, the t1 (key expiration) timer shall be set, and the protocol instance shall transition to *Accepted* state.

If the t0 (retransmission) timer fires, the value of the *Sync* counter shall be checked. If *Sync* is greater than dot11RSNASAESync, the protocol instance shall send a *Del* event to the parent process and transition back to *Nothing* state. If *Sync* is not greater than dot11RSNASAESync, the *Sync* counter shall be incremented, *Sc* shall be incremented, and the protocol instance shall create a new Confirm (with the new *Sc* value) Message, transmit it to the peer, and set the t0 (retransmission) timer.

12.4.8.6.6 Protocol instance behavior - Accepted state

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

Upon receipt of a *Con* event, the *Sync* counter shall be checked. If the value is greater than dot11RSNASAESync, the protocol instance shall send a *Del* event to the parent process and shall transition to *Nothing* state. If the value of *Sync* is not greater than dot11RSNASAESync, the value of send-confirm shall be checked. If the value is not greater than *Rc* or is equal to $2^{16} - 1$, the received frame shall be silently discarded. Otherwise, the Confirm portion of the frame shall be checked according to 12.4.5.6. If the verification fails, the received frame shall be silently discarded. If the verification succeeds, the *Rc* variable shall be set to the send-confirm portion of the frame, the *Sync* shall be incremented and a new SAE Confirm message shall be constructed (with *Sc* set to $2^{16} - 1$) and sent to the peer. The protocol instance shall remain in *Accepted* state.

If the t1 (key expiration) timer fires, the protocol instance shall send the parent process a *Del* event and transition to *Nothing* state.

12.5 RSNA confidentiality and integrity protocols

12.5.1 Overview

This standard defines the following RSNA data confidentiality and integrity protocols: TKIP, CCMP, and GCMP. This standard defines one integrity protocol for Management frames: BIP.

Implementation of TKIP is optional for an RSNA and used only for the protection of Data frames. A design aim for TKIP was that the algorithm should be implementable within the capabilities of most devices supporting only WEP, so that many such devices would be field-upgradable by the supplier to support TKIP.

BIP is a mechanism that is used only when management frame protection is negotiated. BIP provides integrity protection for group addressed robust Management frames. BIP is used only to protect Management frames within the BSS.

12.5.2 Temporal key integrity protocol (TKIP)

12.5.2.1 TKIP overview

12.5.2.1.1 General

The TKIP is a cipher suite enhancing WEP on pre-RSNA hardware. TKIP modifies WEP as follows:

a) A transmitter calculates a keyed cryptographic message integrity code (MIC) over the MSDU SA and DA, the MSDU priority (see 12.5.2.3), and the MSDU plaintext data. TKIP appends the computed MIC to the MSDU data prior to fragmentation into MPDUs. The receiver verifies

the MIC after decryption, ICV checking, and defragmentation of the MPDUs into an MSDU and discards any received MSDUs with invalid MICs. TKIP's MIC provides a defense against forgery attacks.

b) Because of the design constraints of the TKIP MIC, it is still possible for an adversary to compromise message integrity; therefore, TKIP also implements countermeasures. The countermeasures bound the probability of a successful forgery and the amount of information an attacker can learn about a key.

c) TKIP uses a per-MPDU TKIP sequence counter (TSC) to sequence the MPDUs it sends. The receiver drops MPDUs received out of order, i.e., not received with increasing sequence numbers. This provides replay protection. TKIP encodes the TSC value from the sender to the receiver as a WEP IV and extended IV.

d) TKIP uses a cryptographic mixing function to combine a temporal key, the TA, and the TSC into the WEP seed. The receiver recovers the TSC from a received MPDU and utilizes the mixing function to compute the same WEP seed needed to correctly decrypt the MPDU. The key mixing function is designed to defeat weak-key attacks against the WEP key.

TKIP defines additional MIB variables; see Annex C.

12.5.2.1.2 TKIP cryptographic encapsulation

TKIP enhances the WEP cryptographic encapsulation with several additional functions, as depicted in Figure 12-5.

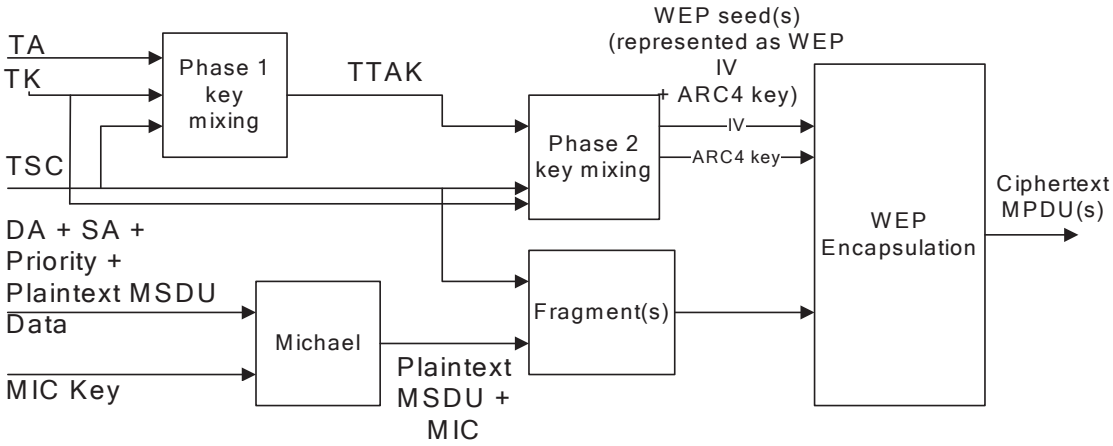

Figure 12-5—TKIP encapsulation block diagram

a) TKIP MIC computation protects the MSDU Data field and corresponding SA, DA, and Priority fields. The computation of the MIC is performed on the ordered concatenation of the SA, DA, Priority, and MSDU Data fields. The MIC is appended to the MSDU Data field. TKIP discards any MIC padding prior to appending the MIC.

b) If needed, the MSDU with MIC is fragmented into one or more MPDUs. TKIP assigns a monotonically increasing TSC value to each MPDU, taking care that all of the MPDUs generated from the same MSDU have the same value of extended IV (see 12.5.2.2).

c) For each MPDU, TKIP uses the key mixing function to compute the WEP seed.

d) TKIP represents the WEP seed as a WEP IV and ARC4 key and passes these with each MPDU to WEP for generation of the ICV (see 9.2.4.7), and for encryption of the plaintext MPDU, including all or part of the MIC, if present. WEP uses the WEP seed as a WEP default key, identified by a key identifier associated with the temporal key.

NOTE—When the TSC space is exhausted, the choices available to an implementation are to replace the temporal key with a new one or to end communications. Reuse of any TSC value compromises already sent traffic. Note that retransmitted MPDUs reuse the TSC without any compromise of security. The TSC is large enough, however, that TSC space exhaustion is not expected to be an issue.

In Figure 12-5, the TKIP-mixed transmit address and key (TTAK) denotes the intermediate key produced by Phase 1 of the TKIP mixing function (see 12.5.2.5).

12.5.2.1.3 TKIP decapsulation

TKIP enhances the WEP decapsulation process with the following additional steps:

a) Before WEP decapsulates a received MPDU, TKIP extracts the TSC sequence number and key identifier from the WEP IV and the extended IV. TKIP discards a received MPDU that violates the sequencing rules (see 12.5.2.6) and otherwise uses the mixing function to construct the WEP seed.

b) TKIP represents the WEP seed as a WEP IV and ARC4 key and passes these with the MPDU to WEP for decapsulation.

c) If WEP indicates the ICV check succeeded, the implementation reassembles the MPDU into an MSDU. If the MSDU defragmentation succeeds, the receiver verifies the TKIP MIC. If MSDU defragmentation fails, then the MSDU is discarded.

d) The MIC verification step recomputes the MIC over the MSDU SA, DA, Priority, and MSDU Data fields (but not the TKIP MIC field). The calculated TKIP MIC result is then compared bitwise to the received MIC.

e) If the received and the locally computed MIC values are identical, the verification succeeds, and TKIP shall deliver the MSDU to the upper layer. If the two differ, then the verification fails; the receiver shall discard the MSDU and shall engage in appropriate countermeasures.

Figure 12-6 depicts this process.

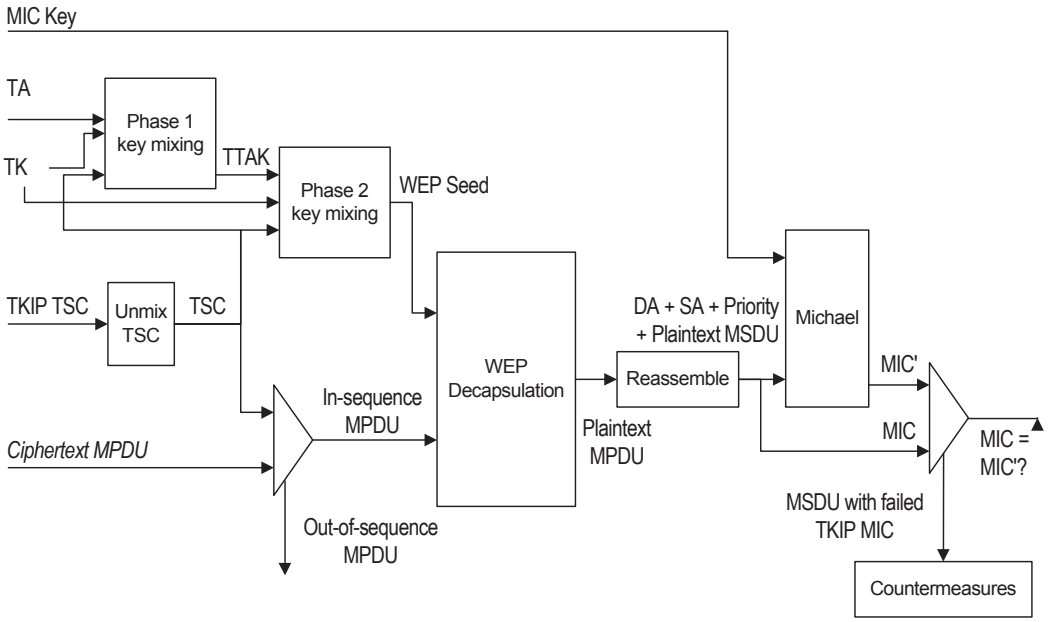

Figure 12-6—TKIP decapsulation block diagram

12.5.2.2 TKIP MPDU formats

TKIP reuses the pre-RSNA WEP MPDU format. It extends the MPDU by 4 octets to accommodate an extension to the WEP IV, denoted by the Extended IV field, and extends the MSDU format by 8 octets to accommodate the new MIC field. TKIP inserts the Extended IV field immediately after the WEP IV field and before the encrypted data. TKIP appends the MIC to the MSDU Data field; the MIC becomes part of the encrypted data.

Once the MIC is appended to the MSDU data, the added MIC octets are considered part of the MSDU for subsequent fragmentation.

Figure 12-7 depicts the layout of the encrypted MPDU when using TKIP. Note that the figure only depicts the case when the MSDU can be encapsulated in a single MPDU.

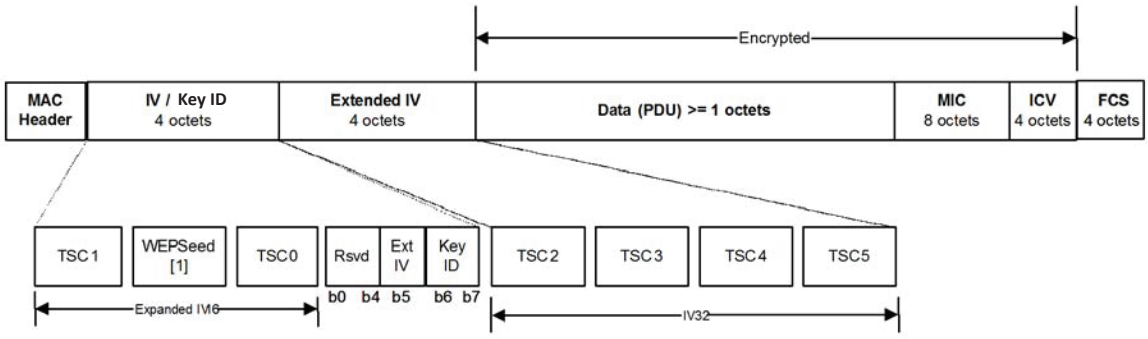

Figure 12-7—Construction of expanded TKIP MPDU

The ExtIV bit in the Key ID octet indicates the presence or absence of an extended IV. If the ExtIV bit is 0, only the nonextended IV is transferred. If the ExtIV bit is 1, an extended IV of 4 octets follows the original IV. For TKIP the ExtIV bit shall be set to 1, and the Extended IV field shall be supplied. The ExtIV bit shall be 0 for WEP frames. The Key ID field shall be set to the key index supplied by the MLME-SETKEYS.request primitive for the key used in cryptographic encapsulation of the frame.

TSC5 is the most significant octet of the TSC, and TSC0 is the least significant. Octets TSC0 and TSC1 form the IV sequence number and are used with the TKIP Phase 2 key mixing. Octets TSC2–TSC5 are used in the TKIP Phase 1 key hashing and are in the Extended IV field. When the lower 16-bit sequence number rolls over (0xFFFF $\rightarrow$ 0x0000), the extended IV value, i.e., the upper 32 bits of the entire 48-bit TSC, shall be incremented by 1.

NOTE—The rationale for this construction is as follows:

- — Aligning on word boundaries eases implementation on legacy devices.
- — Adding 4 octets of extended IV eliminates TSC exhaustion as a reason to rekey.
- — Key ID octet changes. Bit 5 indicates that an extended IV is present. The receiver/transmitter interprets the 4 octets following the Key ID as the extended IV. The receiving/transmitting STA also uses the value of octets TSC0 and TSC1 to detect that the cached TTAK needs to be updated.

The Extended IV field shall not be encrypted.

WEPSeed[1] is not used to construct the TSC, but is set to (TSC1 | 0x20) & 0x7f.

TKIP shall encrypt all of the MPDUs generated from one MSDU under the same temporal key.

12.5.2.3 TKIP MIC

12.5.2.3.1 General

Flaws in the IEEE 802.11 WEP design cause it to fail to meet its goal of protecting data traffic content from casual eavesdroppers. Among the most significant WEP flaws is the lack of a mechanism to defeat message forgeries and other active attacks. To defend against active attacks, TKIP includes a MIC, called *michael*. This MIC offers only weak defenses against message forgeries, but it constitutes the best that can be achieved with the majority of legacy hardware. TKIP uses different MIC keys depending on the direction of the transfer as described in 12.8.1 and 12.8.2.

Annex J contains an implementation of the TKIP MIC. It also provides test vectors for the MIC.

12.5.2.3.2 Motivation for the TKIP MIC

Before defining the details of the MIC, it is useful to review the context in which this mechanism operates. Active attacks enabled by the original WEP design include the following:

— Bit-flipping attacks
— Data (payload) truncation, concatenation, and splicing
— Fragmentation attacks
— Iterative guessing attacks against the key
— Redirection by modifying the MPDU DA or RA field
— Impersonation attacks by modifying the MPDU SA or TA field

The MIC makes it more difficult for any of these attacks to succeed.

All of these attacks remain at the MPDU level with the TKIP MIC. The MIC, however, applies to the MSDU, so it blocks successful MPDU-level attacks. TKIP applies the MIC to the MSDU at the transmitter and verifies it at the MSDU level at the receiver. If a MIC check fails at the MSDU level, the implementation shall discard the MSDU and invoke countermeasures (see 12.5.2.4).

Informative Figure 12-8 depicts different peer layers communicating.

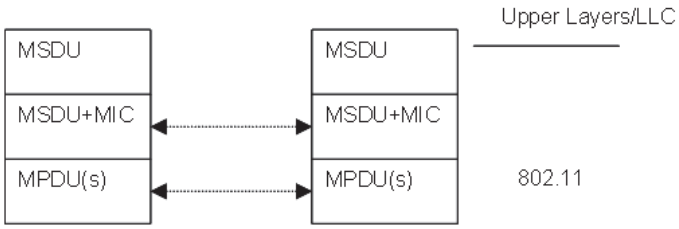

Figure 12-8—TKIP MIC relation to IEEE 802.11 processing (informative)

This figure depicts an architecture where the MIC is logically appended to the raw MSDU in response to the MA-UNITDATA.request primitive. The TKIP MIC is computed over

— The MSDU DA
— The MSDU SA
— The MSDU Priority
— The entire unencrypted MSDU data (payload)

The DA field, SA field, three reserved octets, and a 1-octet Priority field are used only for calculating the MIC. The Priority field refers to the priority parameter of the MA-UNITDATA.request primitive. The fields in Figure 12-9 are treated as an octet stream using the conventions described in 9.2.2.

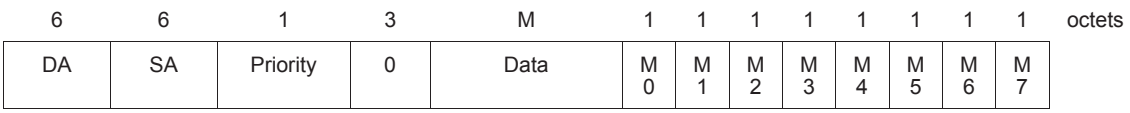

Figure 12-9—TKIP MIC processing format

TKIP appends the MIC at the end of the MSDU. The MIC is 8 octets in size for michael. The IEEE 802.11 MAC then applies its normal processing to transmit this MSDU-with-MIC as a sequence of one or more MPDUs. In other words, the MSDU-with-MIC can be partitioned into one or more MPDUs, the WEP ICV is calculated over each MPDU, and the MIC can even be partitioned to lie in two MPDUs after fragmentation. The TKIP MIC augments, but does not replace, the WEP ICV. Because the TKIP MIC is a weak construction, TKIP protects the MIC with encryption, which makes TKIP MIC forgeries more difficult. The WEP ICV helps to prevent false detection of MIC failures that would cause countermeasures to be invoked.

The receiver reverses this procedure to reassemble the MSDU; and, after the MSDU has been logically reassembled, the IEEE 802.11 MAC verifies the MIC prior to delivery of the MSDU to upper layers. If the MIC validation succeeds, the MAC delivers the MSDU. If the MIC validation fails, the MAC shall discard the MSDU and invoke countermeasures (see 12.5.2.4).

NOTE—TKIP calculates the MIC over the MSDU rather than the MPDU, because doing so increases the implementation flexibility with preexisting WEP hardware.

Note that a MIC alone cannot provide complete forgery protection, as it cannot defend against replay attacks. Therefore, TKIP provides replay detection by TSC sequencing and ICV validation. Furthermore, if TKIP is utilized with a GTK, an insider STA can masquerade as any other STA belonging to the group.

12.5.2.3.3 Definition of the TKIP MIC

Michael generates a 64-bit MIC. The michael key consists of 64 bits, represented as an 8-octet sequence, $k_0 ... k_7$. This is converted to two 32-bit words, K_0 and K_1. Throughout this subclause, all conversions between octets and 32-bit words shall use the little endian conventions, given in 9.2.2.

Michael operates on each MSDU including the Priority field, 3 reserved octets, SA field, and DA field. An MSDU consists of octets $m_0 ... m_{n-1}$ where n is the number of MSDU octets, including SA, DA, Priority, and Data fields. The message is padded at the end with a single octet with value 0x5a, followed by between 4 and 7 zero octets. The number of zero octets is chosen so that the overall length of the padded MSDU is a multiple of four. The padding is not transmitted with the MSDU; it is used to simplify the computation over the final block. The MSDU is then converted to a sequence of 32-bit words $M_0 ... M_{N-1}$, where $N = \lceil (n+5)/4 \rceil$. By construction, $M_{N-1} = 0$ and $M_{N-2} \neq 0$.

The MIC value is computed iteratively starting with the key value (K_0 and K_1) and applying a block function b for every message word, as shown in Figure 12-10. The algorithm loop runs a total of N times (i takes on the values 0 to N–1 inclusive), where N is as above, the number of 32-bit words composing the padded MSDU. The algorithm results in two words (l and r), which are converted to a sequence of 8 octets using the least-significant-octet-first convention:

— M0 = l & 0xff
— M1 = (l/0x100) & 0xff
— M2 = (l/0x10000) & 0xff
— M3 = (l/0x1000000) & 0xff

— M4 = r & 0xff
— M5 = (r/0x100) & 0xff
— M6 = (r/0x10000) & 0xff
— M7 = (r/0x1000000) & 0xff

This is the MIC value. The MIC value is appended to the MSDU as data to be sent.

Input: Key (K_0, K_1) and padded MSDU (represented as 32-bit words) M_0 ... M_{N-1}
Output: MIC value (V_0, V_1)

MICHAEL((K_0, K_1) , (M_0, ... , M_N))
(l,r) ← (K_0, K_1)
for i = 0 **to** N–1 **do**
    $l \leftarrow l \oplus M_i$
    (l,r) ← b(l,r)
return (l,r)

Figure 12-10—Michael message processing

Figure 12-11 defines the michael block function b. It is a Feistel-type construction with alternating additions and XOR operations. It uses <<< to denote the rotate-left operator on 32-bit values, >>> for the rotate-right operator, and XSWAP for a function that swaps the position of the 2 least significant octets. It also uses the position of the two most significant octets in a word.

Input: (l,r)
Output: (l,r)
b(L,R)
    $r \leftarrow r \oplus (l <<< 17)$
    $l \leftarrow (l + r) \bmod 2^{32}$
    $r \leftarrow r \oplus$ XSWAP(l)
    $l \leftarrow (l + r) \bmod 2^{32}$
    $r \leftarrow r \oplus (l <<< 3)$
    $l \leftarrow (l + r) \bmod 2^{32}$
    $r \leftarrow r \oplus (l >>> 2)$
    $l \leftarrow (l + r) \bmod 2^{32}$
    return (l,r)

Figure 12-11—Michael block function

12.5.2.4 TKIP countermeasures procedures

12.5.2.4.1 General

The TKIP MIC trades off security in favor of implementability on pre-RSNA STAs. Michael provides only weak protection against active attacks. A failure of the MIC in a received MSDU indicates a probable active attack. A successful attack against the MIC would mean an attacker could inject forged Data frames and perform further effective attacks against the encryption key itself. If TKIP implementation detects a probable active attack, TKIP shall take countermeasures as specified in this subclause. These countermeasures accomplish the following goals:

— MIC failure events should be logged as a security-relevant matter. A MIC failure is an almost certain indication of an active attack and warrants a follow-up by the system administrator.
— The rate of MIC failures needs to be kept below two per minute. This implies that STAs and APs detecting two MIC failure events within 60 s need to disable all receptions using TKIP for a period of 60 s. The slowdown makes it difficult for an attacker to make a large number of forgery attempts in a short time.

— As an additional security feature, the PTK and, in the case of the Authenticator, the GTK should be changed.

Before verifying the MIC, the receiver shall check the FCS, ICV, and TSC for all related MPDUs. Any MPDU that has an invalid FCS, an incorrect ICV, or a TSC value that is less than or equal to the TSC replay counter shall be discarded before checking the MIC. This avoids unnecessary MIC failure events. Checking the TSC before the MIC makes countermeasure-based denial-of-service attacks harder to perform. While the FCS and ICV mechanisms are sufficient to detect noise, they are insufficient to detect active attacks. The FCS and ICV provide error detection, but not integrity protection.

A single counter or timer shall be used to log MIC failure events. These failure events are defined as follows:

— In an Authenticator:
  — Detection of a MIC failure on a received individually addressed frame.
  — Receipt of Michael MIC Failure Report frame.
— In a Supplicant:
  — Detection of a MIC failure on a received individually addressed or group addressed frame.
  — Attempt to transmit a Michael MIC Failure Report frame.

The number of MIC failures is accrued independent of the particular key context. Any single MIC failure, whether detected by the Supplicant or the Authenticator and whether resulting from a group MIC key failure or a pairwise MIC key failure, shall be treated as cause for a MIC failure event.

The Supplicant uses a single Michael MIC Failure Report frame to report a MIC failure event to the Authenticator. A Michael MIC Failure Report is an EAPOL-Key frame with the following Key Information field bits set to 1: MIC bit, Error bit, Request bit, Secure bit. The Supplicant protects this message with the current PTK; the Supplicant uses the KCK portion of the PTK to compute the IEEE 802.1X EAPOL MIC.

The MLME-MICHAELMICFAILURE.indication primitive is used by the IEEE 802.11 MAC to attempt to indicate a MIC failure to the local IEEE 802.1X Supplicant or Authenticator. MLME-EAPOL.request primitive is used by the Supplicant to send the EAPOL-Key frame containing the Michael MIC Failure report. MLME-EAPOL.confirm primitive indicates to the Supplicant when the EAPOL-Key frame has been transmitted.

The first MIC failure shall be logged, and a timer initiated to enable enforcement of the countermeasures. If the MIC failure event is detected by the Supplicant, it shall also report the event to the AP by sending a Michael MIC Failure Report frame.

If a subsequent MIC failure occurs within 60 s of the most recent previous failure, then a STA whose IEEE 802.1X entity has acted as a Supplicant shall deauthenticate (as defined in 11.3.4.4) itself or deauthenticate all of the STAs with a security association if its IEEE 802.1X entity acted as an Authenticator. In an IBSS STA, both Supplicant and Authenticator actions shall be taken. Furthermore, the device shall not receive or transmit any TKIP-encrypted Data frames, and shall not receive or transmit any unencrypted Data frames other than IEEE 802.1X messages, to or from any peer for a period of at least 60 s after it detects the second failure. If the device is an AP, it shall disallow new associations using TKIP during this 60 s period; at the end of the 60 s period, the AP shall resume normal operations and allow STAs to (re)associate. If the device is an IBSS STA, it shall disallow any new security associations using TKIP during this 60 s period. If the device is a Supplicant, it shall first send a Michael MIC Failure Report frame prior to revoking its PTKSA and deauthenticating itself.

The `aMICFailTime` attribute shall contain the `sysUpTime` value at the time the MIC failure was logged.

12.5.2.4.2 TKIP countermeasures for an Authenticator

The countermeasures used by an Authenticator are depicted in Figure 12-12 and described as follows:

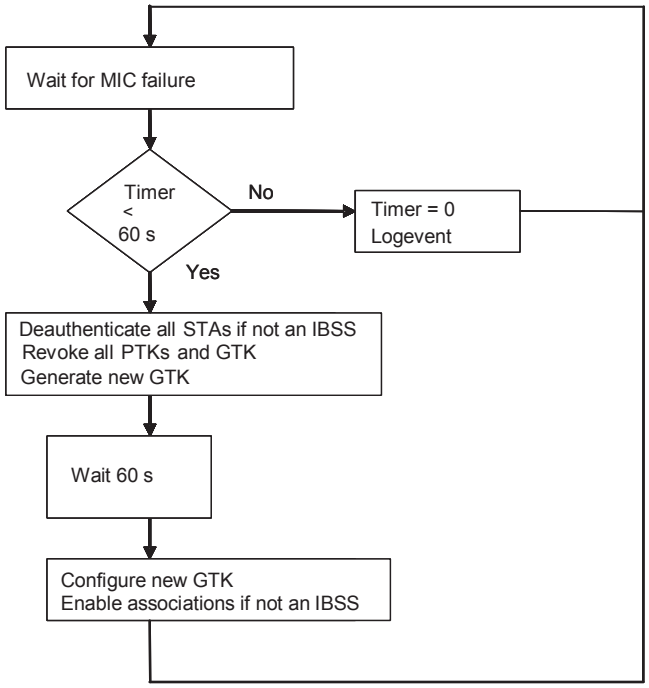

Figure 12-12—Authenticator MIC countermeasures

a) In an Authenticator's STA that receives a frame with a MIC error,
   1) Discard the frame.
   2) Increment the MIC failure counter, dot11RSNAStatsTKIPLocalMICFailures.
   3) Generate an MLME-MICHAELMICFAILURE.indication primitive.

b) In an Authenticator that receives an MLME-MICHAELMICFAILURE.indication primitive or a Michael MIC Failure Report frame,
   1) If it is a Michael MIC Failure Report frame, then increment dot11RSNAStatsTKIP-RemoteMICFailures.
   2) If this is the first MIC failure within the past 60 s, initialize the countermeasures timer.
   3) If less than 60 s have passed since the most recent previous MIC failure, the Authenticator shall deauthenticate and delete all PTKSAs for all STAs using TKIP. If the current GTKSA uses TKIP, that GTKSA shall be discarded, and a new GTKSA constructed, but not used for 60 s. The Authenticator shall refuse the construction of new PTKSAs using TKIP as one or more of the ciphers for 60 s. At the end of this period, the MIC failure counter and timer shall be reset, and creation of PTKSAs accepted as usual.
   4) If the Authenticator is using IEEE 802.1X authentication, the Authenticator shall transition the state of the IEEE 802.1X Authenticator state machine to the INITIALIZE state. This restarts the IEEE 802.1X state machine. If the Authenticator is instead using PSKs, this step is omitted.

Note that a Supplicant's STA may deauthenticate with a reason code of MIC_FAILURE if it is an ESS STA. The Authenticator shall not log the deauthenticate as a MIC failure event to prevent denial-of-service attacks through deauthentications. The Supplicant's STA reports the MIC failure event through the Michael MIC Failure Report frame so that the AP can log the event.

The requirement to deauthenticate all STAs using TKIP includes those using other pairwise ciphers if they are using TKIP as the group cipher.

12.5.2.4.3 TKIP countermeasures for a Supplicant

The countermeasures used by a Supplicant are depicted in Figure 12-13 and described as follows:

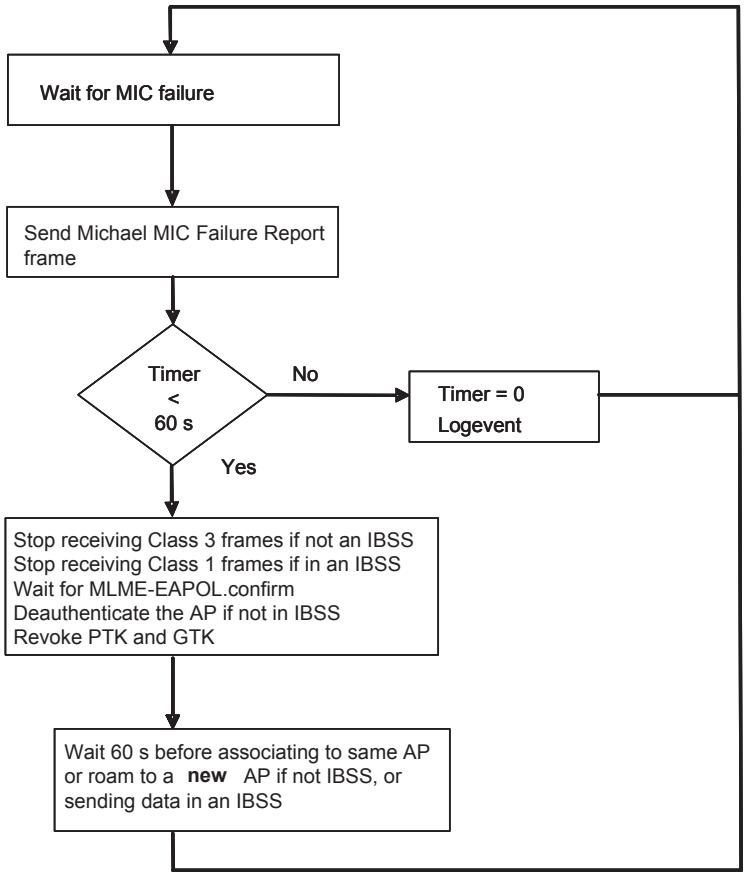

Figure 12-13—Supplicant MIC countermeasures

a) In a Supplicant's STA that receives a frame with a MIC error,
   1) Increment the MIC failure counter, dot11RSNAStatsTKIPLocalMICFailures.
   2) Discard the offending frame.
   3) Generate an MLME-MICHAELMICFAILURE.indication primitive.

b) In a Supplicant that receives an MLME-MICHAELMICFAILURE.indication primitive from its STA,
   1) Send a Michael MIC Failure Report frame to the AP.
   2) If this is the first MIC failure within the past 60 s, initialize the countermeasures timer.
   3) If less than 60 s have passed since the most recent previous MIC failure, delete the PTKSA and GTKSA. Deauthenticate from the AP and wait for 60 s before (re)establishing a TKIP association with the same AP. A TKIP association is any IEEE 802.11 association that uses TKIP for its pairwise or group cipher suite.

c) If a STA receives a deauthenticate frame with the reason code MIC_FAILURE it is unable to verify that the frame has not been forged, since the frame does not contain a MIC. The STA may attempt association with this, or another, AP. If the frame was genuine, then it is probable that attempts to

associate with the same AP requesting the use of TKIP will fail because the AP is conducting countermeasures.

12.5.2.5 TKIP mixing function

12.5.2.5.1 General

Annex J defines a C-language reference implementation of the TKIP mixing function. It also provides test vectors for the mixing function.

The mixing function has two phases. Phase 1 mixes the appropriate temporal key (pairwise or group) with the TA and TSC. A STA may cache the output of this phase to reuse with subsequent MPDUs associated with the same temporal key and TA. Phase 2 mixes the output of Phase 1 with the TSC and temporal key (TK) to produce the WEP seed, also called the *per-frame key*. The WEP seed may be precomputed before it is used. The two-phase process may be summarized as follows:

```
TTAK := Phase1 (TK, TA, TSC)
WEP seed := Phase2 (TTAK, TK, TSC)
```

12.5.2.5.2 S-Box

Both Phase 1 and Phase 2 rely on an S-box, defined in this subclause. The S-box substitutes one 16-bit value with another 16-bit value. This function may be implemented as a table look up.

NOTE—The S-box is a nonlinear substitution. The table look-up is be organized as either a single table with 65 536 entries and a 16-bit index (128K octets of table) or two tables with 256 entries and an 8-bit index (1024 octets for both tables). When the two smaller tables are used, the high-order octet is used to obtain a 16-bit value from one table, the low-order octet is used to obtain a 16-bit value from the other table, and the S-box output is the XOR of the two 16-bit values. The second S-box table is an octet-swapped replica of the first.

```
#define _S_(v16)     (Sbox[0][Lo8(v16)] ^ Sbox[1][Hi8(v16)])

/* 2-byte by 2-byte subset of the full AES S-box table */
const u16b Sbox[2][256]=       /* Sbox for hash (can be in ROM)     */
{ {
   0xC6A5,0xF884,0xEE99,0xF68D,0xFF0D,0xD6BD,0xDEB1,0x9154,
   0x6050,0x0203,0xCEA9,0x567D,0xE719,0xB562,0x4DE6,0xEC9A,
   0x8F45,0x1F9D,0x8940,0xFA87,0xEF15,0xB2EB,0x8EC9,0xFB0B,
   0x41EC,0xB367,0x5FFD,0x45EA,0x23BF,0x53F7,0xE496,0x9B5B,
   0x75C2,0xE11C,0x3DAE,0x4C6A,0x6C5A,0x7E41,0xF502,0x834F,
   0x685C,0x51F4,0xD134,0xF908,0xE293,0xAB73,0x6253,0x2A3F,
   0x080C,0x9552,0x4665,0x9D5E,0x3028,0x37A1,0x0A0F,0x2FB5,
   0x0E09,0x2436,0x1B9B,0xDF3D,0xCD26,0x4E69,0x7FCD,0xEA9F,
   0x121B,0x1D9E,0x5874,0x342E,0x362D,0xDCB2,0xB4EE,0x5BFB,
   0xA4F6,0x764D,0xB761,0x7DCE,0x527B,0xDD3E,0x5E71,0x1397,
   0xA6F5,0xB968,0x0000,0xC12C,0x4060,0xE31F,0x79C8,0xB6ED,
   0xD4BE,0x8D46,0x67D9,0x724B,0x94DE,0x98D4,0xB0E8,0x854A,
   0xBB6B,0xC52A,0x4FE5,0xED16,0x86C5,0x9AD7,0x6655,0x1194,
   0x8ACF,0xE910,0x0406,0xFE81,0xA0F0,0x7844,0x25BA,0x4BE3,
   0xA2F3,0x5DFE,0x80C0,0x058A,0x3FAD,0x21BC,0x7048,0xF104,
   0x63DF,0x77C1,0xAF75,0x4263,0x2030,0xE51A,0xFD0E,0xBF6D,
   0x814C,0x1814,0x2635,0xC32F,0xBEE1,0x35A2,0x88CC,0x2E39,
   0x9357,0x55F2,0xFC82,0x7A47,0xC8AC,0xBAE7,0x322B,0xE695,
   0xC0A0,0x1998,0x9ED1,0xA37F,0x4466,0x547E,0x3BAB,0x0B83,
   0x8CCA,0xC729,0x6BD3,0x283C,0xA779,0xBCE2,0x161D,0xAD76,
   0xDB3B,0x6456,0x744E,0x141E,0x92DB,0x0C0A,0x486C,0xB8E4,
   0x9F5D,0xBD6E,0x43EF,0xC4A6,0x39A8,0x31A4,0xD337,0xF28B,
   0xD532,0x8B43,0x6E59,0xDAB7,0x018C,0xB164,0x9CD2,0x49E0,
```

```
    0xD8B4,0xACFA,0xF307,0xCF25,0xCAAF,0xF48E,0x47E9,0x1018,
    0x6FD5,0xF088,0x4A6F,0x5C72,0x3824,0x57F1,0x73C7,0x9751,
    0xCB23,0xA17C,0xE89C,0x3E21,0x96DD,0x61DC,0x0D86,0x0F85,
    0xE090,0x7C42,0x71C4,0xCCAA,0x90D8,0x0605,0xF701,0x1C12,
    0xC2A3,0x6A5F,0xAEF9,0x69D0,0x1791,0x9958,0x3A27,0x27B9,
    0xD938,0xEB13,0x2BB3,0x2233,0xD2BB,0xA970,0x0789,0x33A7,
    0x2DB6,0x3C22,0x1592,0xC920,0x8749,0xAAFF,0x5078,0xA57A,
    0x038F,0x59F8,0x0980,0x1A17,0x65DA,0xD731,0x84C6,0xD0B8,
    0x82C3,0x29B0,0x5A77,0x1E11,0x7BCB,0xA8FC,0x6DD6,0x2C3A,
  },
  {  /* second half of table is byte-reversed version of first! */
    0xA5C6,0x84F8,0x99EE,0x8DF6,0x0DFF,0xBDD6,0xB1DE,0x5491,
    0x5060,0x0302,0xA9CE,0x7D56,0x19E7,0x62B5,0xE64D,0x9AEC,
    0x458F,0x9D1F,0x4089,0x87FA,0x15EF,0xEBB2,0xC98E,0x0BFB,
    0xEC41,0x67B3,0xFD5F,0xEA45,0xBF23,0xF753,0x96E4,0x5B9B,
    0xC275,0x1CE1,0xAE3D,0x6A4C,0x5A6C,0x417E,0x02F5,0x4F83,
    0x5C68,0xF451,0x34D1,0x08F9,0x93E2,0x73AB,0x5362,0x3F2A,
    0x0C08,0x5295,0x6546,0x5E9D,0x2830,0xA137,0x0F0A,0xB52F,
    0x090E,0x3624,0x9B1B,0x3DDF,0x26CD,0x694E,0xCD7F,0x9FEA,
    0x1B12,0x9E1D,0x7458,0x2E34,0x2D36,0xB2DC,0xEEB4,0xFB5B,
    0xF6A4,0x4D76,0x61B7,0xCE7D,0x7B52,0x3EDD,0x715E,0x9713,
    0xF5A6,0x68B9,0x0000,0x2CC1,0x6040,0x1FE3,0xC879,0xEDB6,
    0xBED4,0x468D,0xD967,0x4B72,0xDE94,0xD498,0xE8B0,0x4A85,
    0x6BBB,0x2AC5,0xE54F,0x16ED,0xC586,0xD79A,0x5566,0x9411,
    0xCF8A,0x10E9,0x0604,0x81FE,0xF0A0,0x4478,0xBA25,0xE34B,
    0xF3A2,0xFE5D,0xC080,0x8A05,0xAD3F,0xBC21,0x4870,0x04F1,
    0xDF63,0xC177,0x75AF,0x6342,0x3020,0x1AE5,0x0EFD,0x6DBF,
    0x4C81,0x1418,0x3526,0x2FC3,0xE1BE,0xA235,0xCC88,0x392E,
    0x5793,0xF255,0x82FC,0x477A,0xACC8,0xE7BA,0x2B32,0x95E6,
    0xA0C0,0x9819,0xD19E,0x7FA3,0x6644,0x7E54,0xAB3B,0x830B,
    0xCA8C,0x29C7,0xD36B,0x3C28,0x79A7,0xE2BC,0x1D16,0x76AD,
    0x3BDB,0x5664,0x4E74,0x1E14,0xDB92,0x0A0C,0x6C48,0xE4B8,
    0x5D9F,0x6EBD,0xEF43,0xA6C4,0xA839,0xA431,0x37D3,0x8BF2,
    0x32D5,0x438B,0x596E,0xB7DA,0x8C01,0x64B1,0xD29C,0xE049,
    0xB4D8,0xFAAC,0x07F3,0x25CF,0xAFCA,0x8EF4,0xE947,0x1810,
    0xD56F,0x88F0,0x6F4A,0x725C,0x2438,0xF157,0xC773,0x5197,
    0x23CB,0x7CA1,0x9CE8,0x213E,0xDD96,0xDC61,0x860D,0x850F,
    0x90E0,0x427C,0xC471,0xAACC,0xD890,0x0506,0x01F7,0x121C,
    0xA3C2,0x5F6A,0xF9AE,0xD069,0x9117,0x5899,0x273A,0xB927,
    0x38D9,0x13EB,0xB32B,0x3322,0xBBD2,0x70A9,0x8907,0xA733,
    0xB62D,0x223C,0x9215,0x20C9,0x4987,0xFFAA,0x7850,0x7AA5,
    0x8F03,0xF859,0x8009,0x171A,0xDA65,0x31D7,0xC684,0xB8D0,
    0xC382,0xB029,0x775A,0x111E,0xCB7B,0xFCA8,0xD66D,0x3A2C,
  }
};
```

12.5.2.5.3 Phase 1 Definition

The inputs to Phase 1 of the temporal key mixing function shall be a temporal key (*TK*), the TA, and the TSC. The temporal key shall be 128 bits in length. Only the 32 MSBs of the TSC and all of the temporal key are used in Phase 1. The output, *TTAK*, shall be 80 bits in length and is represented by an array of 16-bit values: $TTAK_0$ $TTAK_1$ $TTAK_2$ $TTAK_3$ $TTAK_4$.

The description of the Phase 1 algorithm treats all of the following values as arrays of 8-bit values: $TA_0..TA_5$, $TK_0..TK_{15}$. The *TA* octet order is represented according to the conventions from 9.2.2, and the first 3 octets represent the OUI.

The XOR operation, the bitwise-and (&) operation, and the addition (+) operation are used in the Phase 1 specification. A loop counter, *i*, and an array index temporary variable, *j*, are also employed.

One function, *Mk*16, is used in the definition of Phase 1. The function *Mk*16 constructs a 16-bit value from two 8-bit inputs as $Mk16(X,Y) = (256 \cdot X)+Y$.

Two steps make up the Phase 1 algorithm. The first step initializes *TTAK* from *TSC* and *TA*. The second step uses an S-box to iteratively mix the keying material into the 80-bit *TTAK*. The second step sets the PHASE1_LOOP_COUNT to 8. See Figure 12-14.

Input: transmit address *TA*0…*TA*5, Temporal Key *TK*0..*TK*15, and *TSC*0..*TSC*5
Output: intermediate key *TTAK*0..*TTAK*4
PHASE1-KEY-MIXING(*TA*0…*TA*5, *TK*0..*TK*15, *TSC*0..*TSC*5)
    PHASE1_STEP1:
    *TTAK*0 ← *MK16(TSC3, TSC2)*
    *TTAK*1 ← *MK16(TSC5, TSC4)*
    *TTAK*2 ← *Mk*16(*TA*1,*TA*0)
    *TTAK*3 ← *Mk*16(*TA*3,*TA*2)
    *TTAK*4 ← *Mk*16(*TA*5,*TA*4)
    PHASE1_STEP2:
    for *i* = 0 to PHASE1_LOOP_COUNT-1
        j ← 2·(*i* & 1)
        *TTAK*0 ← *TTAK*0 + S[*TTAK*4 ⊕ *Mk*16(*TK*1+*j*,*TK*0+*j*)]
        *TTAK*1 ← *TTAK*1 + S[*TTAK*0 ⊕ *Mk*16(*TK*5+*j*,*TK*4+*j*)]
        *TTAK*2 ← *TTAK*2 + S[*TTAK*1 ⊕ *Mk*16(*TK*9+*j*,*TK*8+*j*)]
        *TTAK*3 ← *TTAK*3 + S[*TTAK*2 ⊕ *Mk*16(*TK*13+*j*,*TK*12+*j*)]
        *TTAK*4 ← *TTAK*4 + S[*TTAK*3 ⊕ *Mk*16(*TK*1+*j*,*TK*0+*j*)] + *i*

Figure 12-14—Phase 1 key mixing

NOTE 1—The TA is mixed into the temporal key in Phase 1 of the hash function. Implementations might achieve a significant performance improvement by caching the output of Phase 1. The Phase 1 output is the same for $2^{16} = 65\,536$ consecutive frames from the same temporal key and TA. Consider the simple case where a STA communicates only with an AP. The STA performs Phase 1 using its own address, and the *TTAK* is used to protect traffic sent to the AP. The STA performs Phase 1 using the AP address, and it is used to unwrap traffic received from the AP.

NOTE 2—The cached *TTAK* from Phase 1 needs to be updated when the lower 16 bits of the TSC wrap and the upper 32 bits need to be updated.

12.5.2.5.4 Phase 2 definition

The inputs to Phase 2 of the temporal key mixing function shall be the output of Phase 1 (*TTAK*) together with the temporal key and the TSC. The *TTAK* is 80 bits in length. Only the 16 LSBs of the TSC are used in Phase 2. The temporal key is 128 bits. The output is the WEP seed, which is a per-frame key, and is 128 bits in length. The constructed WEP seed has an internal structure compliant with the WEP specification. In other words, the first 24 bits of the WEP seed shall be transmitted in plaintext as the WEP IV. As such, these 24 bits are used to convey lower 16 bits of the TSC from the sender (encryptor) to the receiver (decryptor). The rest of the TSC shall be conveyed in the Extended IV field. The temporal key and *TTAK* values are represented as in Phase 1. The WEP seed is treated as an array of 8-bit values: $WEPSeed_0 \ldots WEPSeed_{15}$. The TSC shall be treated as an array of 8-bit values: $TSC_0\ TSC_1\ TSC_2\ TSC_3\ TSC_4\ TSC_5$.

The pseudocode specifying the Phase 2 mixing function employs one variable: *PPK*, which is 96 bits long. The *PPK* is represented as an array of 16-bit values: $PPK_0..PPK_5$. The pseudocode also employs a loop counter, *i*. As detailed in this subclause, the mapping from the 16-bit *PPK* values to the 8-bit *WEPseed* values is explicitly little endian to match the endian architecture of the most common processors used for this application.

The XOR (⊕) operation, the addition (+) operation, the AND (&) operation, the OR (|) operation, and the right bit shift (>>) operation are used in the specification of Phase 2. See Figure 12-15.

Input: intermediate key *TTAK*0…*TTAK*4, *TK*, and TKIP sequence counter *TSC*
Output: WEP Seed *WEPSeed*0…*WEPSeed*15
PHASE2-KEY-MIXING(*TTAK*0…*TTAK*4, *TK*0…*TK*15, *TSC*0…*TSC*5)
    PHASE2_STEP1:
        *PPK*0 ← *TTAK*0
        *PPK*1 ← *TTAK*1
        *PPK*2 ← *TTAK*2
        *PPK*3 ← *TTAK*3
        *PPK*4 ← *TTAK*4
        *PPK*5 ← *TTAK*4 + Mk16(TSC1, TSC0)
    PHASE2_STEP2:
        *PPK*0 ← *PPK*0 + S[*PPK*5 ⊕ *Mk*16(*TK*1,*TK*0)]
        *PPK*1 ← *PPK*1 + S[*PPK*0 ⊕ *Mk*16(*TK*3,*TK*2)]
        *PPK*2 ← *PPK*2 + S[*PPK*1 ⊕ *Mk*16(*TK*5,*TK*4)]
        *PPK*3 ← *PPK*3 + S[*PPK*2 ⊕ *Mk*16(*TK*7,*TK*6)]
        *PPK*4 ← *PPK*4 + S[*PPK*3 ⊕ *Mk*16(*TK*9,*TK*8)]
        *PPK*5 ← *PPK*5 + S[*PPK*4 ⊕ *Mk*16(*TK*11,*TK*10)]
        *PPK*0 ← *PPK*0 + *RotR*1(*PPK*5 ⊕ *Mk*16(*TK*13,*TK*12))
        *PPK*1 ← *PPK*1 + *RotR*1(*PPK*0 ⊕ *Mk*16(*TK*15,*TK*14))
        *PPK*2 ← *PPK*2 + *RotR*1(*PPK*1)
        *PPK*3 ← *PPK*3 + *RotR*1(*PPK*2)
        *PPK*4 ← *PPK*4 + *RotR*1(*PPK*3)
        *PPK*5 ← *PPK*5 + *RotR*1(*PPK*4)
    PHASE2_STEP3:
        *WEPSeed*0 ← *TSC*1
        *WEPSeed*1 ← (TSC1 | 0x20) & 0x7F
        *WEPSeed*2 ← TSC0
        *WEPSeed*3 ← *Lo*8((*PPK*5 ⊕ *Mk*16(*TK*1,*TK*0)) >> 1)
        for *i* = 0 to 5
            *WEPSeed*4+(2·*i*) ← *Lo*8(*PPKi*)
            *WEPSeed*5+(2·*i*) ← *Hi*8(*PPKi*)
        end
    return *WEPSeed*0…*WEPSeed*15

Figure 12-15—Phase 2 key mixing

The algorithm specification relies on four functions:

— The first function, *Lo*8, references the 8 LSBs of the 16-bit input value.
— The second function, *Hi*8, references the 8 MSBs of the 16-bit value.
— The third function, *RotR*1, rotates its 16-bit argument 1 bit to the right.
— The fourth function, *Mk*16, is already used in Phase 1, defined by *Mk*16(*X*,*Y*) = (256·*X*)+*Y*, and constructs a 16-bit output from two 8-bit inputs.

NOTE—The rotate and addition operations in STEP2 make Phase 2 particularly sensitive to the endian architecture of the processor, although the performance degradation due to running this algorithm on a big endian processor is expected to be minor.

Phase 2 comprises three steps:

— STEP1 makes a copy of *TTAK* and brings in the TSC.
— STEP2 is a 96-bit bijective mixing, employing an S-box.
— STEP3 brings in the last of the temporal key *TK* bits and assigns the 24-bit WEP IV value.

The WEP IV format carries 3 octets. STEP3 of Phase 2 determines the value of each of these three octets. The construction was selected to preclude the use of known ARC4 weak keys. The recipient can reconstruct

the 16 LSBs of the TSC used by the originator by concatenating the third and first octets, ignoring the second octet. The remaining 32 bits of the TSC are obtained from the Extended IV field.

12.5.2.6 TKIP replay protection procedures

TKIP implementations shall use the TSC field to defend against replay attacks by implementing the following rules:

a) Each MPDU shall have a unique TKIP TSC value.

b) Each transmitter shall maintain a single TSC (48-bit counter) for each PTKSA, GTKSA, and STKSA.

c) The TSC shall be implemented as a 48-bit strictly increasing counter, initialized to 1 when the corresponding TKIP temporal key is initialized or refreshed.

d) The WEP IV format carries the 16 LSBs of the 48-bit TSC, as defined by the TKIP mixing function (Phase 2, STEP3). The remainder of the TSC is carried in the Extended IV field.

e) A receiver shall maintain a separate set of TKIP TSC replay counters for each PTKSA, GTKSA, and STKSA.

f) TKIP replay detection takes place after the MIC verification and any reordering required by ack processing. Thus, a receiver shall delay advancing a TKIP TSC replay counter until an MSDU passes the MIC check, to prevent attackers from injecting MPDUs with valid ICVs and TSCs, but invalid MICs.

NOTE—This works because if an attacker modifies the TSC, then the encryption key is modified and hence both the ICV and MIC decrypt incorrectly, causing the received MPDU to be dropped.

g) For each PTKSA, GTKSA, and STKSA, the receiver shall maintain a separate replay counter for each frame priority and shall use the TSC in a received frame to detect replayed frames, subject to the limitations on the number of supported replay counters indicated in the RSN Capabilities field, as described in 9.4.2.25. A replayed frame occurs when the TSC in a received frame is less than or equal to the current replay counter value for the frame's priority. A transmitter shall not reorder TKIP protected frames with different priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder TKIP protected frames within a replay counter, but may reorder TKIP protected frames across replay counters. One possible reason for reordering frames is the IEEE 802.11 MSDU priority.

h) A receiver shall discard any MPDU that is received out of order and shall increment dot11RSNAStatsTKIPReplays for this key.

i) For MSDUs sent using the block ack feature, reordering of received MSDUs according to the block ack receiver operation (described in 10.24.4) is performed prior to replay detection.

12.5.3 CTR with CBC-MAC protocol (CCMP)

12.5.3.1 General

Subclause 12.5.3 specifies all variants of CCMP, which provides data confidentiality, authentication, integrity, and replay protection. A non-DMG RSNA STA shall support CCMP-128.

CCMP is based on the CCM of the AES encryption algorithm. CCM combines CTR for data confidentiality and CBC-MAC for authentication and integrity. CCM protects the integrity of both the MPDU Data field and selected portions of the IEEE 802.11 MPDU header.

The AES algorithm is defined in FIPS PUB 197. AES processing used within CCMP uses AES with either a 128-bit key (CCMP-128) or a 256-bit key (CCMP-256).

CCM is defined in IETF RFC 3610. CCM is a generic mode that can be used with any block-oriented encryption algorithm. CCM has two parameters (*M* and *L*).

CCMP-128 uses the following values for the CCM parameters:

— *M* = 8; indicating that the MIC is 8 octets.

— *L* = 2; indicating that the Length field is 2 octets, which is sufficient to hold the length of the largest possible IEEE 802.11 MPDU, expressed in octets.

CCMP-256 uses the following values for the CCM parameters:

— *M* = 16; indicating that the MIC is 16 octets.

— *L* = 2; indicating that the Length field is 2 octets, which is sufficient to hold the length of the largest possible IEEE 802.11 MPDU, expressed in octets.

CCM requires a fresh temporal key for every session. CCM also requires a unique nonce value for each frame protected by a given temporal key, and CCMP uses a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees.

Annex J provides a test vector for CCM.

When CCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust Management frames and the group addressed Management frames that receive "Group Addressed Privacy" as indicated in Table 9-47 shall be protected with CCMP.

12.5.3.2 CCMP MPDU format

Figure 12-16 depicts the MPDU when using CCMP.

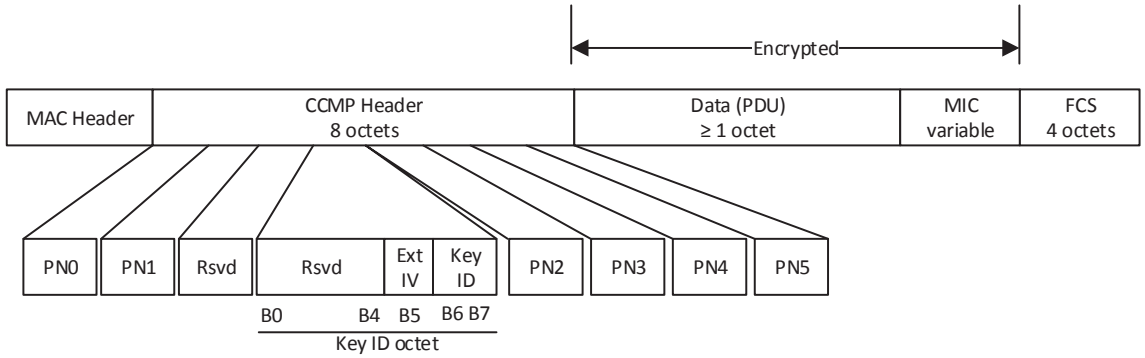

Figure 12-16—Expanded CCMP MPDU

CCMP-128 processing expands the original MPDU size by 16 octets, 8 octets for the CCMP Header field and 8 octets for the MIC field. CCMP-256 processing expands the original MPDU size by 24 octets, 8 octets for the CCMP Header field, and 16 octets for the MIC field. The CCMP Header field is constructed from the PN, ExtIV, and Key ID subfields. PN is a 48-bit PN represented as an array of 6 octets. PN5 is the most significant octet of the PN, and PN0 is the least significant. Note that CCMP does not use the WEP ICV.

The ExtIV subfield (bit 5) of the Key ID octet signals that the CCMP Header field extends the MPDU header by a total of 8 octets, compared to the 4 octets added to the MPDU header when WEP is used. The ExtIV bit (bit 5) is always set to 1 for CCMP.

Bits 6–7 of the Key ID octet are for the Key ID subfield.

12.5.3.3 CCMP cryptographic encapsulation

12.5.3.3.1 General

The CCMP cryptographic encapsulation process is depicted in Figure 12-17.

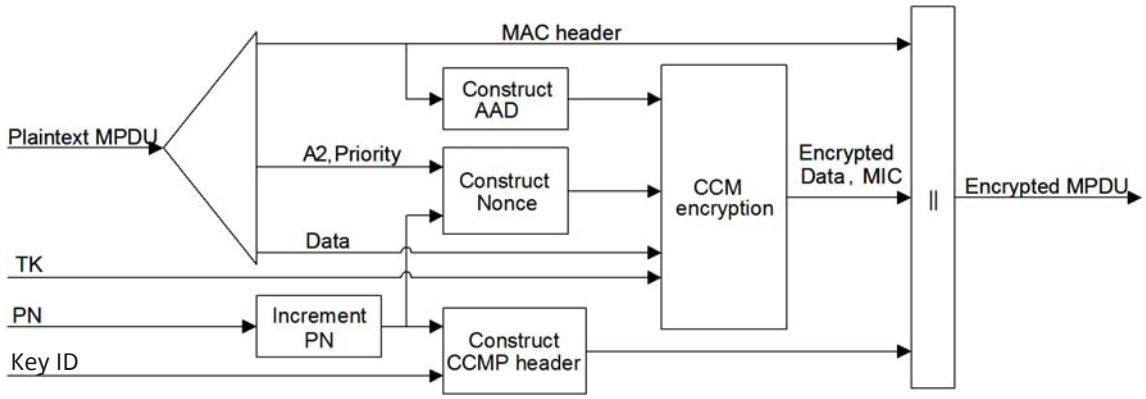

Figure 12-17—CCMP encapsulation block diagram

CCMP encrypts the Frame Body field of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

a) Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key. Note that retransmitted MPDUs are not modified on retransmission.

b) Use the fields in the MPDU header to construct the additional authentication data (AAD) for CCM. The CCM algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are muted by being masked to 0 when calculating the AAD.

c) Construct the CCM Nonce block from the PN, A2, and the priority value of the MPDU where A2 is MPDU Address 2. If the Type field of the Frame Control field is 10 (Data frame) and there is a QoS Control field present in the MPDU header, the priority value of the MPDU is equal to the value of the QC field TID (bits 0 to 3 of the QC field). If the Type field of the Frame Control field is 00 (Management frame), and the frame is a QMF, the priority value of the MPDU is equal to the value in the ACI subfield of the Sequence Number field. Otherwise, the priority value of the MPDU is equal to the fixed value 0.

d) Place the new PN and the key identifier into the 8-octet CCMP header.

e) Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as CCM originator processing.

f) Form the encrypted MPDU by combining the original MPDU header, the CCMP header, the encrypted data and MIC, as described in 12.5.3.2.

The CCM reference describes the processing of the key, nonce, AAD, and data to produce the encrypted output. See 12.5.3.3.2 to 12.5.3.3.6 for details of the creation of the AAD and nonce from the MPDU and the associated MPDU-specific processing.

12.5.3.3.2 PN processing

The PN is incremented by a positive number for each MPDU. The PN shall be incremented in steps of 1 for constituent MPDUs of fragmented MSDUs and MMPDUs. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key.

NOTE—When a group addressed MSDU is retransmitted using GCR, it is concealed from non-GCR capable STAs using the procedures described in 11.24.16.3.5. The MPDU containing this concealed A-MSDU has a different PN from the MPDU that contained the original transmission of the group addressed MSDU.

12.5.3.3.3 Construct AAD

The format of the AAD is shown in Figure 12-18.

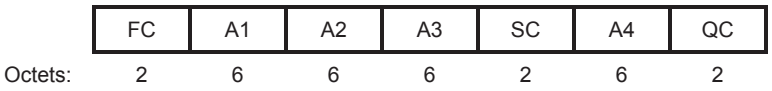

Figure 12-18—AAD construction

The length of the AAD varies depending on the presence or absence of the QC and A4 fields and is shown in Table 12-1.

Table 12-1—AAD length

QC field	A4 field	AAD length (octets)
Absent	Absent	22
Present	Absent	24
Absent	Present	28
Present	Present	30

The AAD is constructed from the MPDU header. The AAD does not include the header Duration field, because the Duration field value might change due to normal IEEE 802.11 operation (e.g., a rate change during retransmission). The AAD includes neither the Duration/ID field nor the HT Control field because the contents of these fields might change during normal operation (e.g., due to a rate change preceding retransmission). The HT Control field might also be inserted or removed during normal operation (e.g., retransmission of an A-MPDU where the original A-MPDU included an MRQ that has already generated a response). For similar reasons, several subfields in the Frame Control field are masked to 0. AAD construction is performed as follows:

a) FC – MPDU Frame Control field, with
   1) Subtype subfield (bits 4 5 6) in a Data frame masked to 0
   2) Retry subfield (bit 11) masked to 0
   3) Power Management subfield (bit 12) masked to 0
   4) More Data subfield (bit 13) masked to 0
   5) Protected Frame subfield (bit 14) always set to 1
   6) +HTC/Order subfield (bit 15) as follows:
      i) Masked to 0 in all Data frames containing a QoS Control field
      ii) Unmasked otherwise
b) A1 – MPDU Address 1 field.
c) A2 – MPDU Address 2 field.
d) A3 – MPDU Address 3 field.

e) SC – MPDU Sequence Control field, with the Sequence Number subfield (bits 4–15 of the Sequence Control field) masked to 0. The Fragment Number subfield is not modified.

f) A4 – MPDU Address field, if present.

g) QC – QoS Control field, if present, a 2-octet field that includes the MSDU priority. The QC TID is used in the construction of the AAD. When in a non-DMG BSS and both the STA and its peer have their SPP A-MSDU Capable fields equal to 1, bit 7 (the A-MSDU Present field) is used in the construction of the AAD. The remaining QC fields are masked to 0 for the AAD calculation (bits 4 to 6, bits 8 to 15, and bit 7 when either the STA or its peer has the SPP A-MSDU Capable field equal to 0). When in a DMG BSS, the A-MSDU Present bit 7 and A-MSDU Type bit 8 are used in the construction of the AAD, and the remaining QC fields are masked to 0 for the AAD calculation (bits 4 to 6, bits 9 to 15).

12.5.3.3.4 Construct CCM nonce

The Nonce field occupies 13 octets, and its structure is shown in Figure 12-19. The structure of the Nonce Flags subfield of the Nonce field is shown in Figure 12-20.

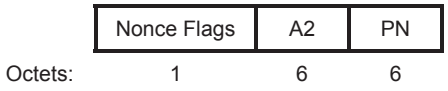

Figure 12-19—Nonce construction

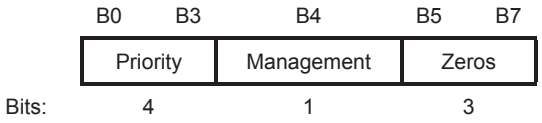

Figure 12-20—Nonce Flags subfield

The Nonce field has an internal structure of Nonce Flags || A2 || PN, where

— The Priority subfield of the Nonce Flags field shall be set to the priority value of the MPDU.

— When management frame protection is negotiated, the Management field of the Nonce Flags field shall be set to 1 if the Type field of the Frame Control field is 00 (Management frame); otherwise it shall be set to 0.

— Bits 5 to 7 of the Nonce Flags field shall be set to 0.

— MPDU address A2 field occupies octets 1–6. This shall be encoded with the octets ordered with A2 octet 0 at octet index 1 and A2 octet 5 at octet index 6.

— The PN field occupies octets 7–12. The octets of PN shall be ordered so that PN0 is at octet index 12 and PN5 is at octet index 7.

12.5.3.3.5 Construct CCMP header

The format of the 8-octet CCMP header is given in 12.5.3.2. The header encodes the PN, Key ID, and ExtIV field values used to encrypt the MPDU.

12.5.3.3.6 CCM originator processing

CCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, CCM is used with the AES block cipher.

There are four inputs to CCM originator processing:

a) *Key:* the temporal key (16 octets).
b) *Nonce:* the nonce (13 octets) constructed as described in 12.5.3.3.4.
c) *Frame body:* the plaintext frame body of the MPDU.
d) *AAD:* the AAD (22–30 octets) constructed from the MPDU header as described in 12.5.3.3.3.

The CCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the CCM originator processing consists of the encrypted data and an encrypted MIC (see Figure 12-16).

The PN values sequentially number each MPDU. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA. The PN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding temporal key is initialized or refreshed.

A transmitter shall not use an IEEE 802.11 MSDU or A-MSDU priority if this would cause the total number of priorities used during the lifetime of the SA to exceed the number of replay counters supported by the receiver (for a pairwise SA) or all the receivers (for a group SA) for that SA. The transmitter shall not reorder CCMP protected frames that are transmitted to the same RA within a replay counter, but may reorder frames across replay counters. One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority.

The transmitter shall preserve the order of protected robust Management frames that are transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust IQMFs within an AC when the frames are transmitted to the same RA.

A CCMP protected individually addressed robust Management frame shall be protected using the same TK as a Data frame.

12.5.3.4 CCMP decapsulation

12.5.3.4.1 General

Figure 12-21 depicts the CCMP decapsulation process.

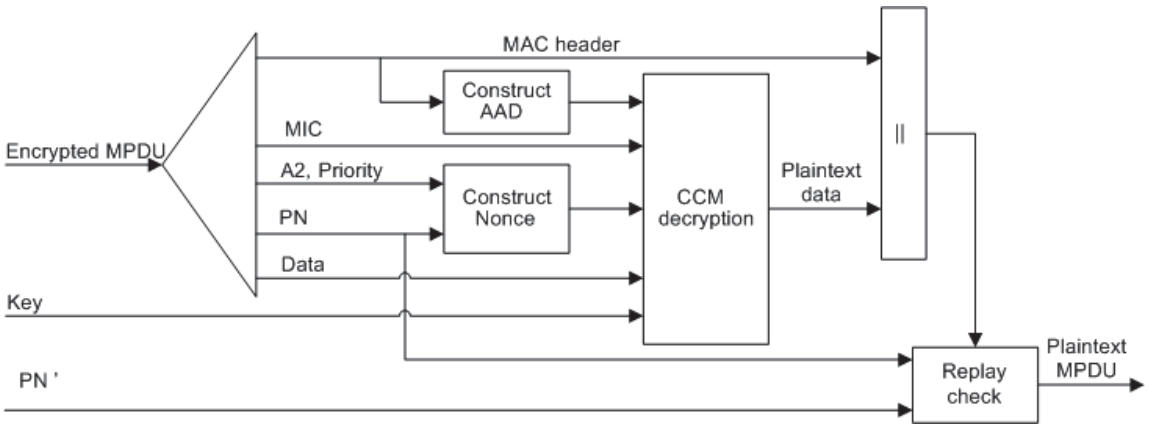

Figure 12-21—CCMP decapsulation block diagram

CCMP decrypts the Frame Body field of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

a) The encrypted MPDU is parsed to construct the AAD and nonce values.
b) The AAD is formed from the MPDU header of the encrypted MPDU.
c) The Nonce value is constructed from the A2, PN, and Nonce Flags fields.
d) The MIC is extracted for use in the CCM integrity checking.
e) The CCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data.
f) The received MPDU header and the MPDU plaintext data from the CCM recipient processing are concatenated to form a plaintext MPDU.
g) The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session.

See 12.5.3.4.2 to 12.5.3.4.4 for details of this processing.

When the received frame is a CCMP protected individually addressed robust Management frame, contents of the MMPDU body after protection is removed shall be delivered to the SME via the MLME primitive designated for that MMPDU rather than through the MA-UNITDATA.indication primitive.

12.5.3.4.2 CCM recipient processing

CCM recipient processing uses the same parameters as CCM originator processing. A CCMP protected individually addressed robust Management frame shall use the same TK as a Data frame.

There are four inputs to CCM recipient processing:

— *Key:* the temporal key (16 octets).
— *Nonce:* the nonce (13 octets) constructed as described in 12.5.3.3.4.
— *Encrypted frame body:* the encrypted frame body from the received MPDU. The encrypted frame body includes the MIC.
— *AAD:* the AAD (22–30 octets) that is the canonical MPDU header as described in 12.5.3.3.3.

The CCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful.

There is one output from error-free CCM recipient processing:

— *Frame body:* the plaintext frame body, which is 8 octets (CCMP-128) or 16 octets (CCMP-256) smaller than the encrypted frame body.

12.5.3.4.3 Decrypted CCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful CCM recipient processing to create the plaintext MPDU.

12.5.3.4.4 PN and replay detection

To effect replay detection, the receiver extracts the PN from the CCMP header. See 12.5.3.2 for a description of how the PN is encoded in the CCMP header. The following processing rules are used to detect replay:

a) The receiver shall maintain a separate set of replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted CCMP MPDUs.

b) For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each TID, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 9.4.2.25), and shall use the PN from a received frame to detect replayed frames. A replayed frame occurs when the PN from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type.

c) If dot11RSNAProtectedManagementFramesActivated is true, the recipient shall maintain a single replay counter for received individually addressed robust Management frames that are received with the To DS subfield equal to 0 and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received individually addressed robust Management frames that are received with the To DS subfield equal to 1. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current value of the management frame replay counter that corresponds to the ACI of the frame.

d) The receiver shall discard any Data frame that is received with its PN less than or equal to the value of the replay counter that is associated with the TA and priority value of the received MPDU. The receiver shall discard MSDUs and MMPDUs whose constituent MPDU PN values are not incrementing in steps of 1. If dot11RSNAProtectedManagementFramesActivated is true, the receiver shall discard any individually addressed robust Management frame that is received with its PN less than or equal to the value of the replay counter associated with the TA of that individually addressed Management frame.

e) When discarding a frame, the receiver shall increment by 1 dot11RSNAStatsCCMPReplays for Data frames or dot11RSNAStatsRobustMgmtCCMPReplays for robust Management frames.

f) For MSDUs or A-MSDUs sent using the block ack feature, reordering of received MSDUs or A-MSDUs according to the block ack receiver operation (described in 10.24.4) is performed prior to replay detection.

12.5.4 Broadcast/multicast integrity protocol (BIP)

12.5.4.1 BIP overview

BIP provides data integrity and replay protection for group addressed robust Management frames after successful establishment of an IGTKSA (see 12.6.1.1.9).

BIP-CMAC-128 provides data integrity and replay protection, using AES-128 in CMAC Mode with a 128-bit integrity key and a CMAC TLen value of 128 (16 octets). BIP-CMAC-256 provides data integrity and replay protection, using AES-256 in CMAC Mode with a 256-bit integrity key and a CMAC TLen value of 128 (16 octets). NIST Special Publication 800-38B defines the CMAC algorithm, and NIST SP 800-38D defines the GMAC algorithm. BIP processing uses AES with a 128-bit or 256-bit integrity key and a CMAC TLen value of 128 (16 octets). The CMAC output for BIP-CMAC-256 is not truncated and shall be 128 bits (16 octets). The CMAC output for BIP-CMAC-128 is truncated to 64 bits:

MIC = Truncate-64(CMAC Output).

BIP-GCMP-128 uses AES with a 128-bit integrity key, and BIP-GCMP-256 uses AES with a 256-bit integrity key. The authentication tag for both BIP-GCMP-128 and BIP-GCMP-256 is not truncated and shall be 128 bits (16 octets).

BIP uses the IGTK to compute the MMPDU MIC. The authenticator shall distribute one new IGTK and IGTK PN (IPN) whenever it distributes a new GTK. The IGTK is identified by the MAC address of the transmitting STA plus an IGTK identifier that is encoded in the MME Key ID field.

12.5.4.2 BIP MMPDU format

The Management MIC element shall follow all of the other elements in the management frame body but precede the FCS. See 9.4.2.55 for the format of the Management MIC element. Figure 12-22 shows the BIP MMPDU.

Figure 12-22—BIP Encapsulation

12.5.4.3 BIP AAD construction

The BIP Additional Authentication Data (AAD) shall be constructed from the MPDU header. The Duration field in the AAD shall be masked to 0. The AAD construction shall use a copy of the IEEE 802.11 header without the SC field for the MPDU, with the following exceptions:

a) FC—MPDU Frame Control field, with:
   1) Retry subfield (bit 11) masked to 0
   2) Power Management subfield (bit 12) masked to 0
   3) More Data subfield (bit 13) masked to 0
b) A1—MPDU Address 1 field.
c) A2—MPDU Address 2 field.
d) A3—MPDU Address 3 field.

Figure 12-23 depicts the format of the AAD. The length of the AAD is 20 octets.

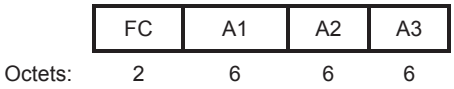

Figure 12-23—BIP AAD Construction

12.5.4.4 BIP replay protection

The MME Sequence Number field represents a sequence number whose length is 6 octets.

When management frame protection is negotiated, the receiver shall maintain a 48-bit replay counter for each IGTK. The receiver shall set the receive replay counter to the value of the IPN in the IGTK key data encapsulation (KDE) (see 12.7.2) provided by the Authenticator in either the 4-way handshake, FT 4-way handshake, FT handshake, or group key handshake. The transmitter shall maintain a single IPN for each IGTK. The IPN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding IGTK is initialized. The transmitter may reinitialize the sequence counter when the IGTK is refreshed. See 12.5.4.5 and 12.5.4.6 for per packet BIP processing.

NOTE—When the IPN space is exhausted, the choices available to an implementation are to replace the IGTK or to end communications.

When dot11QMFActivated is true, the receiver shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use QMF. The receiver shall use the ACI encoded in the Sequence Number field of received GQMFs protected by BIP to select the replay counter to use for the received frame, and shall use the IPN from the received frame to detect replays.

If dot11RSNAProtectedManagementFramesActivated is true and dot11MeshSecurityActivated is true, the recipient shall maintain a single replay counter for received group addressed robust Management frames that do not use the QMF service and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received group addressed robust Management frames that use the QMF service. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the value of the management frame replay counter that corresponds to the ACI of the frame. The transmitter shall preserve the order of protected robust Management frames transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust GQMFs within an AC when the frames are transmitted to the same RA.

12.5.4.5 BIP transmission

When a STA transmits a protected group addressed robust Management frame, it shall

a) Select the IGTK currently active for transmission of frames to the intended group of recipients and construct the MME (see 9.4.2.55) with the MIC field masked to 0 and the Key ID field set to the corresponding IGTK Key ID value. If the frame is not a GQMF, the transmitting STA shall insert a strictly increasing integer into the MME IPN field. If the frame is a GQMF, then the transmitting STA shall maintain a 48-bit counter for use as the IPN, the counter shall be incremented for each GQMF until the two least significant bits of the counter match the ACI of the AC that is used to transmit the frame, and the counter value shall be inserted into the MME IPN field of the frame. For BIP-GMAC-128 and BIP-GMAC-256, the initialization vector passed to GMAC shall be a concatenation of Address 2 from the MAC header of the MPDU and the non-negative integer inserted into the MMP IPN field.

b) Compute AAD as specified in 12.5.4.3.

c) Compute an integrity value over the concatenation of AAD and the management frame body including MME, and insert the output into the MME MIC field. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC.

d) Compose the frame as the IEEE 802.11 header, management frame body, including MME, and FCS. The MME shall appear last in the frame body.

e) Transmit the frame.

12.5.4.6 BIP reception

When a STA with management frame protection negotiated receives a group addressed robust Management frame protected by BIP-CMAC-128, BIP-CMAC-256, BIP-GMAC-128, or BIP-GMAC-256, it shall

a) Identify the appropriate IGTK and associated state based on the MME Key ID field. If no such IGTK exists, silently drop the frame and terminate BIP processing for this reception.

b) Perform replay protection on the received frame. The receiver shall interpret the MME IPN field as a 48-bit unsigned integer.

   1) If the frame is not a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field. If the integer value from the received MME IPN field is less than or equal to the replay counter value for this IGTK, the receiver shall discard the frame and increment the dot11RSNAStatsCMACReplays counter by 1.

2) If the frame is a GQMF, the receiver shall compare this MME IPN integer value to the value of the receive replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame. If the integer value from the received MME IPN field is less than or equal to the replay counter value for this IGTK and AC, the receiver shall discard the frame and increment the dot11RSNAStatsCMACReplays counter by 1.

c) Compute AAD for this Management frame, as specified in 12.5.4.3. For BIP-GMAC-128 and BIP-GMAC-256, an initialization vector for GMAC is constructed as the concatenation of Address 2 from the MAC header of the MPDU and the 48-bit unsigned integer from the MME IPN field.

d) Extract and save the received MIC value, and compute a verifier over the concatenation of AAD, the management frame body, and MME, with the MIC field masked to 0 in the MME. For BIP-CMAC-128, the integrity value is 64 bits and is computed using AES-128-CMAC; for BIP-CMAC-256, the integrity value is 128 bits and is computed using AES-256-CMAC; for BIP-GMAC-128, the integrity value is 128 bits and is computed using AES-128-GMAC; and, for BIP-GMAC-256, the integrity value is 128 bits and is computed using AES-256-GMAC. If the result does not match the received MIC value, then the receiver shall discard the frame, increment the dot11RSNAStatsBIPMICErrors counter by 1, and terminate BIP processing for this reception.

e) If the frame is not a GQMF, update the replay counter for the IGTK identified by the MME Key ID field with the integer value of the MME IPN field.

f) If the frame is a GQMF, update the replay counter for the IGTK identified by the MME Key ID field and the AC represented by the value of the ACI subfield of the received frame with the integer value of the MME IPN field.

If management frame protection is negotiated, group addressed robust Management frames that are received without BIP protection shall be discarded.

12.5.5 GCM protocol (GCMP)

12.5.5.1 GCMP overview

Subclause 12.5.5 specifies the GCMP, which provides data confidentiality, authentication, integrity, and replay protection. A DMG RSNA STA shall support GCMP-128.

GCMP is based on the GCM of the AES encryption algorithm. GCM protects the integrity of both the MPDU Data field and selected portions of the MPDU header.

The AES algorithm is defined in FIPS PUB 197. All AES processing used within GCMP uses AES with a 128-bit key (GCMP-128) or a 256-bit key (GCMP-256).

GCM is defined in NIST Special Publication 800-38D. GCM is a generic mode that can be used with any block-oriented encryption algorithm.

GCM requires a fresh temporal key for every session. GCM also requires a unique nonce value for each frame protected by a given temporal key, and GCMP uses a 96-bit nonce that includes a 48-bit packet number (PN) for this purpose. Reuse of a PN with the same temporal key voids all security guarantees. GCMP uses a 128-bit MIC.

Annex J provides test vectors for GCM.

When GCMP is selected as the RSN pairwise cipher and management frame protection is negotiated, individually addressed robust Management frames and the group addressed Management frames that receive “Group Addressed Privacy” as indicated in Table 9-47 shall be protected with GCMP.

12.5.5.2 GCMP MPDU format

Figure 12-24 shows the MPDU format when using GCMP.

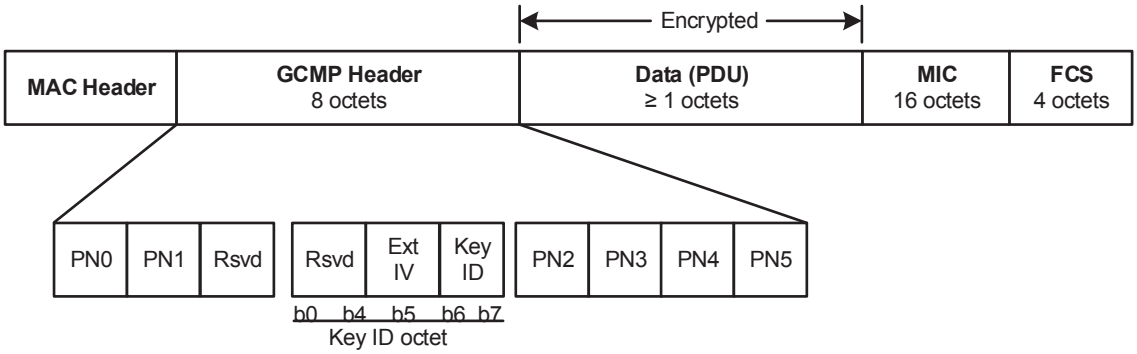

Figure 12-24—Expanded GCMP MPDU

GCMP processing expands the original MPDU size by 24 octets, 8 octets for the GCMP Header field and 16 octets for the MIC field. The GCMP Header field is constructed from the PN and Key ID subfields. The 48-bit PN is represented as an array of 6 octets. PN5 is the most significant octet of the PN, and PN0 is the least significant.

The ExtIV subfield (bit 5) of the Key ID octet is always set to 1 for GCMP.

Bits 6–7 of the Key ID octet are for the Key ID subfield. The remaining bits of the Key ID octet are reserved.

12.5.5.3 GCMP cryptographic encapsulation

12.5.5.3.1 General

The GCMP cryptographic encapsulation process is depicted in Figure 12-25.

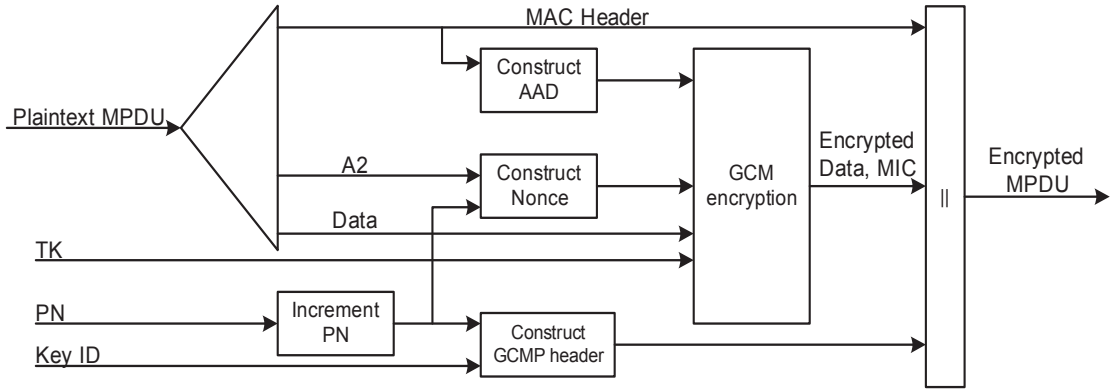

Figure 12-25—GCMP encapsulation block diagram

GCMP encrypts the Frame Body field of a plaintext MPDU and encapsulates the resulting cipher text using the following steps:

a) Increment the PN, to obtain a fresh PN for each MPDU, so that the PN never repeats for the same temporal key.

NOTE—Retransmitted MPDUs are not modified on retransmission.

b) Use the fields in the MPDU header to construct the additional authentication data (AAD) for GCM. The GCM algorithm provides integrity protection for the fields included in the AAD. MPDU header fields that may change when retransmitted are masked to 0 when calculating the AAD.

c) Construct the GCM Nonce block from the PN and A2, where A2 is MPDU Address 2.

d) Place the new PN and the key identifier into the 8-octet GCMP Header.

e) Use the temporal key, AAD, nonce, and MPDU data to form the cipher text and MIC. This step is known as GCM originator processing.

f) Form the encrypted MPDU by combining the original MPDU header, the GCMP header, the encrypted data and MIC, as described in 12.5.5.2.

The GCM reference describes the processing of the key, nonce, AAD, and data to produce the encrypted output. See 12.5.5.3.2 to 12.5.5.3.6 for details of the creation of the AAD and nonce from the MPDU and the associated MPDU-specific processing.

12.5.5.3.2 PN processing

The PN is incremented by a positive number for each MPDU. The PN shall be incremented in steps of 1 for constituent MPDUs of fragmented MSDUs and MMPDUs. The PN shall never repeat for a series of encrypted MPDUs using the same temporal key.

If the PN is larger than dot11PNExhaustionThreshold, an MLME-PN-EXHAUSTION.indication primitive shall be generated.

NOTE—When a group addressed MSDU is retransmitted using GCR, it is concealed from non-GCR capable STAs using the procedures described in 11.24.16.3.5. The MPDU containing this concealed A-MSDU has a different PN from the MPDU that contained the original transmission of the group addressed MSDU.

12.5.5.3.3 Construct AAD

The AAD construction is as defined in 12.5.3.3.3.

12.5.5.3.4 Construct GCM nonce

The Nonce field occupies 12 octets, and its structure is shown in Figure 12-26.

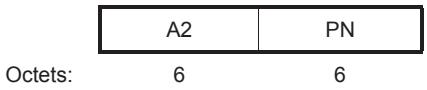

Figure 12-26—Nonce construction

The Nonce field has an internal structure of A2 || PN, where

— MPDU address A2 field occupies octets 0 to 5. This shall be encoded with the octets ordered with A2 octet 0 at octet index 0 and A2 octet 5 at octet index 5.

— The PN field occupies octets 6 to 11. The octets of PN shall be ordered so that PN0 is at octet index 11 and PN5 is at octet index 6.

12.5.5.3.5 Construct GCMP header

The format of the 8-octet GCMP Header is given in 12.5.5.2. The header encodes the PN and Key ID field values used to encrypt the MPDU.

12.5.5.3.6 GCM originator processing

GCM is a generic authenticate-and-encrypt block cipher mode, and in this standard, GCM is used with the AES block cipher.

There are four inputs to GCM originator processing:

a) *Key:* the temporal key (16 octets).
b) *Nonce:* the nonce (12 octets) constructed as described in 12.5.5.3.4.
c) *Frame body:* the plaintext frame body of the MPDU.
d) *AAD:* the AAD (22-30 octets) constructed from the MPDU header as described in 12.5.5.3.3.

The GCM originator processing provides authentication and integrity of the frame body and the AAD as well as data confidentiality of the frame body. The output from the GCM originator processing consists of the encrypted data and 16 additional octets of encrypted MIC (see Figure 12-24).

The PN values sequentially number each MPDU. Each transmitter shall maintain a single PN (48-bit counter) for each PTKSA, GTKSA, and STKSA. The PN shall be implemented as a 48-bit strictly increasing integer, initialized to 1 when the corresponding temporal key is initialized or refreshed.

A transmitter shall not use IEEE 802.11 MSDU or A-MSDU priorities without ensuring that the receiver supports the required number of replay counters. The transmitter shall not reorder GCMP protected frames that are transmitted to the same RA within a replay counter, but may reorder frames across replay counters. One possible reason for reordering frames is the IEEE 802.11 MSDU or A-MSDU priority.

The transmitter shall preserve the order of protected robust Management frames that are transmitted to the same DA without the QMF service. When the QMF service is used, the transmitter shall not reorder robust IQMFs within an AC when the frames are transmitted to the same RA.

A GCMP protected individually addressed robust Management frame shall be protected using the same TK as a Data frame.

12.5.5.4 GCMP decapsulation

12.5.5.4.1 General

Figure 12-27 shows the GCMP decapsulation process.

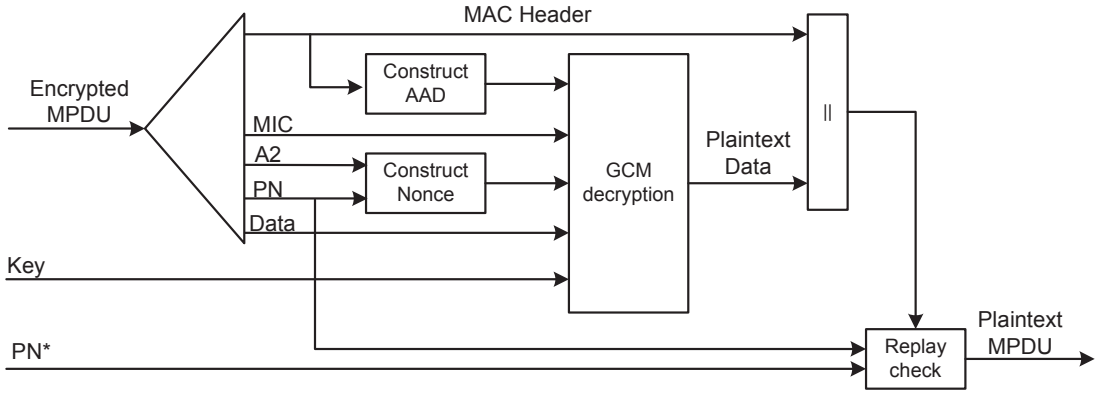

Figure 12-27—GCMP decapsulation block diagram

GCMP decrypts the Frame Body field of a cipher text MPDU and decapsulates a plaintext MPDU using the following steps:

a) The encrypted MPDU is parsed to construct the AAD and nonce values.
b) The AAD is formed from the MPDU header of the encrypted MPDU.
c) The Nonce value is constructed from the A2 and PN fields.
d) The MIC is extracted for use in the GCM integrity checking.
e) The GCM recipient processing uses the temporal key, AAD, nonce, MIC, and MPDU cipher text data to recover the MPDU plaintext data as well as to check the integrity of the AAD and MPDU plaintext data.
f) The received MPDU header and the MPDU plaintext data from the GCM recipient processing are concatenated to form a plaintext MPDU.
g) The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session.

See 12.5.5.4.2 to 12.5.5.4.4 for details of this processing.

When the received frame is a GCMP protected individually addressed robust Management frame, the contents of the MMPDU body after protection is removed and shall be delivered to the SME via the MLME primitive designated for that MMPDU rather than through the MA-UNITDATA.indication primitive.

12.5.5.4.2 GCM recipient processing

GCM recipient processing shall use the same parameters as GCM originator processing. A GCMP protected individually addressed robust Management frame shall use the same TK as a Data frame.

There are four inputs to GCM recipient processing:

— *Key:* the temporal key (16 octets).
— *Nonce:* the nonce (12 octets) constructed as described in 12.5.5.3.4.
— *Encrypted frame body:* the encrypted frame body from the received MPDU. The encrypted frame body includes a 16-octet MIC.
— *AAD*: the AAD (22-30 octets) that is the canonical MPDU header as described in 12.5.5.3.3.

The GCM recipient processing checks the authentication and integrity of the frame body and the AAD as well as decrypting the frame body. The plaintext is returned only if the MIC check is successful.

There is one output from error-free GCM recipient processing:

— *Frame body:* the plaintext frame body, which is 16 octets smaller than the encrypted frame body.

12.5.5.4.3 Decrypted GCMP MPDU

The decapsulation process succeeds when the calculated MIC matches the MIC value obtained from decrypting the received encrypted MPDU. The original MPDU header is concatenated with the plaintext data resulting from the successful GCM recipient processing to create the plaintext MPDU.

12.5.5.4.4 PN and replay detection

To effect replay detection, the receiver extracts the PN from the GCMP header. See 12.5.5.2 for a description of how the PN is encoded in the GCMP header. The following processing rules are used to detect replay:

a) The receiver shall maintain a separate set of replay counters for each PTKSA, GTKSA, and STKSA. The receiver initializes these replay counters to 0 when it resets the temporal key for a peer. The replay counter is set to the PN value of accepted GCMP MPDUs.

b) For each PTKSA, GTKSA, and STKSA, the recipient shall maintain a separate replay counter for each TID, subject to the limitation of the number of supported replay counters indicated in the RSN Capabilities field (see 9.4.2.25), and shall use the PN from a received frame to detect replayed frames. A replayed frame occurs when the PN from a received frame is less than or equal to the current replay counter value for the frame's MSDU or A-MSDU priority and frame type.

c) If dot11RSNAProtectedManagementFramesActivated is true, the recipient shall maintain a single replay counter for received individually addressed robust Management frames that are received with the To DS subfield equal to 0 and shall use the PN from the received frame to detect replays. If dot11QMFActivated is also true, the recipient shall maintain an additional replay counter for each ACI for received individually addressed robust Management frames that are received with the To DS subfield equal to 1. The QMF receiver shall use the ACI encoded in the Sequence Number field of the received frame to select the replay counter to use for the received frame, and shall use the PN from the received frame to detect replays. A replayed frame occurs when the PN from the frame is less than or equal to the current value of the management frame replay counter that corresponds to the ACI of the frame.

d) The receiver shall discard any Data frame that is received with its PN less than or equal to the value of the replay counter that is associated with the TA and priority value of the received MPDU. The receiver shall discard MSDUs and MMPDUs whose constituent MPDU PN values are not incrementing in steps of 1. If dot11RSNAProtectedManagementFramesActivated is true, the receiver shall discard any individually addressed robust Management frame that is received with its PN less than or equal to the value of the replay counter associated with the TA of that individually addressed Management frame.

e) When discarding a frame, the receiver shall increment by 1 dot11RSNAStatsGCMPReplays for Data frames or dot11RSNAStatsRobustMgmtGCMPReplays for robust Management frames.

f) For MSDUs or A-MSDUs sent using the block ack feature, reordering of received MSDUs or A-MSDUs according to the block ack receiver operation (described in 10.24.4) is performed prior to replay detection.

12.6 RSNA security association management

12.6.1 Security associations

12.6.1.1 Security association definitions

12.6.1.1.1 General

IEEE Std 802.11 uses the notion of a security association to describe secure operation. Secure communications are possible only within the context of a security association, as this is the context providing the state—cryptographic keys, counters, sequence spaces, etc.—needed for correct operation of the IEEE 802.11 cipher suites.

A security association is a set of policy(ies) and key(s) used to protect information. The information in the security association is stored by each party of the security association, needs to be consistent among all parties, and needs to have an identity. The identity is a compact name of the key and other bits of security association information to fit into a table index or an MPDU. The following types of security associations are supported by an RSNA STA:

— PMKSA: A result of a successful IEEE 802.1X exchange, SAE authentication, or preshared PMK information. A PMKSA can be cached.

- PMK-R0 security association: A result of a successful FT initial mobility domain association.
- PMK-R1 security association: A result of a successful FT initial mobility domain association or FT authentication sequence.
- Mesh PMKSA: A result of successful completion of the active authentication protocol.
- PTKSA: A result of a successful 4-way handshake, FT 4-way handshake, or FT authentication sequence.
- Mesh TKSA: A result of a successful authenticated mesh peering exchange (AMPE).
- GTKSA: A result of a successful group key handshake, 4-way handshake, FT 4-way handshake, or FT authentication sequence.
- IGTKSA: A result of a successful group key handshake, successful 4-way handshake, FT 4-way handshake, or the Reassociation Response frame of the fast BSS transition protocol.
- Mesh GTKSA: A result of a successful AMPE or mesh group key handshake.
- SMKSA: A result of a successful initial SMK handshake.
- STKSA: A result of a successful 4-way STK handshake following the initial SMK handshake or subsequent rekeying.

In order to set up a security association to a peer STA, a SME that does not know the security policy of the peer should send a Probe Request frame to the peer STA to find its security policy before setting up a security association to the peer STA.

In order to set up a security association to a peer STA, a STA that received a 4-way handshake but does not know the security policy of the peer should send a Probe Request frame to the peer STA to find its security policy before setting up a security association to the peer STA.

12.6.1.1.2 PMKSA

When the PMKSA is the result of a successful IEEE 802.1X authentication, it is derived from the EAP authentication and authorization parameters provided by the AS. When the PMKSA is the result of a successful SAE authentication, it is generated as a result of the successful completion of the SAE exchange. This security association is bidirectional. In other words, both parties use the information in the security association for both sending and receiving. The PMKSA is created by the Supplicant's SME when the EAP authentication completes successfully or the PSK is configured. The PMKSA is created by the Authenticator's SME when the PMK is created from the keying information transferred from the AS, when IEEE 802.1X authentication is utilized, when the SAE exchange successfully completes, or when the PSK is configured. The PMKSA is used to create the PTKSA. PMKSAs have a certain lifetime. The PMKSA consists of the following:

- PMKID, as defined in 12.7.1.3. The PMKID identifies the security association.
- Authenticator's or peer's MAC address. For multi-band RSNA, the MAC address is associated with the operating band in use when the PMKSA is established.
- PMK.
- Lifetime, as defined in 12.7.1.3.
- AKMP.
- All authorization parameters specified by the AS or local configuration. This might include parameters such as the STA's authorized SSID.

12.6.1.1.3 PMK-R0 security association

The PMK-R0 security association is the result of a successful completion of the IEEE 802.1X authentication, SAE authentication, or use of PSK during the FT initial mobility domain association. This security association is bidirectional. It consists of the following:

— SSID
— Mobility domain identifier (MDID)
— PMK-R0
— R0KH-ID
— PMKR0Name
— S0KH-ID
— PMK-R0 lifetime
— Pairwise cipher suite selector
— All authorization parameters specified by the AS or local configuration

12.6.1.1.4 PMK-R1 security association

The PMK-R1 security association is the result of

— A successful completion of the IEEE 802.1X authentication, SAE authentication, or use of PSK during the FT initial mobility domain association or
— A successful completion of the authentication phase in the fast BSS transition to the target AP

This security association is bidirectional. It consists of the following:

— SSID
— MDID
— PMK-R1
— PMK-R1 lifetime
— PMKR1Name
— R1KH-ID
— R0KH-ID
— PMKR0Name
— S0KH-ID
— S1KH-ID
— Pairwise cipher suite selector
— All authorization parameters specified by the AS or local configuration

12.6.1.1.5 Mesh PMKSA

The mesh PMKSA is the result of successful completion of the active authentication protocol. This security association is bidirectional. The two authenticated parties use the information in the security association for both sending and receiving. The mesh PMKSA is created by the Mesh STA's SME when the active authentication protocol completes successfully with the peer mesh STA. The mesh PMKSA is used to create the mesh TKSA. Mesh PMKSAs have a certain lifetime. Mesh PMKSAs contain the following, and are identified by their PMKID.

— PMKID, as defined in 12.4.5.4
— Mesh STA's MAC address
— Peer mesh STA's MAC address
— PMK
— AEK, as defined in 14.5.7
— Lifetime, as defined in 12.7.1.3
— Selected AKM suite (see 9.4.2.25.3)

12.6.1.1.6 PTKSA

The PTKSA is a result of the 4-way handshake, FT 4-way handshake, FT protocol, or FT resource request protocol. This security association is also bidirectional. PTKSAs have the same lifetime as the PMKSA or PMK-R1 security Association, whichever comes first. Because the PTKSA is tied to the PMKSA or to a PMK-R1 security association, it only has the additional information from the 4-way handshake or FT Protocol authentication. For the PTKSA derived as a result of the 4-way handshake, there shall be only one PTKSA per band (see 12.6.19) with the same Supplicant and Authenticator MAC addresses. For the PTKSA derived as a result of an initial mobility domain association or fast BSS transition, there shall be only one PTKSA with the same STA's MAC address and BSSID.

During the 4-way handshake defined in 12.7.6.5 and the FT 4-way handshake defined in 13.4.2, there is state created between message 1 and message 3 of the handshake. This does not create a PTKSA until message 3 is validated by the Supplicant and message 4 is validated by the Authenticator.

During the FT authentication sequence defined in 13.8, the PTKSA is validated when message 3 is validated by the R1KH and message 4 is validated by the S1KH.

The PTKSA consists of the following:

- PTK
- Pairwise cipher suite selector
- Supplicant MAC address or STA's MAC address
- Authenticator MAC address or BSSID
- Key ID
- If FT key hierarchy is used,
  - R1KH-ID
  - S1KH-ID
  - PTKName

12.6.1.1.7 Mesh TKSA

The mesh TKSA is a result of the AMPE. This security association is also bidirectional. The mesh TKSA shall be deleted when the lifetime expires. The mesh TKSA contains the following:

- MTK, as defined in 14.5.7
- PMKID
- Local mesh STA MAC address
- Peer mesh STA MAC address
- Local Link ID
- Peer Link ID
- Local nonce
- Peer nonce
- Lifetime as defined in 12.6.16
- Pairwise cipher suite selector

12.6.1.1.8 GTKSA

The GTKSA results from a successful 4-way handshake, FT 4-way handshake, FT protocol, FT resource request protocol or the group key handshake and is unidirectional. In an infrastructure BSS, there is one GTKSA, used exclusively for encrypting group addressed MPDUs that are transmitted by the AP and for

decrypting group addressed transmissions that are received by the STAs. In an IBSS or in a PBSS, each STA defines its own GTKSA, which is used to encrypt its group addressed transmissions, and stores a separate GTKSA for each peer STA so that encrypted group addressed traffic received from other STAs may be decrypted. A GTKSA is created by the Supplicant's SME when message 3 of the 4-way handshake is received, when message 1 of the group key handshake is received, or when the Reassociation Response frame of the FT handshake is received. The GTKSA is created by the Authenticator's SME when the SME changes the GTK and has sent the GTK to all STAs with which it has a PTKSA. A GTKSA consists of the following:

— Direction vector (whether the GTK is used for transmit or receive).
— Group cipher suite selector.
— GTK.
— Authenticator MAC address.
— Key ID.
— All authorization parameters specified by local configuration. This might include parameters such as the STA's authorized SSID.

When the GTK is used to encrypt individually addressed traffic (the selectable cipher suite is "Use group cipher suite"), the GTKSA is bidirectional.

12.6.1.1.9 IGTKSA

When management frame protection is enabled, a non-AP STA's SME creates an IGTKSA when it receives a valid message 3 of the 4-way handshake or FT 4-way handshake, the Reassociation Response frame of the fast BSS transition protocol with a status code indicating success, a Mesh Peering Open Message of the Authenticated Mesh Peering Exchange (AMPE) protocol, or a valid message 1 of the group key handshake. The Authenticator's SME creates an IGTKSA when it establishes or changes the IGTK with all STAs to which it has a valid PTKSA or mesh TKSA.

An IGTKSA consists of the following:

— Direction vector (whether the IGTK is used for transmit or receive)
— Key ID
— IGTK
— Authenticator MAC address

12.6.1.1.10 Mesh GTKSA

The mesh GTKSA results from a successful AMPE or mesh group key handshake and is unidirectional. In an MBSS, each mesh STA defines its own "transmit mesh GTKSA," which is used to encrypt its group addressed transmissions. Also, each mesh STA stores a separate "receive mesh GTKSA" for each peer mesh STA so that encrypted group addressed traffic received from the peer mesh STAs may be decrypted.

A transmit mesh GTKSA is created by a mesh STA after the SME has changed the mesh GTK (MGTK) and the new MGTK has been sent to all peer mesh STAs. A receive mesh GTKSA is created by a mesh STA after successfully completing the AMPE in which a wrapped MGTK has been received, or after receiving a valid message 1 of the mesh group key handshake. The receive mesh GTKSA shall be deleted when the lifetime expires or a new receive mesh GTKSA is created with the same Key ID for the same MGTK source mesh STA. See 14.6.1.

The MGTK and the GTK shall be independently selected from a uniform distribution. The MGTK source mesh STA MAC address in the mesh GTKSA shall not be the same as the Authenticator MAC address in the GTKSA.

NOTE—The use of a distinct Transmit MGTK and ESS GTK with identical transmit MAC addresses is precluded by limitations on key rollover and reception by STAs in an infrastructure BSS (see 14.11.5 for collocated mesh STA rules). If the distinct MGTKs were to use different Key IDs, then rollover would be impossible. Since Key ID 0 is reserved for individually addressed frame transmission, there are at most three available Key IDs (only two if extended Key IDs for individually addressed frames are in use), and the different MGTKs would contend for the single remaining Key ID upon rollover. If the distinct MGTKs were to use the same Key IDs, then STAs would incorrectly attempt to decrypt mesh broadcast traffic using the ESS GTK, causing error counters (such as dot11RSNAStatsCCMPDecryptErrors) to continuously increment. (See 12.9.2.6 for a description of the procedure for receiving encrypted frames.)

The mesh GTKSA contains the following:

— MGTK
— MGTK source mesh STA MAC address (mesh STA that uses this GTK to encrypt transmissions)
— Group cipher suite selector
— Direction vector (whether this is a receive mesh GTKSA or transmit mesh GTKSA)
— Key Index

12.6.1.1.11 SMKSA

An SMKSA is the result of a successful SMK handshake by the initiator STA (described in 12.7.8 and 12.11). It is derived from parameters provided by the STAs and AP. This security association is bidirectional between the initiator and the peer STA. In other words, both parties use the information in the security association for both sending and receiving. The SMKSA is created as a result of a successful SMK handshake (see 12.7.8 and 12.11). The SMKSA is used to create the STKSA. The SMKSA consists of the following:

— SMKID, as defined in 12.7.8. The SMKID identifies the security association.
— BSSID
— Initiator MAC address
— Peer MAC address
— SMK
— Lifetime, as defined in 12.7.8.
— Pairwise cipher suite selector list, as proposed by initiator STA
— Pairwise cipher suite selector, as selected by peer STA

12.6.1.1.12 STKSA

The STKSA is a result of successful completion of the 4-way STK handshake. This security association is bidirectional between the initiator and the peer STAs. The STKSA is used to create session keys to protect this STSL. STKSAs have the same lifetime as the SMKSA or the STSL, whichever comes first. There shall be only one STKSA with the same initiator STA and peer MAC addresses at any one time. STKSA is created as a result of PeerKey handshake (see 12.7.8) or the AP PeerKey protocol (see 12.11). The STKSA consists of the following:

— STK
— Pairwise cipher suite selector
— Initiator MAC address
— Peer MAC address
— Key ID

12.6.1.2 TPKSA

The TPKSA results from a successful completion of the TPK handshake. This security association is bidirectional between the TDLS initiator STA and the TDLS responder STA. The TPKSA is used to create

session keys to protect this TDLS session. The TPKSA has the lifetime indicated in the TPK handshake or the lifetime of the TDLS direct link, whichever comes first.

The TPKSA consists of the following:

— MAC addresses of the TDLS initiator STA and the TDLS responder STA
— Pairwise cipher suite selector
— TPK Lifetime
— TPK
— Link Identifier

12.6.1.3 Security association life cycle

12.6.1.3.1 General

A STA can operate in an ESS, an IBSS, an MBSS, or a PBSS, and a security association has a distinct life cycle for each. A STA that operates OCB does not have a security association life cycle.

12.6.1.3.2 Security association in an ESS

In an ESS there are two cases:

— Initial contact between the STA and the ESS
— Roaming by the STA within the ESS

A STA and AP establish an initial security association via the following steps:

a) The STA selects an authorized ESS by selecting among APs that advertise an appropriate SSID.
b) The STA then performs IEEE 802.11 authentication followed by association to the chosen AP. Confirmation of security parameters takes place during association. A STA performing IEEE 802.1X authentication uses Open System authentication. A STA performing password-based authentication can use SAE authentication.

NOTE 1—It is possible for more than one PMKSA to exist. As an example, a second PMKSA might come into existence through PMKSA caching. A STA might leave the ESS and flush its cache. Before its PMKSA expires in the AP's cache, the STA returns to the ESS and establishes a second PMKSA from the AP's perspective.

NOTE 2—An attack altering the security parameters is detected by the key derivation procedure.

NOTE 3—IEEE 802.11 Open System authentication provides no security, but is included to maintain backward compatibility with the IEEE 802.11 state machine (see 11.3).

c) SAE authentication provides mutual authentication and derivation of a PMK. If Open System authentication is chosen instead, the Authenticator or the Supplicant initiates IEEE 802.1X authentication. The EAP method used by IEEE Std 802.1X-2010 needs to support mutual authentication, as the STA needs assurance that the AP is a legitimate AP.

NOTE 1—Prior to the completion of IEEE 802.1X authentication and the installation of keys, the IEEE 802.1X Controlled Port in the AP blocks all Data frames. The IEEE 802.1X Controlled Port returns to the unauthorized state and blocks all Data frames before invocation of an MLME-DELETEKEYS.request primitive. The IEEE 802.1X Uncontrolled Port allows IEEE 802.1X frames to pass between the Supplicant and Authenticator. Although IEEE Std 802.1X-2010 does not require a Supplicant Controlled Port, this standard assumes that the Supplicant has a Controlled Port in order to provide the needed level of security. Supplicants without a Controlled Port compromise RSN security and are not used.

NOTE 2—Any secure network cannot support promiscuous association, e.g., an unsecured operation of IEEE Std 802.11. A trust relationship is needed between the STA and the AS of the targeted SSID prior to association and secure operation, in order for the association to be trustworthy. The reason is that an attacker can deploy a rogue AP just as easily as a legitimate network provider can deploy a legitimate AP, so some sort of prior relationship is necessary to establish credentials between the ESS and the STA.

d) The last step is key management. The authentication process, whether SAE authentication utilizing Authentication frames or IEEE 802.1X authentication utilizing Data frames post association, creates cryptographic keys shared between the cryptographic endpoints—the AP and STA, or the IEEE 802.1X AS and the STA, when using SAE or IEEE Std 802.1X, respectively. When using IEEE Std 802.1X, the AS transfers these keys to the AP, and the AP and STA uses one of the key confirmation handshakes, e.g., the 4-way handshake or FT 4-way handshake, to complete security association establishment. When using SAE authentication there is no AS and therefore no key transfer; the 4-way handshake is performed directly between the AP and STA. The key confirmation handshake indicates when the link has been secured by the keys and is ready to allow normal data traffic and protected robust Management frames.

When FT is not enabled, a STA roaming within an ESS establishes a new PMKSA by one of the four schemes:

— In the case of (re)association followed by IEEE 802.1X or PSK authentication, the STA repeats the same actions as for an initial contact association, but its Supplicant also deletes the PTKSA when it roams from the old AP. The Supplicant also deletes the PTKSA when it disassociates/deauthenticates from all BSSIDs in the ESS.

— In the case of SAE authentication followed by (re)association, the STA repeats the same actions as for initial contact association, but the non-AP STA also deletes the PTKSA when it roams from the old AP. Note that a STA can take advantage of the fact that it can perform SAE authentication to multiple APs while maintaining a single association with one AP, and then use any of the PMKSAs created during authentication to effect a fast BSS transition.

— A STA (AP) can cache PMKSAs for APs (STAs) in the ESS to which it has previously performed a full IEEE 802.1X authentication or SAE authentication. If a STA wishes to roam to an AP for which it has cached one or more PMKSAs, it can include one or more PMKIDs in the RSNE of its (Re)Association Request frame. An AP that has retained the PMK for one or more of the PMKIDs can proceed with the 4-way handshake. The AP shall include the PMKID of the selected PMKSA in message 1 of the 4-way handshake. If none of the PMKIDs of the cached PMKSAs matches any of the supplied PMKIDs, or if the AKM of the cached PMKSA differs from that offered in the (Re)Association Request, then the Authenticator, in the case of Open System authentication, shall perform another IEEE 802.1X authentication and, in the case of SAE authentication, shall transmit a Deauthentication frame to the STA. Similarly, if the STA fails to send a PMKID, the STA and AP need to perform a full IEEE 802.1X authentication.

— A STA already associated with the ESS can request its IEEE 802.1X Supplicant to authenticate with a new AP before associating to that new AP. The normal operation of the DS via the old AP provides the communication between the STA and the new AP. The SME delays reassociation with the new AP until IEEE 802.1X authentication completes via the DS. If IEEE 802.1X authentication completes successfully, then PMKSAs shared between the new AP and the STA are cached, thereby enabling the possible usage of reassociation without requiring a subsequent full IEEE 802.1X authentication procedure.

The MLME-DELETEKEYS.request primitive deletes the temporal key(s) established for the security association so that they cannot be used to protect subsequent IEEE 802.11 traffic. An SME uses this primitive when it deletes a PTKSA, GTKSA, or IGTKSA.

12.6.1.3.3 Security association in an IBSS

In an IBSS utilizing IEEE 802.11 Open System authentication and IEEE Std 802.1X, when a STA's SME establishes a security association with a peer STA, it creates both an IEEE 802.1X Supplicant and Authenticator for the peer. A STA in such an IBSS might also receive IEEE 802.1X messages from a previously unknown MAC address.

In an IBSS utilizing IEEE 802.11 SAE authentication, a STA creates a security association for a peer upon successful SAE authentication.

Any IBSS STA may decline to form a security association with a STA joining the IBSS. An attempt to form a security association may also fail because, for example, the peer uses a different PSK or password from what the STA expects.

In an IBSS each STA defines its own group key, i.e., GTK, to secure its group addressed transmissions. Each STA shall use either the 4-way handshake or the group key handshake to distribute its transmit GTK to its new peer STA. When the STA generates a new GTK, it also uses the group key handshake to distribute the new GTK to each established peer.

12.6.1.3.4 Security association in an MBSS

In order to create a secure peering, mesh STAs first authenticate each other and create a mesh PMKSA. This can be done using either SAE or IEEE Std 802.1X. A mesh STA shall support SAE authentication (see 12.4). A mesh STA may support IEEE 802.1X authentication (see 4.10).

When dot11MeshActiveAuthenticationProtocol is sae (1), the scanning mesh STA shall initiate SAE to the candidate mesh STA. If SAE terminates unsuccessfully, the scanning mesh STA shall terminate the peering establishment procedure. Otherwise, the PMK that results from successful SAE authentication shall be used to create a mesh PMKSA.

When dot11MeshActiveAuthenticationProtocol is ieee8021x (2), then the scanning mesh STA shall initiate the MPM protocol to establish a peering. If the MPM protocol fails then the scanning mesh STA shall terminate the peering establishment procedure. Otherwise, IEEE 802.1X authentication shall be performed between the two peers according to the following:

a) If only one mesh STA has the Connected to AS field set to 1, that STA shall act as the IEEE 802.1X Authenticator and the other STA shall act as the IEEE 802.1X Supplicant;
b) If both mesh STAs have the Connected to AS field set to 1, then the mesh STA with the higher MAC address shall act as the IEEE 802.1X Authenticator and the other mesh STA shall act as the IEEE 802.1X Supplicant (see 12.7.1 for MAC address comparison).

If IEEE 802.1X authentication fails, the peering establishment procedure shall be terminated and the peering established between the two mesh STAs shall be closed. Otherwise, the peering established between the two mesh STAs shall be closed and a mesh PMKSA shall be created using the PMK that resulted from the successful IEEE 802.1X authentication.

12.6.1.3.5 Security association in a PBSS

A STA and a peer establish an initial security association via the following steps:

a) The STA selects an authorized PBSS by identifying a peer from a DMG Beacon, Announce, Probe Response, or Information Response frames.
b) A STA may associate with a peer if the peer is a PCP.
c) If authentication is required, the STA or the peer initiates IEEE 802.1X authentication. The EAP method used by IEEE Std 802.1X-2010 shall support mutual authentication.
d) The last step is key management. The authentication process creates cryptographic keys shared between the IEEE 802.1X AS and the STA. The AS transfers these keys to the peer and the peer and the STA use one of the key confirmation handshakes, e.g., the 4-way handshake or FT 4-way handshake, to complete security association establishment. The key confirmation handshake indicates when the link has been secured by the keys and is ready to allow data traffic and protected robust Management frames.

12.6.2 RSNA selection

A STA prepared to establish RSNAs shall advertise its capabilities by including the RSNE in Beacon, Information Response, and Probe Response frames and may also include the RSNE in DMG Beacon and Announce frames. The included RSNE shall specify all of the authentication and cipher suites enabled by the STA's policy. A STA shall not advertise any authentication or cipher suite that is not enabled.

The SME shall utilize the MLME-SCAN.request primitive to identify neighboring STAs that assert robust security and advertise an SSID identifying an authorized ESS, PBSS, or IBSS. A STA may decline to communicate with STAs that fail to advertise an RSNE in their Beacon, Information Response, and Probe Response frames or that do not advertise an authorized SSID. A STA may also decline to communicate with other STAs that do not advertise authorized authentication and cipher suites within their RSNEs.

A STA shall advertise the same RSNE in its Beacon, DMG Beacon, Announce, Information Response, and Probe Response frames.

NOTE—Whether a STA with robust security enabled attempts to communicate with a STA that does not include the RSNE is a matter of policy.

A STA shall observe the following rules when processing an RSNE:

— A STA shall advertise the highest version it supports.
— A STA shall request the highest Version field value it supports that is less than or equal to the version advertised by the peer STA.
— Two peer STAs without overlapping supported Version field values shall not use RSNA methods to secure their communication.
— A STA shall ignore suite selectors that it does not recognize.

12.6.3 RSNA policy selection in an infrastructure BSS

RSNA policy selection in an infrastructure BSS utilizes the normal IEEE 802.11 association procedure. RSNA policy selection is performed by the associating STA. The STA does this by including an RSNE in its (Re)Association Requests.

In an RSN, an AP shall not associate with pre-RSNA STAs, i.e., with STAs that fail to include the RSNE in the (Re)Association Request frame.

An SME initiating an association shall insert an RSNE into its (Re)Association Request via the MLME-ASSOCIATE.request or MLME-REASSOCIATE.request primitive, when the targeted AP indicates RSNA support. The initiating STA's RSNE shall include one authentication and pairwise cipher suite from among those advertised by the targeted AP in its Beacon and Probe Response frames. It shall also specify the group cipher suite specified by the targeted AP. If at least one RSNE field from the AP's RSNE fails to overlap with any value the STA supports, the STA shall decline to associate with that AP. An HT STA shall eliminate TKIP as a choice for the pairwise cipher suite if CCMP is advertised by the AP or if the AP included an HT Capabilities element in its Beacon and Probe Response frames. The elimination of TKIP as a choice for the pairwise cipher suite may result in a lack of overlap of the remaining pairwise cipher suite choices, in which case the STA shall decline to create an RSN association with that AP.

If an RSNA-capable AP receives a (Re)Association Request frame that includes an RSNE and if it chooses to accept the association as a secure association, then it shall use the authentication and pairwise cipher suites in the (Re)Association Request frame, unless the AP includes an optional second RSNE in message 3 of the 4-way handshake. If the second RSNE is supplied in message 3, then the pairwise cipher suite used by the security association, if established, shall be the pairwise cipher from the second RSNE.

In order to accommodate local security policy, a STA may choose not to associate with an AP that does not support any pairwise cipher suites. An AP may indicate that it does not support any pairwise keys by advertising 00-0F-AC:0 (Use group cipher suite) as the pairwise cipher suite selector.

NOTE—When an ESS uses PSKs, STAs negotiate a pairwise cipher. However, any STA in the ESS can derive the pairwise keys of any other that uses the same PSK by capturing the first two messages of the 4-way handshake. This provides malicious insiders with the ability to eavesdrop as well as the ability to establish a man-in-the-middle attack.

An RSNA-enabled AP shall use Table 12-2 and the values of the Management Frame Protection Capable (MFPC) and Management Frame Protection Required (MFPR) bits advertised in the RSNEs to determine if it may associate with a non-AP STA. An RSNA enabled non-AP STA shall use Table 12-2 and the values of the Management Frame Protection Capable and Management Frame Protection Required bits advertised in the RSNEs to determine if it may associate with an AP. Management frame protection is enabled when dot11RSNAProtectedManagementFramesActivated is set to 1. Management frame protection is negotiated when an AP and non-AP STA set the Management Frame Protection Capable field to 1 in their respective RSNEs in the (re)association procedure, and both parties confirm the Management Frame Protection Capable bit set to 1 in the 4-way handshake, FT 4-way handshake, or the FT fast BSS transition protocol.

Table 12-2—Robust management frame selection in an infrastructure BSS

AP MFPC	AP MFPR	STA MFPC	STA MFPR	AP action	STA action
0	0	0	0	The AP may associate with the STA	The STA may associate with the AP
1	0	0	0	The AP may associate with the STA	The STA may associate with the AP
1	0 or 1	1	0 or 1	The AP may associate with the STA	The STA may associate with the AP
1	1	0	0	The AP shall reject associations from the STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION	The STA shall not associate with the AP
0	0	1	1	No action	The STA shall not try to associate with the AP
0	0	1	0	The AP may associate with the STA	The STA may associate with the AP
1	0 or 1	0	1	The STA advertises an invalid setting. The AP shall reject associations from the STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION	The STA shall not try to associate with the AP
0	1	1	0 or 1	No action	The AP advertises an invalid setting. The STA shall not try to associate with the AP

12.6.4 TSN policy selection in an infrastructure BSS

In a TSN, an RSNA STA shall include the RSNE in its (Re)Association Requests.

An RSNA-capable AP configured to operate in a TSN shall include the RSNE and may associate with both RSNA and pre-RSNA STAs. In other words, an RSNA-capable AP shall respond to an associating STA that includes the RSNE just as in an RSN.

If an AP operating within a TSN receives a (Re)Association Request frame without an RSNE, its IEEE 802.1X Controlled Port shall initially be blocked. The SME shall unblock the IEEE 802.1X Controlled Port when WEP has been enabled.

12.6.5 RSNA policy selection in an IBSS and for DLS

In an IBSS all STAs use a single group cipher suite, and all STAs support a common subset of pairwise cipher suites. However, the SMEs of any pair of non-HT STAs may negotiate to use any common pairwise cipher suite they both support. Each STA shall include the group cipher suite and its list of pairwise cipher suites in its Beacon and Probe Response frames. Two STAs shall not establish a PMKSA unless they have advertised the same group cipher suite. Similarly, the two STAs shall not establish a PMKSA if the STAs have advertised disjoint sets of pairwise cipher suites.

An HT STA that is in an IBSS or that is transmitting frames through a direct link shall eliminate TKIP as a choice for the pairwise cipher suite if CCMP is advertised by the other STA or if the other STA included an HT Capabilities element in any of its Beacon, Probe Response, DLS Request, or DLS Response frames.

NOTE—The elimination of TKIP as a choice for the pairwise cipher suite might result in a lack of overlap of the remaining pairwise cipher suite choices, in which case the STAs do not exchange encrypted frames.

In order to set up a security association with a peer STA, the SME of an IBSS STA that does not know the peer's policy needs first to obtain the peer's security policy using a Probe Request frame. The SME entities of the two STAs select the pairwise cipher suites using one of the 4-way handshakes. The SMEs of each pair of STAs within an IBSS may use the EAPOL-Key 4-way handshake to select a pairwise cipher suite. As specified in 12.7.2, message 2 and message 3 of the 4-way handshake convey an RSNE. The message 2 RSNE includes the selected pairwise cipher suite, and message 3 includes the RSNE that the STA would send in a Probe Response frame.

If the 4-way handshake is successfully completed, then the pair of STAs shall use the pairwise cipher suite specified in message 2 of the 4-way handshake initiated by the Authenticator STA with the higher MAC address (see 12.7.1).

The SME shall check that the group cipher suite and AKMP match those in the Beacon and Probe Response frames for the IBSS.

NOTE 1—The RSNEs in message 2 and message 3 are not the same as in the Beacon frame. The group cipher and AKMP are the same, but the pairwise ciphers might differ because Beacon frames from different STAs might advertise different pairwise ciphers. Thus, IBSS STAs use the same AKM suite and group cipher, while different pairwise ciphers might be used between STA pairs.

NOTE 2—When an IBSS uses PSKs, STAs can negotiate a pairwise cipher. However, any STA in the IBSS can derive the PTKs of any other that uses the same PSK by capturing the first two messages of the 4-way handshake. This provides malicious insiders with the ability to eavesdrop as well as the ability to establish a man-in-the-middle attack.

To establish a connection with a peer STA, an RSNA enabled STA that implements management frame protection shall use Table 12-3 and the MFPC and MFPR values advertised in the RSNEs exchanged in the 4-way handshake initiated by the Authenticator of the STA with the larger MAC address to determine if the communication is allowed. Management frame protection is enabled when dot11RSNAProtectedManagementFramesActivated is set to 1. The STAs negotiate protection of Management frames when the both STAs set the Management Frame Protection Capable subfield to 1 during the 4-way handshake.

Table 12-3—Robust management frame selection in an IBSS

MFPC	MFPR	Peer STA MFPC	Peer STA MFPR	STA action
0	0	0	0	The STA may exchange data with the peer STA.
1	0	0	0	The STA may exchange data with the peer STA.
1	0 or 1	1	0 or 1	The STA may exchange data with the peer STA.
1	1	0	0	The STA shall not exchange data with the peer STA and shall reject security association attempts from the peer STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION.
0	0	1	1	The STA shall not exchange data with the peer STA and shall reject security association attempts from the peer STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION.
0	0	1	0	The STA may establish a security association with the peer STA.
1	0 or 1	0	1	The STA shall not establish a security association with the peer STA and shall reject security association attempts from the peer STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION because the peer STA is advertising an invalid setting. The STA shall not exchange data with the peer STA.
0	1	1	0 or 1	The peer STA shall not establish a security association with the peer STA and shall reject security association attempts from the STA with the Status Code ROBUST_MANAGEMENT_POLICY_VIOLATION because the STA is advertising an invalid setting.

12.6.6 TSN policy selection in an IBSS

Pre-RSNA STAs generate Beacon and Probe Response frames without an RSNE and ignore the RSNE because it is unknown to them. This allows an RSNA STA to identify the pre-RSNA STAs from which it has received Beacon and Probe Response frames.

If an RSNA STA's SME instead identifies a possible IBSS member on the basis of a received group addressed message, via MLME-PROTECTEDFRAMEDROPPED.indication primitive, it cannot identify the peer's security policy directly. The SME might attempt to obtain the peer STA's security policy via a Probe Request frame.

12.6.7 RSNA policy selection in an MBSS

RSNA policy is advertised in Beacon frames and Probe Response frames. A mesh STA identifies a candidate peer by parsing its neighbor STA's Beacon frames and Probe Response frames (see 14.2). An HT mesh STA shall eliminate TKIP as a choice for the pairwise cipher suite if CCMP is advertised by the peer or if the peer included an HT Capabilities element in any of its Beacon or Probe Response frames.

All mesh STAs in an MBSS use the same group cipher suite. Mesh STAs establish authenticated peerings with each other using the AMPE protocol (see 14.5). In AMPE, mesh STAs negotiate a pairwise cipher suite, and establish a mesh TKSA, to protect individually addressed frames and state a group cipher suite and establish a mesh GTKSA to process incoming group addressed frames from a peer.

The AMPE performs key confirmation of a secret, authenticated, and shared PMK derived by an authenticated key management protocol (see 12.4) and derives pairwise symmetric keys.

12.6.8 RSNA policy selection in a PBSS

RSNA policy selection in a PBSS utilizes the association procedure (11.3.1) if the initiating STA chooses to associate with a PCP. RSNA policy selection is performed by the associating STA. The STA does this by including an RSNE in its (Re)Association Requests.

The STA follows the procedures in 12.5.3 to select RSNA policy with the PCP, with the PCP taking the role of the AP. If the initiating STA chooses not to associate with a peer in a PBSS, it follows the procedures in 12.5.5 to select RSNA policy with the peer.

12.6.9 RSN management of the IEEE 802.1X Controlled Port

When the policy selection process chooses IEEE 802.1X authentication, this standard assumes that IEEE 802.1X Supplicants and Authenticators exchange protocol information via the IEEE 802.1X Uncontrolled port. The IEEE 802.1X Controlled Port is blocked from passing general data traffic between the STAs until an IEEE 802.1X authentication procedure completes successfully over the IEEE 802.1X Uncontrolled Port. The security of an RSNA depends on this assumption being true.

In an infrastructure BSS or PBSS, if the STA associates with the AP or PCP, the STA indicates the IEEE 802.11 link is available by invoking the MLME-ASSOCIATE.confirm or MLME-REASSOCIATE.confirm primitive. This signals the Supplicant that the MAC has transitioned from the disabled to enabled state. At this point, the Supplicant's Controlled Port is blocked, and communication of all non-IEEE-802.1X MSDUs sent or received via the port is not authorized.

In an infrastructure BSS or PBSS, if the AP or PCP associates with a STA, the AP or PCP indicates that the IEEE 802.11 link is available by invoking the MLME-ASSOCIATE.indication or MLME-REASSOCIATE.indication primitive. At this point the Authenticator's Controlled Port corresponding to the STA's association is blocked, and communication of all non-IEEE-802.1X MSDUs sent or received via the Controlled Port is not authorized.

In an IBSS the STA shall block all IEEE 802.1X ports at initialization. Communication of all non-IEEE 802.1X MSDUs sent or received via the Controlled Port is not authorized.

In a PBSS, if a STA chooses not to associate with the PCP, the STA shall block all IEEE 802.1X ports at initialization. Communication of all non-IEEE-802.1X MSDUs sent or received via the Controlled Port is not authorized.

This standard assumes each Controlled Port remains blocked until the IEEE 802.1X state variables portValid and keyDone both become true. This assumption means that the IEEE 802.1X Controlled Port discards MSDUs sent across the IEEE 802.11 channel prior to the installation of cryptographic keys into the MAC. This protects the STA's host from forged MSDUs written to the channel while it is still being initialized.

The MAC does not distinguish between MSDUs for the Controlled Port, and MSDUs for the Uncontrolled Port. In other words, EAPOL-Start frames and EAPOL-Key frames are encrypted only after invocation of the MLME-SETPROTECTION.request primitive.

This standard assumes that IEEE Std 802.1X-2010 does not block the Controlled Port when authentication is triggered through reauthentication. During IEEE 802.1X reauthentication, an existing RSNA can protect all MSDUs exchanged between the STAs. Blocking MSDUs is not required during reauthentication over an RSNA.

12.6.10 RSNA authentication in an infrastructure BSS

12.6.10.1 General

When establishing an RSNA in a non-FT environment or during an FT initial mobility domain association, a STA shall use IEEE 802.11 SAE authentication or Open System authentication prior to (re)association.

SAE authentication is initiated when a STA's MLME-SCAN.confirm primitive finds another AP within the current ESS that advertises support for SAE in its RSNE.

IEEE 802.1X authentication is initiated by any one of the following mechanisms:

— If a STA negotiates to use IEEE 802.1X authentication during (re)association, the STA's management entity may respond to the MLME-ASSOCIATE.confirm (or indication) or MLME-REASSOCIATE.confirm (or indication) primitive by requesting the Supplicant (or Authenticator) to initiate IEEE 802.1X authentication. Thus, in this case, authentication is driven by the STA's decision to associate and the AP's decision to accept the association.

— If a STA's MLME-SCAN.confirm primitive finds another AP within the current ESS, a STA may signal its Supplicant to use IEEE Std 802.1X-2010 to preauthenticate with that AP.

NOTE—A roaming STA's IEEE 802.1X Supplicant can initiate preauthentication by sending an EAPOL-Start frame via its old AP, through the DS, to a new AP.

— If a STA receives an IEEE 802.1X message, it delivers this to its Supplicant or Authenticator, which may initiate a new IEEE 802.1X authentication.

12.6.10.2 Preauthentication and RSNA key management

Preauthentication allows a STA to perform RSNA authentication with an AP prior to attempting (re)association. This might reduce the time that the IEEE 802.1X port is not valid.

A STA shall not use preauthentication except when pairwise keys are employed. A STA shall not use preauthentication within the same mobility domain if AKM suite type 00-0F-AC:3 or 00-0F-AC:4 is used in the current association. Preauthentication shall not be used unless the new AP advertises the preauthentication capability in the RSNE.

When preauthentication is used, then:

a) Authentication is independent of roaming.

b) The Supplicant may authenticate with multiple APs at a time.

NOTE—Preauthentication might be useful as a performance enhancement, as reassociation does not include the protocol overhead of a full reauthentication when it is used.

Preauthentication uses the IEEE 802.1X protocol and state machines with EtherType 88-C7, rather than the EtherType 88-8E. Only IEEE 802.1X frame types EAP-Packet and EAPOL-Start are valid for preauthentication.

A Supplicant may initiate preauthentication when it has completed the 4-way handshake and configured the required temporal key(s). To effect preauthentication, the Supplicant sends an EAPOL-Start frame with the DA being the BSSID of a targeted AP and the RA being the BSSID of the AP with which it is associated. The target AP shall use a BSSID equal to the MAC address of its Authenticator. As preauthentication frames do not use the IEEE 802.1X EAPOL EtherType field, the AP with which the STA is currently associated need not apply any special handling. The AP and the MAC in the STA shall handle these frames in the same way as other frames with arbitrary EtherType field values that require distribution via the DS.

An AP's Authenticator that receives an EAPOL-Start frame via the DS may initiate IEEE 802.1X authentication to the STA via the DS. The DS forwards this message to the AP with which the STA is associated.

The result of preauthentication may be a PMKSA, if the IEEE 802.1X authentication completes successfully. The AKM shall be set to 00-0F-AC:1 in the PMKSA that results from preauthentication. If preauthentication produces a PMKSA, then, when the Supplicant's STA associates with the preauthenticated AP, the Supplicant may use the PMKSA with the 4-way handshake.

Successful completion of EAP authentication over IEEE Std 802.1X establishes a PMKSA at the Supplicant. The Authenticator has the PMKSA when the AS completes the authentication, passes the keying information (the master session key (MSK), a portion of which is the PMK) to the Authenticator, and the Authenticator creates a PMKSA using the PMK. The PMKSA is inserted into the PMKSA cache. Therefore, if the Supplicant and Authenticator lose synchronization with respect to the PMKSA, the 4-way handshake fails. In such circumstances, dot11RSNA4WayHandshakeFailures shall be incremented.

A Supplicant may initiate preauthentication with any AP within its present ESS with preauthentication enabled regardless of whether the targeted AP is within radio range.

Even if a STA has preauthenticated, it is still possible that it may have to undergo a full IEEE 802.1X authentication, as the AP's Authenticator may have purged its PMKSA due to, for example, unavailability of resources, delay in the STA associating, etc.

12.6.10.3 Cached PMKSAs and RSNA key management

In a non-FT environment, a STA might cache PMKSAs it establishes as a result of previous authentication. The PMKSA shall not be changed while cached. The PMKSA in the PMKSA is used with the 4-way handshake to establish fresh PTKs.

If a STA in an infrastructure BSS has determined it has a valid PMKSA with an AP to which it is about to (re)associate, it performs Open System authentication to the AP, and then it includes the PMKID for the PMKSA in the RSNE in the (Re)Association Request. When the PMKSA was not created using pre-authentication, the AKM indicated in the RSNE by the STA in the (Re)Association Request shall be identical to the AKM used to establish the cached PMKSA in the first place.

Upon receipt of a (Re)Association Request frame with one or more PMKIDs, an AP checks whether its Authenticator has cached a PMKSA for the PMKIDs and whether the AKM in the cached PMKSA matches the AKM in the (Re)Association Request; and if so, it shall assert possession of that PMKSA by beginning the 4-way handshake after association has completed. If the Authenticator does not have a PMKSA for the PMKIDs in the (Re)Association Request, its behavior depends on how the PMKSA was established. If SAE authentication was used to establish the PMKSA, then the AP shall reject (re)association by sending a (Re)Association Response frame with status code STATUS_INVALID_PMKID. Note that this allows the non-AP STA to fall back to full SAE authentication to establish another PMKSA. If IEEE 802.1X authentication was used to establish the PMKSA, the AP begins a full IEEE 802.1X authentication after association has completed.

If both sides assert possession of a cached PMKSA, but the 4-way handshake fails, both sides may delete the cached PMKSA for the selected PMKID.

If the lifetime of a cached PMKSA expires, the STA shall delete the expired PMKSA.

If a STA roams to an AP with which it is preauthenticating and the STA does not have a PMKSA for that AP, the STA needs to initiate a full IEEE 802.1X EAP authentication.

12.6.11 RSNA authentication in an IBSS

When authentication is used in an IBSS, it is driven by each STA wishing to establish communications. The management entity of this STA chooses a set of STAs with which it might need to authenticate and then may cause the MAC to send an IEEE 802.11 Open System authentication message to each targeted STA. Candidate STAs can be identified from Beacon frames, Probe Response frames, and Data frames from the same BSSID. Before communicating with STAs identified from Data frames, the security policy of the STAs may be obtained, e.g., by sending a Probe Request frame to the STA and obtaining a Probe Response frame. Targeted STAs that wish to respond may return an IEEE 802.11 Open System authentication message to the initiating STA.

When IEEE 802.1X authentication is used, the STA management entity requests its local IEEE 802.1X entity to create a Supplicant PAE for the peer STA. The Supplicant PAE initiates the authentication to the peer STA by sending an EAPOL-Start frame to the peer. The STA management entity also requests its local IEEE 802.1X entity to create an Authenticator PAE for the peer STA on receipt of the EAPOL-Start frame. The Authenticator initiates authentication to the peer STA by sending an EAP-Request message or, if PSK mode is in effect, message 1 of the 4-way handshake.

Upon initial authentication between any pair of STAs, Data frames, other than IEEE 802.1X messages, are not allowed to flow between the pair of STAs until both STAs in each pair of STAs have successfully completed AKM and have provided the supplied encryption keys.

Upon the initiation of an IEEE 802.1X reauthentication by any STA of a pair of STAs, Data frames continue to flow between the STAs while authentication completes. Upon a timeout or failure in the authentication process, the Authenticator of the STA initiating the reauthentication shall cause a Deauthentication message to be sent to the Supplicant of the STA targeted for reauthentication. The Deauthentication message causes both STAs in the pair of STAs to follow the deauthentication procedure defined in 11.3.4.4 and 11.3.4.5.

The IEEE 802.1X reauthentication timers in each STA are independent. If the reauthentication timer of the STA with the higher MAC address (see 12.7.1 for MAC comparison) triggers the reauthentication via its Authenticator, its Supplicant shall send an EAPOL-Start frame to the authenticator of the STA with the lower MAC address (see 12.7.1 for MAC comparison) to trigger reauthentication on the other STA. This process keeps the pair of STAs in a consistent state with respect to derivation of one or more fresh temporal keys upon an IEEE 802.1X reauthentication.

When it receives an MLME-AUTHENTICATE.indication primitive due to an Open System authentication request, the IEEE 802.11 management entity on a targeted STA shall, if it intends to set up a security association with the peer STA, request its Authenticator to begin IEEE 802.1X authentication, i.e., to send an EAP-Request/Identity message or message 1 of the 4-way handshake to the Supplicant.

The EAPOL-Key frame is used to exchange information between the Supplicant and the Authenticator to negotiate a fresh PTK. The 4-way handshake produces a single PTK from the PMK. The 4-way handshake and group key handshake use the PTK to protect the GTK as it is transferred to the receiving STA.

Password or PSK authentication may also be used in an IBSS. When a single password or PSK is shared among the IBSS STAs, an SAE capable STA wishing to establish communication with a STA that advertises support for SAE in Beacon and Probe Response frames invokes SAE authentication, and upon successful conclusion of SAE, sends 4-way handshake message 1 to the target STA. If the STA does not support SAE authentication or the target STA does not advertise support for SAE in Beacon and Probe Response frames, the STA may use the PSK as a PMK and initiate the 4-way handshake by sending a 4-way handshake message 1 to the target STA. In either case, the targeted STA responds to message 1 with message 2 of the 4-way handshake and begins its 4-way handshake by sending message 1 to the initiating STA. The two 4-way handshakes establish PTKSAs and GTKSAs to be used between the initiating STA and

the targeted STA. PSK PMKIDs have security vulnerabilities when used with low-entropy keys and should be used only after taking this into account.

The model for security in an IBSS is not general. In particular, it assumes the following:

a) The sets of use cases for which the authentication procedures described in this subclause are valid are as follows:
   1) Password or PSK-based authentication using SAE to perform mutual authentication and generation of a shared PMK.
   2) An alternate form of PSK-based authentication, typically managed by the pass-phrase hash method as described in J.4. This method has security vulnerabilities and should only be used when SAE authentication is not possible.
   3) EAP-based authentication, using credentials that have been issued and preinstalled on the STAs within a common administrative domain, such as a single organization
b) All of the STAs are in direct radio communication. In particular, there is no routing, bridging, or forwarding of traffic by a third STA to effect communication. This assumption is made, because the model makes no provision to protect IBSS topology information from tampering by one of the members.

12.6.12 RSNA authentication in an MBSS

When establishing an RSNA in an MBSS, a mesh STA shall use IEEE 802.11 SAE authentication (see 12.4), or optionally IEEE 802.1X authentication, prior to establishment of an authenticated peering. An RSNA security association, called a *mesh PMKSA*, is created upon successful completion of authentication. The mesh PMKSA is used with the AMPE to create a mesh TKSA.

A mesh PMKSA may be cached and used with the AMPE to establish a fresh mesh TKSA. A STA includes the PMKID for the PMKSA in the Mesh Peering Open frame. If the PMKID in a received Mesh Peering Open frame is unknown, the AMPE handshake fails. If both sides assert possession of a cached mesh PMKSA but the AMPE handshake fails, a STA may delete the cached mesh PMKSA for the selected PMKID. A mesh PMKSA shall not be changed while cached.

Authentication using IEEE 802.11 SAE authentication is based on a password. A password is required to be shared between two mesh STAs in order to successfully complete authentication. This password can be pairwise – each pair of mesh STAs in an MBSS has a unique password – or it can be shared-all mesh STAs in the MBSS share the same password.

Due to the security properties of IEEE 802.11 SAE authentication, an adversary has no greater possibility in determining a shared password than in determining a pairwise password. The potential for misuse, though, is greater if a shared password becomes known to an adversary because an unlimited number of mesh STAs under the control of the adversary can be added to the MBSS.

12.6.13 RSNA authentication in a PBSS

IEEE 802.11 Open System authentication is not used in a PBSS.

When establishing an RSNA with a PCP, a STA may associate to the PCP and initiate RSNA authentication with the PCP following the procedures of 12.6.10, with the PCP taking the role of the AP. When a STA chooses not to associate to a peer, it initiates RSNA authentication with the peer following the procedures of 12.6.11 with the following caveat: if both peers simultaneously initiate RSNA authentication, the peer with the lower MAC address shall abandon the authentication it initiated in favor of the authentication initiated by the peer with the higher MAC address.

12.6.14 RSNA key management in an infrastructure BSS

When the IEEE 802.1X authentication completes successfully, this standard assumes that the STA's IEEE 802.1X Supplicant and the IEEE 802.1X AS share a secret, called a *PMK*. In a non-FT environment, the AS transfers the PMK, within the MSK, to the AP, using a technique that is outside the scope of this standard; the derivation of the PMK from the MSK is EAP-method-specific. With the PMK in place, the AP initiates a key confirmation handshake with the STA. The key confirmation handshake sets the IEEE 802.1X state variable portValid (as described in IEEE Std 802.1X-2010) to true.

When SAE authentication completes, both STAs share a PMK. With this PMK in place, the AP initiates the key confirmation handshake with the STA.

The key confirmation handshake is implemented by the 4-way handshake. The purposes of the 4-way handshake are as follows:

a) Confirm the existence of the PMK at the peer.
b) Ensure that the security association keys are fresh.
c) Synchronize the installation of one or more temporal keys into the MAC.
d) Transfer the GTK from the Authenticator to the Supplicant.
e) Confirm the selection of cipher suites.

NOTE 1—It is possible to forge message 1 of the 4-way handshake. However, the forgery attempt is detected in the failure of the 4-way handshake.

NOTE 2—Neither the AP nor the STA can use the PMK for any purpose but the one specified herein without compromising the key. If the AP uses it for another purpose, then the STA can masquerade as the AP; similarly if the STA reuses the PMK in another context, then the AP can masquerade as the STA.

The Supplicant and Authenticator signal the completion of key management by utilizing the MLME-SETKEYS.request primitive to configure the agreed-upon temporal pairwise key into the IEEE 802.11 MAC and by calling the MLME-SETPROTECTION.request primitive to enable its use.

A second key exchange, the group key handshake, is also defined. It distributes a subsequent GTK. The AP's Authenticator can use the group key handshake to update the GTK at the Supplicant. The group key handshake uses the EAPOL-Key frames for this exchange. When it completes, the Supplicant can use the MLME-SETKEYS.request primitive to configure the GTK into the IEEE 802.11 MAC.

12.6.15 RSNA key management in an IBSS

To establish a security association between two IBSS STAs, each STA's SME has an accompanying IEEE 802.1X Authenticator and Supplicant. Each SME initiates the 4-way handshake from the Authenticator to the peer STA's Supplicant (see 12.6.11). Two separate 4-way handshakes are conducted.

The 4-way handshake is used to negotiate the pairwise cipher suites, as described in 12.6.5. The IEEE 802.11 SME configures the temporal key portion of the PTK into the IEEE 802.11 MAC. Each Authenticator uses the KCK and KEK portions of the PTK negotiated by the exchange it initiates to distribute its own GTK and if management frame protection is enabled, its own IGTK. Each Authenticator generates its own GTK and if management frame protection is enabled, its own IGTK, and uses either the 4-way handshake or the group key handshake to transfer the GTK and if management frame protection is negotiated, the IGTK, to other STAs with whom it has completed a 4-way handshake. The pairwise key used between any two STAs shall be the pairwise key from the 4-way handshake initiated by the STA with the highest MAC address.

A STA joining an IBSS is required to adopt the security configuration of the IBSS, which includes the group cipher suite, pairwise cipher suite, AKMP, and if management frame protection is enabled, group

management cipher suite (see 12.6.5). The STA shall not set up a security association with any STA having a different security configuration. The Beacon and Probe Response frames of the various STAs within an IBSS need to reflect a consistent security policy, as the beacon initiation rotates among the STAs.

A STA joining an IBSS shall support and advertise in the Beacon frame the security configuration of the IBSS, which includes the group cipher suite, advertised pairwise cipher suite, AKMP, and if management frame protection is enabled, group management cipher suite (see 12.6.5). The STA may use the Probe Request frame to discover the security policy of a STA, including additional individual cipher suites the STA supports. If enabled, management frame protection shall only be used as a required feature (MFPR) in an IBSS.

NOTE—Because of the requirement for a STA joining an IBSS to support the security configuration of the IBSS, all Beacon frames transmitted in an IBSS have the same security policy.

12.6.16 RSNA key management in an MBSS

Upon successful completion of the AMPE, a secure mesh peering is established between two mesh STAs. This secure mesh peering includes a mesh PMKSA and a mesh TKSA. Multiple mesh TKSAs may be created using a single mesh PMKSA (a limit to that number is a policy decision outside the scope of this standard).

A mesh TKSA is logically a child of the mesh PMKSA. A mesh TKSA shall be deleted if the corresponding mesh PMKSA, which was used by the AMPE to create it, is deleted. Mesh PMKSAs are limited by their lifetime (see 12.7.1.3).

12.6.17 RSNA key management in a PBSS

Upon successful association and authentication in a PBSS, a STA performs a key confirmation handshake with the PCP, following the procedures in 12.6.14 with the PCP taking the role of the AP. If a STA chooses not to associate to a peer, after successful authentication, it performs a key confirmation handshake with the peer following the procedures in 12.6.15 with the following caveat: if both peers simultaneously initiate the key confirmation handshake, the peer with the lower MAC address shall abandon the handshake it initiated in favor of the handshake initiated by the peer with the higher MAC address.

12.6.18 RSNA security association termination

When a non-AP STA's SME receives a successful MLME-ASSOCIATE.confirm or MLME-REASSOCIATE.confirm primitive that is not part of a fast BSS transition or receives or invokes an MLME Disassociation or Deauthentication primitive, it deletes some security associations. Similarly, when an AP's SME

- receives an MLME-ASSOCIATE.indication or MLME-REASSOCIATE.indication primitive from a STA that has not negotiated management frame protection, or
- receives an MLME-ASSOCIATE.indication or MLME-REASSOCIATE.indication primitive from a STA that has negotiated management frame protection that a) has resulted in an MLME (re)association response that is successful, and b) is not part of a fast BSS transition, or receives an MLME-DEAUTHENTICATE.indication or MLME-DISASSOCIATE.indication primitive or issues an MLME-DEAUTHENTICATE.request or MLME-DISASSOCIATE.request primitive,

it deletes some security associations. In the case of an ESS, the non-AP STA's SME shall delete the PTKSA, GTKSA, IGTKSA, SMKSA, any TPKSA, and any STKSA, and the AP's SME shall delete the PTKSA and invoke an STSL application teardown procedure for any of its STKSAs. An example of an STSL application teardown procedure is described in 11.7.4. In the case of an IBSS, the SME shall delete the PTKSA and the receive GTKSA and IGTKSA. Once the security associations have been deleted, the SME then invokes

MLME-DELETEKEYS.request primitive to delete all temporal keys associated with the deleted security associations.

If a STA loses key state synchronization, it can apply the following rules to recover:

a) Any protected frame(s) received shall be discarded, and MLME-PROTECTEDFRAMEDROPPED.indication primitive is invoked.

b) If the STA is RSNA-enabled and has joined an IBSS, the SME shall execute the authentication procedure as described in 11.3.4.2.

c) If the STA is RSNA-enabled and has joined an ESS, the SME shall execute the deauthentication procedures as described in 11.3.4.4. However, if the STA has initiated the RSN security association, but has not yet invoked the MLME-SETPROTECTION.request primitive, then no additional action is required.

NOTE 1—There is a race condition between when MLME-SETPROTECTION.request primitive is invoked on the Supplicant and when it is invoked on the Authenticator. During this time, the STA might receive an MPDU that it is unable to decrypt; and the MPDU is discarded without a deauthentication occurring.

NOTE 2—Because the IEEE 802.11 Null frame does not derive from an MA-UNITDATA.request primitive, it is not protected.

If the selected AKMP fails between a STA and an AP that are associated, then both the STA and the AP shall invoke the MAC deauthentication procedure described in 11.3.4.4.

If the SMK handshake fails between a pair of associated STAs and AP, then the STAs and the AP shall invoke an STSL application teardown procedure.

When a STA's SME receives an MLME-PN-EXHAUSTION.indication primitive and the PN is associated with a PTKSA, the STA's SME shall invoke an MLME-DISASSOCIATE.request primitive and delete the PTKSA.

When a STA's SME receives an MLME-PN-EXHAUSTION.indication primitive and the PN is associated with a GTKSA, the STA's SME shall delete the GTKSA.

When a STA's SME receives an MLME-PN-EXHAUSTION.indication primitive and the PN is associated with a STKSA, the STA's SME shall invoke a STSL application teardown procedure for the STKSA and delete the STKSA.

Once the security associations have been deleted, the SME then invokes MLME-DELETEKEYS.request primitive to delete all temporal keys associated with the deleted security associations.

12.6.19 Protection of robust Management frames

This subclause defines rules that shall be followed by STAs that implement Management Frame protection and have dot11RSNAActivated equal to true.

A STA with dot11RSNAProtectedManagementFramesActivated equal to false shall transmit and receive unprotected individually addressed robust Management frames to and from any associated STA and shall discard protected individually addressed robust Management frames received from any associated STA.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true and dot11RSNAUnprotectedManagementFramesAllowed equal to true shall transmit and receive unprotected individually addressed robust Management frames to and from any associated STA that advertised MFPC = 0 and shall discard protected individually addressed robust Management frames received from any associated STA that advertised MFPC = 0.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true and dot11RSNAUnprotectedManagementFramesAllowed equal to true shall transmit and receive protected individually addressed robust Management frames to and from any associated STA that advertised MFPC = 1, shall discard unprotected individually addressed robust Action frames received from any STA that advertised MFPC = 1, and shall discard unprotected individually addressed Disassociation and Deauthentication frames received from a STA that advertised MFPC = 1 after the PTK and IGTK have been installed. The receiver shall process unprotected individually addressed Disassociation and Deauthentication frames before the PTK and IGTK are installed.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true and dot11RSNAUnprotectedManagementFramesAllowed equal to false shall transmit and receive protected individually addressed robust Action frames to and from any STA, shall not transmit unprotected individually addressed robust Action frames to any STA, and shall discard unprotected individually addressed robust Action frames received from a STA after the PTK and IGTK have been installed. The receiver shall process unprotected individually addressed Disassociation and Deauthentication frames before the PTK and IGTK are installed.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true shall protect transmitted group addressed robust Management frames using the group management cipher suite.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true shall discard group addressed robust Management frames received from any associated STA that advertised MFPC = 1 if the frames are unprotected or if a matching IGTK is not available.

A STA with dot11RSNAProtectedManagementFramesActivated equal to true and dot11RSNAUnprotectedManagementFramesAllowed equal to false shall discard received group addressed robust Management frames that are unprotected or for which a matching IGTK is not available.

A STA with dot11RSNAProtectedManagementFramesActivated equal to false shall transmit group addressed robust Management frames unprotected and shall ignore the protection on received group addressed robust Management frames.

NOTE—BIP does not provide protection against forgery by associated and authenticated STAs. A STA that has left the group can successfully forge Management frames until the IGTK is updated.

Protection of group addressed robust Management frames shall be provided by a service in the MLME as described in 11.13.

Robust management frame protection cannot be applied until the PTK and IGTK has been established with the STA. A STA shall not transmit robust Action frames until it has installed the PTK for the peer STA, or in the case of group addressed frames, has installed the IGTK. The STA shall discard any robust Action frames received before the PTK and IGTK are installed.

12.6.20 Robust management frame selection procedure

A STA with dot11RSNAProtectedManagementFramesActivated equal to true shall negotiate robust management frame protection with a STA that advertised MFPC = 1.

When a Public Action frame is transmitted for which a Protected Dual of Public Action frame is defined, (see 9.6.11), which variant (i.e., protected or not protected) is used depends on the setting of the “Protected” parameter of the corresponding MLME .request or .confirm primitive. Where there is no such parameter, the protected variant is used when management frame protection has been negotiated.

12.6.21 RSNA rekeying

When a PTKSA is deleted, a non-AP and non-PCP STA may reassociate with the same AP or PCP and/or establish a new RSNA with the AP or PCP. If the non-AP and non-PCP STA has cached one or more PMKSAs, it may skip the PMKSA establishment and proceed with the creation of a new PTKSA by using 4-way handshake.

When a GTKSA is deleted, an originating STA may create a new GTKSA by using 4-way handshake or group key handshake.

When a STKSA is deleted, the STA_I may establish a new STSL with the STA_P. If the SMK between the STA pair has not expired, the STA_I may initiate a 4-way handshake and create a new STKSA with STA_P. If the SMK has expired, the STA_I shall create both a new SMKSA and a new STKSA with the STA_P.

An Authenticator/STA_I may initiate a 4-way handshake for the purpose of renewing the key associated with a PTKSA or STKSA. A supplicant/STA_P may send an EAPOL request message to the authenticator/ STA_I to request rekeying. In addition, if both the Authenticator and the Supplicant support multiple keys for individually addressed traffic, a smooth switchover to the new key is possible using the following procedure.

The IEEE 802.11 MAC shall issue an MLME-PN-WARNING.indication primitive when the Packet Number assignment for a particular PTKSA, GTKSA, or STKSA reaches or exceeds the threshold that is defined in dot11PNWarningThresholdLow and dot11PNWarningThresholdHigh for the first time. The indication shall be issued only once for a given PTKSA, GTKSA, or STKSA. The SME may use the indication as a trigger to establish a new PTKSA, GTKSA, or STKSA before the Packet Number space is exhausted.

A PTKSA or STKSA has a limited lifetime, either in absolute time or due to exhausting the PN space. To maintain an uninterrupted security association, a STA should establish a new PTKSA or STKSA prior to the expiration of the old PTKSA or STKSA.

When both ends of the link support extended Key IDs for individually addressed frames, it is possible to install the new PTKSA or STKSA without data loss, provided the new PTKSA or STKSA uses a different Key ID from the old PTKSA or STKSA. Data loss might occur if the same Key ID is used because it is not possible to precisely coordinate (due to software processing delays) when the new key is used for transmit at one end and when it is applied to receive at the other end. If a different Key ID is used for the new PTKSA or STKSA, then provided the new key is installed at the receive side prior to its first use at the transmit side there is no need for precise coordination. During the transition, received packets are unambiguously identified using the Key ID as belonging to either the old or new PTKSA or STKSA.

12.6.22 Multi-band RSNA

12.6.22.1 General

A STA is multi-band capable and RSNA-capable if the values of both dot11MultibandImplemented and dot11RSNAActivated are true.

A STA that is multi-band capable and RSNA-capable shall set the Pairwise Cipher Suite Present field of the Multi-band element to 1 and shall include the Pairwise Cipher Suite Count field and the Pairwise Cipher Suite List field in the Multi-band element. The STA may include the RSNE and the Multi-band element in the DMG Beacon and Announce frames and shall include the RSNE and the Multi-band element in Beacon, Probe Response, and Information Response frames. The included RSNE shall specify all of the authentication and cipher suites enabled by the STA’s policy for the band where this element is transmitted, and the included Multi-band element shall specify all of the pairwise cipher suites enabled by the STA’s

policy for the band specified within the Multi-band element. A STA shall not advertise any cipher suite that is not enabled.

A multi-band capable and RSNA-capable STA shall include the Multi-band element in the (Re)Association Request frame and in message 2 and message 3 of the 4-way handshake.

In order to set up an RSNA with a peer STA for a supported band/channel, a STA that does not know the peer's policy for the band/channel shall first obtain the peer STA's policy for the supported band/channel by using a Probe Request frame or Information Request frame. The STA initiating RSNA establishment for a supported band/channel is called *RSNA initiator*, and the targeted STA of the RSNA establishment is called *RSNA responder*.

A multi-band capable device can create its own group key(s) for one or more supported bands/channels. If the STA uses different MAC addresses in different bands/channels, different GTKSAs shall be created for different bands. If the STA uses a same MAC address in all supported bands/channels, a single GTKSA shall be created for all supported bands/channels.

If the pairwise and group cipher suites used by a pair of multi-band capable devices to communicate with each other in the current operating band/channel is also supported after the transfer to another band/channel that was performed using transparent FST, the devices shall continue using the same cipher suites to communicate with each other after the transfer. In all other cases, a separate RSNA has to be established for the other band/channel (see 12.6.22).

12.6.22.2 Nontransparent multi-band RSNA

An RSNA initiator can establish a nontransparent (11.33) multi-band RSNA with an RSNA responder for a supported band/channel other than the current operating band/channel. The two STAs use the same PMKSA for both the supported band/channel and the current operating band/channel and create different PTKSAs for different bands/channels.

If the RSNA initiator does not have an existing PMKSA with the RSNA responder, the RSNA initiator shall first establish a PMKSA with the RSNA responder in the current operating band/channel and then use the PMKSA to create a PTKSA with the RSNA responder for the supported band/channel. If the RSNA initiator has already established a PMKSA with the RSNA responder, the PMKSA shall be used to create a PTKSA between the two STAs for the supported band/channel.

With the PMK in place, the RSNA initiator and RSNA responder can proceed in two ways depending on the setting of the Joint Multi-band RSNA subfield within the RSN Capabilities field in the RSNE of both STAs.

If the Joint Multi-band RSNA subfield within the RSN Capabilities field of either the RSNA initiator or RSNA responder is 0, the STA pair uses a 4-way handshake to establish a PTKSA for the current band/channel and may start a separate 4-way handshake in the current operating band/channel to negotiate a pairwise cipher suite for the supported band/channel and establish a PTKSA for the supported band/channel. As specified in 12.7.6, message 2 and message 3 of the 4-way handshake convey the Multi-band element associated with the supported band/channel. The Multi-band element in message 2 includes the selected pairwise cipher suite for the supported band/channel. message 3 includes the Multi-band element that the STA would send in a Beacon, DMG Beacon, Announce, Probe Response, or Information Response frame. message 3 may include a second Multi-band element that indicates the STA's pairwise cipher suite assignment for the supported band/channel.

If the Joint Multi-band RSNA subfield within the RSN Capabilities field is 1 for both the RSNA initiator and the RSNA responder and at least one of the STAs uses different MAC addresses for different bands/channels, the STAs shall use a single 4-way handshake to negotiate pairwise cipher suites and establish PTKSAs for both the current operating band/channel and the other supported band(s)/channel(s). As

specified in 12.7.6, message 2 and message 3 of the 4-way handshake convey the RSNE and the Multi-band element(s). The RSNE in message 2 includes the selected pairwise cipher suite for the current operating band/channel, and the Multi-band element(s) in message 2 includes the selected pairwise cipher suite(s) for the other supported band(s)/channel(s). message 3 includes the RSNE and the Multi-band element(s) that the STA would send in a Beacon, DMG Beacon, Announce, Probe Response, or Information Response frame. message 3 may include a second RSNE and Multi-band element(s) that indicate the STA's pairwise cipher suite assignments for the current operating band/channel and the other supported band(s)/channel(s). KCK and KEK associated with the current operating band/channel shall be used in the 4-way handshake.

12.6.22.3 Transparent multi-band RSNA

An RSNA initiator can establish a transparent (11.33) multi-band RSNA with an RSNA responder for both the current operating band/channel and the other supported band(s)/channel(s) if

a) Both STAs use the same MAC address in the current operating band/channel and the other supported band(s)/channel(s); and
b) At least one common pairwise cipher suite is supported by both STAs in the current operating band/channel and the other supported band(s)/channel(s).

Two STAs that establish a transparent multi-band RSNA create one PMKSA and one PTKSA for both the current operating band/channel and other supported band(s)/channel.

A STA shall use a single PN counter (12.5.3.3 and 12.5.5.3) for transmission in both the current operating band and the other supported band(s) when transparent multi-band RSNA is used.

If the RSNA initiator does not have an existing PMKSA with the RSNA responder, the RSNA initiator shall first establish a PMKSA with the RSNA responder in the current operating band/channel and then use the PMKSA to create a PTKSA with the RSNA responder for all involved bands/channels. If the RSNA initiator has already established a PMKSA with the RSNA responder, the PMKSA shall be used to create a PTKSA between the two STAs for all involved bands/channels.

With the PMK in place, the STA pair shall use a single 4-way handshake in the current operating band/channel to negotiate a pairwise cipher suite for all involved bands/channels and also establish a single PTKSA for all involved bands/channels. As specified in 12.7.6, message 2 and message 3 of the 4-way handshake convey the RSNE and the Multi-band element(s). The RSNE and the Multi-band element(s) in message 2 include one selected pairwise cipher suite for all involved bands/channels. message 3 includes the RSNE and the Multi-band element(s) that the STA would send in a Beacon, DMG Beacon, Announce, Probe Response, or Information Response frame. message 3 may include a second RSNE and Multi-band element(s) that indicate the STA's pairwise cipher suite assignment for all involved bands/channels.

12.6.22.4 Multi-band RSNA with TDLS in a non-DMG BSS

When two multi-band capable devices operate in a non-DMG BSS and set up a TDLS direct link in the BSS, the TPK handshake protocol can be used to create a PTKSA for use in another supported band/channel that is supported by both STAs and that was indicated in the Multi-band element in each of the STAs. Only TK in PTKSA is required for the supported band/channel and it shall be equal to the TPK-TK of the TPK.

If at least one of the peer STAs has a different MAC address in the supported band/channel from that of the current operating band/channel, the TPK handshake protocol may be used to establish a PTKSA for the supported band/channel. In this case, the TPK creation method shall be used to calculate a different PTKSA in the supported band/channel: the TDLS peer MAC addresses and cipher suite shall be replaced by the MAC addresses and cipher suite indicated within the corresponding Multi-band elements contained in the TDLS Setup Request and TDLS Setup Response frames used to establish the PTKSA for the supported band/channel.

If two TDLS peer STAs use the same MAC addresses and pairwise cipher suites in the operating band/channel and in the supported band/channel, the TPKSA that is acquired by the successful completion of the TPK handshake protocol may be used as the PTKSA for the supported band/channel.

12.7 Keys and key distribution

12.7.1 Key hierarchy

12.7.1.1 General

RSNA defines the following key hierarchies:

a) Pairwise key hierarchy, to protect individually addressed traffic
b) GTK, a hierarchy consisting of a single key to protect group addressed traffic

NOTE—Pairwise key support with enhanced data cryptographic encapsulation mechanisms allows a receiving STA to detect MAC address spoofing and data forgery. The RSNA architecture binds the transmit and receive addresses to the pairwise key. If an attacker creates an MPDU with the spoofed TA, then the decapsulation procedure at the receiver generates an error. GTKs do not have this property.

c) Integrity GTK (IGTK), a hierarchy consisting of a single key to provide integrity protection for group addressed robust Management frames

The description of the key hierarchies uses the pseudorandom function producing *n* bits of output, PRF-*n,* defined in 12.7.1.2.

In an infrastructure BSS, the IEEE 802.1X Authenticator MAC address (AA) and the AP's MAC address are the same, and the Supplicant's MAC address (SPA) and the STA's MAC address are equal. For the purposes of comparison, the MAC address is encoded as 6 octets, taken to represent an unsigned integer. The first octet of the MAC address shall be used as the most significant octet. The bit numbering conventions in 9.2.2 shall be used within each octet. This results in a 48-bit unsigned integer labeled B0 (least significant) to B47 (most significant), where the I/G bit is in B40.

An RSNA STA shall support at least one pairwise key for any <TA,RA> pair for use with enhanced data cryptographic encapsulation mechanisms. The <TA,RA> identifies the pairwise key, which does not correspond to any WEP key identifier.

In a a mixed environment, an AP may simultaneously communicate with some STAs using WEP with shared WEP keys and to STAs using enhanced data cryptographic encapsulation mechanisms with pairwise keys. The STAs running WEP use default keys 0–3 for shared WEP keys; the important point here is that WEP can still use WEP default key 0. The AP might be configured to use the WEP key in WEP default key 0 for WEP; if the AP is configured in this way, STAs that cannot support WEP default key 0 simultaneously with a TKIP pairwise key shall specify the No Pairwise subfield in the RSN Capabilities field. If an AP is configured to use WEP default key 0 as a WEP key and a "No Pairwise" STA associates, the AP shall not set the Install bit in the 4-way handshake. In other words, the STA does not install a pairwise temporal key and instead uses WEP default key 0 for all traffic.

NOTE—The behavior of "No Pairwise" STAs is intended only to support the migration of WEP to RSNA.

TKIP STAs in a mixed environment are expected to support a single pairwise key either by using a key mapping key or by mapping to default key 0. The AP uses a pairwise key for individually addressed traffic between the AP and the STA. If a key mapping key is available, the <RA,TA> pair identifies the key; if there is no key mapping key, then the default key 0 is used because the key index in the message is 0.

A STA that cannot support TKIP keys and WEP default key 0 simultaneously advertises this deficiency by setting the No Pairwise subfield in the RSNE it sends in the (Re)Association Request frame to the AP. In response, the AP sets the Install bit to 0 in message 3 of the 4-way handshake to notify the STA not to install the pairwise key. The AP instead sends the WEP shared key to the STA to be plumbed as the WEP default key 0; this key is then used with WEP to send and receive individually addressed traffic between the AP and the STA.

The TKIP STA that has this limitation might not know that it will be forced to use WEP for all transmissions until it has associated with the AP and been given the keys to use. (The STA cannot know that the AP has been configured to use WEP default key 0 for WEP communication.) If this does not satisfy the security policy configured at the STA, the STA's only recourse is to disassociate and try a different AP.

STAs using enhanced data cryptographic encapsulation mechanisms in a TSN shall support pairwise keys and WEP default key 0 simultaneously. It is invalid for the STA to negotiate the No Pairwise subfield when an enhanced data cryptographic encapsulation mechanism other than TKIP is one of the configured ciphers.

12.7.1.2 PRF

A PRF is used in a number of places in this standard. Depending on its use, it may need to output 128, 192, 256, 384, 512, or 704 bits. This subclause defines six functions:

— PRF-128, which outputs 128 bits
— PRF-192, which outputs 192 bits
— PRF-256, which outputs 256 bits
— PRF-384, which outputs 384 bits
— PRF-512, which outputs 512 bits
— PRF-704, which outputs 704 bits

In the following, K is a key; A is a unique label for each different purpose of the PRF, treated as an ASCII string; B is a variable-length string; Y is a single octet containing 0; X is a single octet containing the loop parameter i:

H-SHA-1(K, A, B, X) ← HMAC-SHA-1(K, A || Y || B || X)

PRF(K, A, B, Len)
    for $i \leftarrow 0$ **to** $\left\lceil \frac{Len}{160} \right\rceil$ **do**
        $R \leftarrow R$ || H-SHA-1(K, A, B, i)
    return L(R, 0, Len)

PRF-128(K, A, B) = PRF(K, A, B, 128)
PRF-192(K, A, B) = PRF(K, A, B, 192)
PRF-256(K, A, B) = PRF(K, A, B, 256)
PRF-384(K, A, B) = PRF(K, A, B, 384)
PRF-512(K, A, B) = PRF(K, A, B, 512)

When the negotiated AKM is 00-0F-AC:5, 00-0F-AC:6, or 00-0F-AC:11, the KDF specified in 12.7.1.7.2 shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context, and the PRF functions are defined as follows:

PRF-128(K, A, B) = KDF-SHA-256-128(K, A, B)
PRF-192(K, A, B) = KDF-SHA-256-192(K, A, B)

PRF-256(K, A, B) = KDF-SHA-256-256(K, A, B)
PRF-384(K, A, B) = KDF-SHA-256-384(K, A, B)
PRF-512(K, A, B) = KDF-SHA-256-512(K, A, B)

When the negotiated AKM is 00-0F-AC:12, the KDF specified in 12.7.1.7.2 shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context, and the PRF function is defined as follows:

PRF-704(K, A, B) = KDF-SHA-384-704(K, A, B)

When the negotiated AKM is 00-0F-AC:13, the KDF specified in 12.7.1.7.2 shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context, and the PRF functions are defined as follows:

PRF-384(K, A, B) = KDF-SHA-384-384(K, A, B)
PRF-512(K, A, B) = KDF-SHA-384-512(K, A, B)
PRF-704(K, A, B) = KDF-SHA-384-704(K, A, B)

12.7.1.3 Pairwise key hierarchy

Except when preauthentication is used, the pairwise key hierarchy utilizes PRF-384, PRF-512, or PRF-704 to derive session-specific keys from a PMK, as depicted in Figure 12-28. When using AKM suite selector 00-0F-AC:12, the length of the PMK, PMK_bits, shall be 384 bits. With all other AKM suite selectors, the length of the PMK, PMK_bits, shall be 256 bits. The pairwise key hierarchy takes a PMK and generates a PTK. The PTK is partitioned into KCK, KEK, and a temporal key, which is used by the MAC to protect individually addressed communication between the Authenticator's and Supplicant's respective STAs. PTKs are used between a single Supplicant and a single Authenticator.

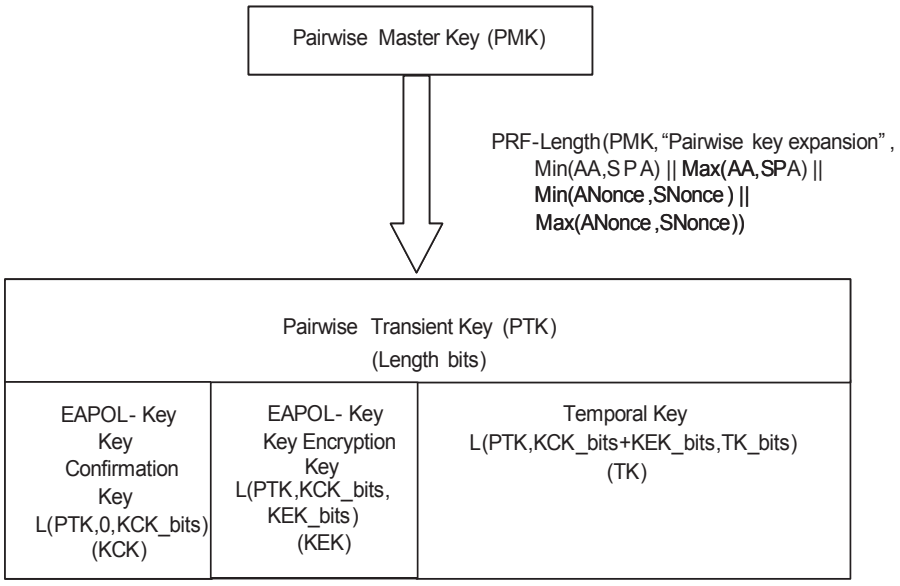

Figure 12-28—Pairwise key hierarchy

When not using a PSK, the PMK is derived from the MSK. The PMK shall be computed as the first PMK_bits bits (bits 0 to PMK_bits–1) of the MSK: PMK = L(MSK, 0, PMK_bits).

The PTK shall not be used longer than the PMK lifetime as determined by the minimum of the PMK lifetime indicated by the AS, e.g., Session-Timeout + dot1xAuthTxPeriod or from dot11RSNAConfigPMK-Lifetime. When RADIUS is used and the Session-Timeout attribute is not in the RADIUS Accept message, and if the key lifetime is not otherwise specified, then the PMK lifetime is infinite.

NOTE 1—If the protocol between the Authenticator (or AP) and AS is RADIUS, then the MS-MPPE-Recv-Key attribute (vendor-id = 17; see Section 2.4.3 in IETF RFC 2548 [B32]) is available to be used to transport the first 32 octets of the MSK to the AP, and the MS-MPPE-Send-Key attribute (vendor-id = 16; see Section 2.4.2 in IETF RFC 2548 [B32]) is available to be used to transport the remaining 32 octets of the MSK.

NOTE 2—When reauthenticating and changing the pairwise key, a race condition might occur when using TKIP. If a frame is received while MLME-SETKEYS.request primitive is being processed, the received frame might be decrypted with one key and the MIC checked with a different key. Two possible options to avoid this race condition are as follows: the frame might be checked against the old MIC key, and the received frames might be queued while the keys are changed.

NOTE 3—If the AKMP is RSNA-PSK, then a 256-bit PSK might be configured into the STA and AP or a pass-phrase might be configured into the Supplicant or Authenticator. The method used to configure the PSK is outside this standard, but one method is via user interaction. If a pass-phrase is configured, then a 256-bit key is derived and used as the PSK. In any RSNA-PSK method, the PSK is used directly as the PMK. Implementations might support different PSKs for each pair of communicating STAs.

Here, the following assumptions apply:

— SNonce is a random or pseudorandom value contributed by the Supplicant; its value is taken when a PTK is instantiated and is sent to the PTK Authenticator.

— ANonce is a random or pseudorandom value contributed by the Authenticator.

— The PTK shall be derived from the PMK by

PTK = PRF-Length(PMK, “Pairwise key expansion”, Min(AA,SPA) || Max(AA,SPA) || Min(ANonce,SNonce) || Max(ANonce,SNonce))

where Length = KCK_bits + KEK_bits + TK_bits. The values of KCK_bits and KEK_bits are AKM suite dependent and are listed in Table 12-8. The value of TK_bits is cipher-suite dependent and is defined in Table 12-4. The Min and Max operations for IEEE 802 addresses are with the address converted to a positive integer treating the first transmitted octet as the most significant octet of the integer. The nonces are encoded as specified in 9.2.2.

NOTE—The Authenticator and Supplicant normally derive a PTK only once per association. A Supplicant or an Authenticator use the 4-way handshake to derive a new PTK. Both the Authenticator and Supplicant create a new nonce value for each 4-way handshake instance.

— The KCK shall be computed as the first KCK_bits bits (bits 0 to KCK_bits–1) of the PTK:

KCK= L(PTK, 0, KCK_bits)

The KCK is used by IEEE Std 802.1X-2010 to provided data origin authenticity in the 4-way handshake and group key handshake messages.

— The KEK shall be computed as the next KEK_bits bits of the PTK:

KEK = L(PTK, KCK_bits, KEK_bits)

The KEK is used by the EAPOL-Key frames to provide data confidentiality in the 4-way handshake and group key handshake messages.

— The temporal key (TK) shall be computed as the next TK_bits bits of the PTK:

TK = L(PTK, KCK_bits+KEK_bits, TK_bits)

The EAPOL-Key state machines (see 12.7.10 and 12.7.11) use the MLME-SETKEYS.request primitive to configure the temporal key into the STA. The STA uses the temporal key with the pairwise cipher suite; interpretation of this value is cipher-suite-specific.

When the negotiated AKM is 00-0F-AC:5 or 00-0F-AC:6, the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-256(PMK, "PMK Name" || AA || SPA))

When the negotiated AKM is 00-0F-AC:11, the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-256(KCK, "PMK Name" || AA || SPA))

When the negotiated AKM is 00-0F-AC:12, and the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-384(KCK, "PMK Name" || AA || SPA))

Otherwise, the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-1(PMK, "PMK Name" || AA || SPA))

In all these cases, “PMK Name” is treated as an ASCII string.

When the PMKID is calculated for the PMKSA as part of preauthentication, the AKM has not yet been negotiated. In this case, the HMAC-SHA-1 based derivation is used for the PMKID calculation.

12.7.1.4 Group key hierarchy

The group temporal key (GTK) shall be a random number. The following is an example method for deriving a random GTK. Any other pseudorandom function, such as that specified in 12.7.1.2, could also be used.

A group master key (GMK) is an auxiliary key that may be used to derive a GTK at a time interval configured into the AP to reduce the exposure of data if the GMK is ever compromised.

The Authenticator might update the GTK for a number of reasons:

— The Authenticator might change the GTK on disassociation or deauthentication of a STA.
— An event within the SME might trigger a group key handshake.

Figure 12-29 depicts an example of a relationship among the keys of the group key hierarchy. In this model, the group key hierarchy takes a GMK and generates a GTK. The GTK is a temporal key, which is used to protect group addressed communication. GTKs are used between a single Authenticator and all Supplicants authenticated to that Authenticator. The Authenticator derives new GTKs when it needs to update the GTKs.

In this example, the following assumptions apply:

a) Group nonce (GNonce) is a random or pseudorandom value contributed by the IEEE 802.1X Authenticator.
b) The GTK is derived from the GMK by
c) GTK =PRF-Length(GMK, “Group key expansion”, AA || GNonce)
d) Length = TK_bits. The value of TK_bits is cipher-suite dependent and is defined in Table 12-4. AA is represented as an IEEE 802 address and GNonce as a bit string as defined in 9.2.2.
e) The temporal key (TK) is bits 0 to (TK_bits – 1) of the GTK:
TK =L(GTK, 0, TK_bits)
f) The EAPOL-Key state machines (see 12.7.10 and 12.7.11) configure the temporal key into a STA via the MLME-SETKEYS.request primitive. Its interpretation is cipher-suite-specific.

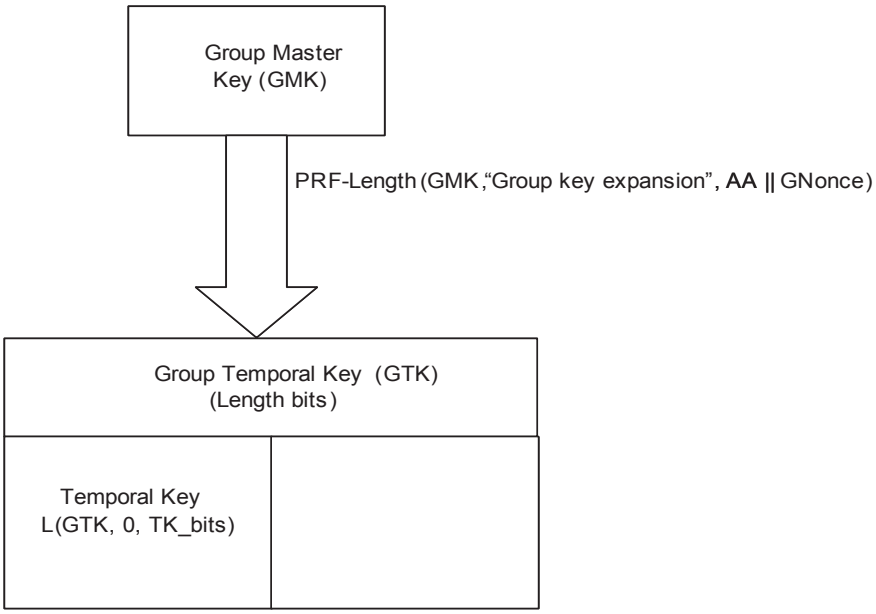

Figure 12-29—Group key hierarchy (informative)

12.7.1.5 Integrity group key hierarchy

The Authenticator shall select the IGTK as a random value each time it is generated.

The Authenticator may update the IGTK for any reason, including:

a) The disassociation or deauthentication of a STA.
b) An event within the SME that triggers a group key handshake.

The IGTK is configured via the MLME-SETKEYS.request primitive; see 6.3.19. IGTK configuration is described in the EAPOL-Key state machines; see 12.7.10 and 12.7.11.

The IPN is used to provide replay protection.

12.7.1.6 PeerKey key hierarchy

The station-to-station key hierarchy utilizes PRF-384 or PRF-512 to derive session-specific keys from an SMK, as depicted in Figure 12-30. The SMK shall be 256 bits. The pairwise key hierarchy takes an SMK and generates an STK. The STK is partitioned into SKCK, SKEK, and a temporal key, which is used by the MAC to protect individually addressed communication between the initiator and peer STAs. STKs are used between a single initiator STA and a single peer STA.

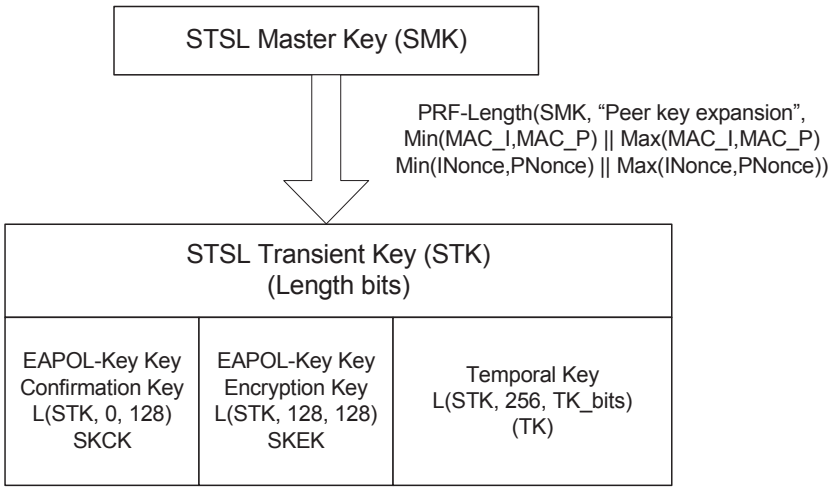

Figure 12-30—PeerKey hierarchy

The following apply and are depicted in Figure 12-30:

a) INonce is a random or pseudorandom value contributed by the initiator STA.

b) PNonce is a random or pseudorandom value contributed by the peer STA.

c) The STK shall be derived from the SMK by

STK =PRF-Length(SMK, "Peer key expansion", Min(MAC_I,MAC_P) || Max(MAC_I,MAC_P) || Min(INonce,PNonce) || Max(INonce,PNonce))

where Length = 256 + TK_bits. The value of TK_bits is cipher-suite dependent and is defined in Table 12-4. The Min and Max operations for IEEE 802 addresses are with the address converted to a positive integer treating the first transmitted octet as the most significant octet of the integer. For the Min and Max operations, nonces are encoded as specified in 9.2.2.

d) The SKCK shall be computed as the first 128 bits (bits 0–127) of the STK:

SKCK =L(STK, 0, 128)

The SKCK is used to provide data origin authenticity in the 4-way STK handshake.

e) The SKEK shall be computed as bits 128–255 of the STK:

SKEK =L(STK, 128, 128)

The SKEK is used by the EAPOL-Key frames to provide confidentiality in the 4-way STK handshake.

f) The temporal key (TK) shall be computed as bits 256 to (255 + TK_bits) of the STK:

TK =L(STK, 256, TK_bits)

The EAPOL-Key state machines (see 12.7.10 and 12.7.11) use the MLME-SETKEYS.request primitive to configure the temporal key into the STA. The STA uses the temporal key with the pairwise cipher suite; interpretation of this value is cipher-suite-specific.

When the negotiated AKM is 00-0F-AC:5 or 00-0F-AC:6, the SMK identifier is defined as

SMKID = Truncate-128(HMAC-SHA-256(SMK, "SMK Name" || PNonce || MAC_P || INonce || MAC_I))

Otherwise, the SMK identifier is defined as

SMKID = Truncate-128(HMAC-SHA-1(SMK, "SMK Name" || PNonce || MAC_P || INonce || MAC_I))

In both these cases, “SMK Name” is treated as an ASCII string.

12.7.1.7 FT key hierarchy

12.7.1.7.1 Overview

This subclause describes the FT key hierarchy and its supporting architecture. The FT key hierarchy is designed to allow a STA to make fast BSS transitions between APs without the need to perform an SAE or IEEE 802.1X authentication at every AP within the mobility domain.

The FT key hierarchy can be used with SAE, IEEE 802.1X authentication, or PSK authentication.

A three-level key hierarchy provides key separation between the key holders. The FT key hierarchy for the Authenticator is shown in Figure 12-31. An identical key hierarchy exists for the Supplicant, and identical functions are performed by the corresponding S0KH and S1KH.

The FT key hierarchy shown in Figure 12-31 consists of three levels whose keys are derived using the key derivation function (KDF) described in 12.7.1.7.2 as follows:

a) PMK-R0 – the first-level key of the FT key hierarchy. This key is derived as a function of the master session key (MSK) or PSK. It is stored by the PMK-R0 key holders, R0KH and S0KH.
b) PMK-R1 – the second-level key of the FT key hierarchy. This key is mutually derived by the S0KH and R0KH.
c) PTK – the third-level key of the FT key hierarchy that defines the IEEE 802.11 and IEEE 802.1X protection keys. The PTK is mutually derived by the PMK-R1 key holders, R1KH and S1KH.

As shown in Figure 12-31, the R0KH computes the PMK-R0 from the key obtained from SAE authentication (for the purposes of FT this key is identified as the Master PMK, or MPMK), from the PSK, or from the MSK resulting (per IETF RFC 3748 [B42]) from a successful IEEE 802.1X authentication between the AS and the Supplicant. Upon a successful authentication, the R0KH shall delete any prior PMK-R0 security association for this mobility domain pertaining to this S0KH. The R0KH shall also delete all PMK-R1 security associations derived from that prior PMK-R0 security association. The PMK-R1s are generated by the R0KH and are assumed to be delivered from the R0KH to the R1KHs within the same mobility domain. The PMK-R1s are used for PTK generation. Upon receiving a new PMK-R1 for an S0KH, an R1KH deletes the prior PMK-R1 security association and PTKSAs derived from the prior PMK-R1.

It is assumed by this standard that the PSK is specific to a single S0KH and a single R0KH.

The lifetime of the PMK-R0, PMK-R1, and PTK are bound to the lifetime of the MPMK, PSK, or MSK from which it was derived. For example, the AS may communicate the MSK lifetime with the MSK. If such an attribute is provided, the lifetime of the PMK-R0 shall be not more than the lifetime of the MSK. The lifetime of the PTK and PMK-R1 is the same as that of the PMK-R0. When the key lifetime expires, each key holder shall delete its respective PMK-R0, PMK-R1 or PTKSA.

The FT key hierarchy derives its keys using the KDF defined in 12.7.1.7.2 with separate labels to further distinguish derivations.

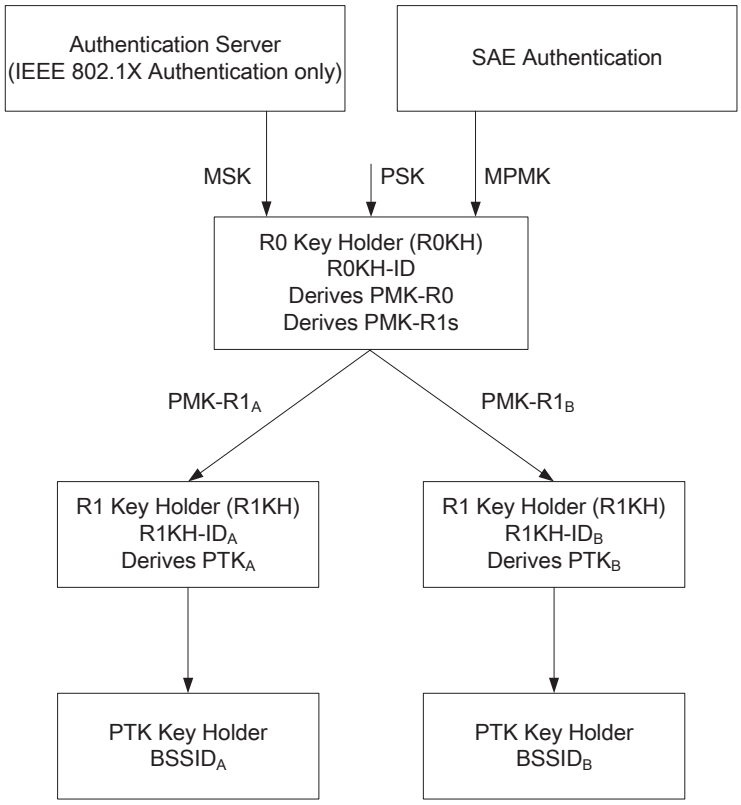

Figure 12-31—FT key hierarchy at an Authenticator

During a fast BSS transition, a STA shall negotiate the same pairwise cipher suite with target APs as was negotiated in the FT initial mobility domain association. Using the pairwise cipher suite selector value in the PMK-R1 security association received from the R0KH, the target AP shall verify that the same pairwise cipher suite selector is being used.

The distribution of keys from the R0KH to the R1KHs is outside the scope of this standard. It is assumed that the PMK-R1s are distributed from the R0KH to the R1KHs following the requirements specified in 13.2.2.

The PMK-R0 may be deleted by the R0KH after PMK-R1s have been derived. When the PMK-R0 is deleted, the R0KH needs only to maintain the PMK-R1 security associations.

12.7.1.7.2 Key derivation function (KDF)

The KDF is a variant of the pseudorandom function (PRF) defined in 12.7.1.2 and is defined as follows:

Output ← KDF-Hash-Length (K, Label, Context) where
Input: *K*, a derivation key whose length equals the block size of the hash function
Hash, a cryptographically strong hash function
Label, a string identifying the purpose of the keys derived using this KDF, treated as an ASCII string
Context, a bit string that provides context to identify the derived key
Length, the length of the derived key in bits
Output: a *Length*-bit derived key

result ← ""
iterations ← ⌈*Length*/*Hashlen*⌉
do *i* = 1 **to** *iterations*
    result ← *result* || HMAC-Hash(*K*, *i* || *Label* || *Context* || *Length*)
od
return first *Length* bits of *result,* and irretrievably delete the other bits

where

i and *Length* are encoded as 16-bit unsigned integers, represented using the bit ordering conventions of 9.2.2
K, *Label*, and *Context* are bit strings and are represented using the ordering conventions of 9.2.2
Hashlen is the length, in bits, of the digest produced by Hash

12.7.1.7.3 PMK-R0

The first-level key in the FT key hierarchy, PMK-R0, is derived using the KDF defined in 12.7.1.7.2. The PMK-R0 is the first level keying material used to derive the next level keys (PMK-R1s):

R0-Key-Data = KDF-Hash-Length(XXKey, "FT-R0", SSIDlength || SSID || MDID || R0KHlength || R0KH-ID || S0KH-ID)
PMK-R0 = L(R0-Key-Data, 0, Q)
PMK-R0Name-Salt = L(R0-Key-Data, Q, 128)
Length = Q + 128

where

— KDF-Hash-Length is the KDF as defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133).
— If the AKM negotiated is 00-0F-AC:3, then Q shall be 256, and XXKey shall be the second 256 bits of the MSK (which is derived from the IEEE 802.1X authentication), i.e., XXKey = L(MSK, 256, 256). If the AKM negotiated is 00-0F-AC:4, then Q shall be 256, and XXKey shall be the PSK. If the AKM negotiated is 00-0F-AC:9, then Q shall be 256, and XXKey shall be the MPMK generated as the result of SAE authentication. If the AKM negotiated is 00-0F-AC:13, then Q shall be 384, and XXKey shall be the first 384 bits of the MSK (which is derived from the IEEE 802.1X authentication), i.e., XXKey = L(MSK, 0, 384).
— SSIDlength is a single octet whose value is the number of octets in the SSID.
— SSID is the service set identifier, a variable-length sequence of octets, as it appears in the Beacon and Probe Response frames.
— MDID is the Mobility Domain Identifier field from the Mobile Domain element (MDE) that was used during FT initial mobility domain association.
— R0KHlength is a single octet whose value is the number of octets in the R0KH-ID.
— R0KH-ID is the identifier of the holder of PMK-R0 in the Authenticator.
— S0KH-ID is the Supplicant's MAC address (SPA).

The PMK-R0 is referenced and named as follows:
PMKR0Name = Truncate-128(SHA-256("FT-R0N" || PMK-R0Name-Salt))

where

— "FT-R0N" is treated as an ASCII string.

The PMKR0Name is used to identify the PMK-R0.

12.7.1.7.4 PMK-R1

The second-level key in the FT key hierarchy, PMK-R1, is a key used to derive the PTK. The PMK-R1 is derived using the KDF defined in 12.7.1.7.2:

PMK-R1 = KDF-Hash-Length(PMK-R0, "FT-R1", R1KH-ID || S1KH-ID)

where

— KDF-Hash-Length is the KDF as defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133) to generate a key whose length is equal to the length of the hash algorithm's digest.
— PMK-R0 is the first level key in the FT key hierarchy.
— R1KH-ID is a MAC address of the holder of the PMK-R1 in the Authenticator of the AP.
— S1KH-ID is the SPA.

The PMK-R1 is referenced and named as follows:

PMKR1Name = Truncate-128(SHA-256("FT-R1N" || PMKR0Name || R1KH-ID || S1KH-ID))

where

— "FT-R1N" is treated as an ASCII string.

PMKR1Name is used to identify the PMK-R1.

12.7.1.7.5 PTK

The third-level key in the FT key hierarchy is the PTK. This key is mutually derived by the S1KH and the R1KH used by the target AP, with the key length being a function of the negotiated cipher suite as defined by Table 12-4 in 12.7.2.

Using the KDF defined in 12.7.1.7.2, the PTK derivation is as follows:

PTK = KDF-Hash-Length(PMK-R1, "FT-PTK", SNonce || ANonce || BSSID || STA-ADDR)

where

— KDF-Hash-Length is the KDF as defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133).
— PMK-R1 is the key that is shared between the S1KH and the R1KH.
— SNonce is a 256-bit random bit string contributed by the S1KH.
— ANonce is a 256-bit random bit string contributed by the R1KH.
— STA-ADDR is the non-AP STA's MAC address.
— BSSID is the BSSID of the target AP.
— Length is the total number of bits to derive, i.e., number of bits of the PTK. The length is dependent on the negotiated cipher suites and AKM suites as defined by Table 12-4 in 12.7.2 and Table 12-8 in 12.7.3.

Each PTK has three component keys, KCK, KEK, and a temporal key, derived as follows:

The KCK shall be computed as the first KCK_bits bits (bits 0 to KCK_bits–1) of the PTK:

KCK = L(PTK, 0, KCK_bits).

The KCK is used to provide data origin authenticity in EAPOL-Key frames, as defined in 12.7.2, and in the FT authentication sequence, as defined in 13.8.

The KEK shall be computed as the next KEK_bits of the PTK:

KEK = L(PTK, KCK_bits, KEK_bits)

The KEK is used to provide data confidentiality for certain fields (KeyData) in EAPOL-Key frames, as defined in 12.7.2, and in the FT authentication sequence, as defined in 13.8.

The temporal key (TK) shall be computed as the next TK_bits (see Table 12-4) bits of the PTK:

TK = L(PTK, KCK_bits+KEK_bits, TK_bits)

For vendor-specific cipher suites, the length of the temporal key (and the value of Length) depend on the vendor-specific algorithm.

The temporal key is configured into the STA by the SME through the use of the MLME-SETKEYS.request primitive. The STA uses the temporal key with the pairwise cipher suite; interpretation of this value is specific to the cipher suite.

The PTK is referenced and named as follows:

PTKName = Truncate-128(SHA-256(PMKR1Name || “FT-PTKN” || SNonce || ANonce || BSSID || STA-ADDR))

where

— "FT-PTKN" is treated as an ASCII string.

The PTKName is used to identify the PTK.

12.7.2 EAPOL-Key frames

IEEE Std 802.11 uses EAPOL-Key frames to exchange information between STAs’ Supplicants and Authenticators. These exchanges result in cryptographic keys and synchronization of security association state. EAPOL-Key frames are used to implement three different exchanges:

— 4-way handshake, to confirm that the PMK between associated STAs is the same and live and to transfer the GTK to the STA.
— Group key handshake, to update the GTK at the STA.
— PeerKey initial SMK handshake to deliver the SMK and final 4-way STK handshake to deliver the STK to the initiating and peer STAs.

When priority processing of Data frames is supported, an SME should send EAPOL-Key frames at the highest priority.

The RSNA key descriptor used by IEEE Std 802.11 maps to the IEEE 802.1X Key Descriptor as described in this subclause.

The bit and octet convention for fields in the EAPOL-Key frame are defined in 11.2 of IEEE Std 802.1X-2010. EAPOL-Key frames containing invalid field values shall be silently discarded. Figure 12-32 depicts the format of an EAPOL-Key frame.

<table>
<tr><td>Protocol Version – 1 octet</td><td>Packet Type – 1 octet</td><td>Packet Body Length – 2 octets</td></tr>
<tr><td colspan="3">Descriptor Type – 1 octet</td></tr>
<tr><td colspan="2">Key Information – 2 octets</td><td>Key Length – 2 octets</td></tr>
<tr><td colspan="3">Key Replay Counter – 8 octets</td></tr>
<tr><td colspan="3">Key Nonce – 32 octets</td></tr>
<tr><td colspan="3">EAPOL - Key IV – 16 octets</td></tr>
<tr><td colspan="3">Key RSC – 8 octets</td></tr>
<tr><td colspan="3">Reserved - 8 octets</td></tr>
<tr><td colspan="3">Key MIC – variable</td></tr>
<tr><td colspan="2">Key Data Length – 2 octets</td><td>Key Data – n octets</td></tr>
</table>

Figure 12-32—EAPOL-Key frame

The fields of an EAPOL-Key frame body are as follows:

a) **Descriptor Type.** This field is 1 octet and has a value defined by IEEE Std 802.1X-2010, identifying the IEEE 802.11 key descriptor.

b) **Key Information**. This field is 2 octets and specifies characteristics of the key. See Figure 12-33.

<table>
<tr><th>B0 B2</th><th>B3</th><th>B4 B5</th><th>B6</th><th>B7</th><th>B8</th><th>B9</th><th>B10</th><th>B11</th><th>B12</th><th>B13</th><th>B14 B15</th></tr>
<tr><td>Key Descriptor Version</td><td>Key Type</td><td>Reserved</td><td>Install</td><td>Key Ack</td><td>Key MIC</td><td>Se-cure</td><td>Error</td><td>Request</td><td>Encrypted Key Data</td><td>SMK Message</td><td>Reserved</td></tr>
</table>

Figure 12-33—Key Information bit layout

The bit convention used is as in 11.2 of IEEE Std 802.1X-2010. The subfields of the Key Information field are as follows:

1) Key Descriptor Version (bits 0–2) shall be set to 0 on all transmitted EAPOL-Key frames except under the following circumstances:
   i) The value 1 shall be used for all EAPOL-Key frames to a STA when the negotiated AKM is 00-0F-AC:1 or 00-0F-AC:2 and the pairwise cipher is TKIP or "Use group cipher suite" for Key Descriptor 1. This value indicates the following:
      — HMAC-MD5 is the EAPOL-Key MIC.
      — ARC4 is the EAPOL-Key encryption algorithm used to protect the Key Data field.
      — The MIC is 16 octets.
   ii) The value 2 shall be used for all EAPOL-Key frames to a STA when the negotiated AKM is 00-0F-AC:1 or 00-0F-AC:2 and either the pairwise or the group cipher is an enhanced

data cryptographic encapsulation mechanism other than TKIP for Key Descriptor 2. This value indicates the following:

- HMAC-SHA-1-128 is the EAPOL-Key MIC. HMAC is defined in IETF RFC 2104; and SHA-1, by FIPS PUB 180-4. The output of the HMAC-SHA-1 shall be truncated to its 128 MSBs (octets 0–15 of the digest output by HMAC-SHA-1), i.e., the last four octets generated shall be irretrievably deleted).
- The NIST AES key wrap is the EAPOL-Key encryption algorithm used to protect the Key Data field. IETF RFC 3394 defines the NIST AES key wrap algorithm.
- The MIC is 16 octets.

iii) The value 3 shall be used for all EAPOL-Key frames to a STA when the negotiated AKM is 00-0F-AC:3, 00-0F-AC:4, 00-0F-AC:5, or 00-0F-AC:6. This value indicates the following:

- AES-128-CMAC is the EAPOL-Key MIC. AES-128-CMAC is defined by NIST Special Publication 800-38B and also found in IETF RFC 4493. The output of the AES-128-CMAC shall be 128 bits.
- The NIST AES key wrap is the EAPOL-Key encryption algorithm used to protect the Key Data field. IETF RFC 3394 defines the NIST AES key wrap algorithm.
- The MIC is 16 octets.

2) Key Type (bit 3) specifies whether this EAPOL-Key frame is part of a 4-way handshake deriving a PTK.

i) The value 0 (Group/SMK) indicates the message is not part of a PTK derivation.

ii) The value 1 (Pairwise) indicates the message is part of a PTK derivation.

3) Reserved (bits 4–5).

4) Install (bit 6).

i) If the value of Key Type (bit 3) is 1, then for the Install bit,

- The value 1 means the IEEE 802.1X component shall configure the temporal key derived from this message into its IEEE 802.11 STA.
- The value 0 means the IEEE 802.1X component shall not configure the temporal key into the IEEE 802.11 STA.

ii) If the value of Key Type (bit 3) is 0, then this bit is reserved.

5) Key Ack (bit 7) is set to 1 in messages from the Authenticator if an EAPOL-Key frame is required in response to this message and is 0 otherwise. The Supplicant's response to this message shall use the same replay counter as this message.

6) Key MIC (bit 8) is set to 1 if a MIC is in this EAPOL-Key frame and is set to 0 if this message contains no MIC.

7) Secure (bit 9) is set to 1 once the initial key exchange is complete.

The Authenticator shall set the Secure bit to 0 in all EAPOL-Key frames sent before the Supplicant has the PTK and the GTK. The Authenticator shall set the Secure bit to 1 in all EAPOL-Key frames it sends to the Supplicant containing the last key needed to complete the Supplicant's initialization.

The Supplicant shall set the Secure bit to 0 in all EAPOL-Key frames it sends before it has the PTK and the GTK and before it has received an EAPOL-Key frame from the Authenticator with the Secure bit equal to 1 (this should be before receiving message 3 of the 4-way handshake). The Supplicant shall set the Secure bit to 1 in all EAPOL-Key frames sent after this until it loses the security association it shares with the Authenticator.

8) Error (bit 10) is set by a Supplicant to report that a MIC failure occurred in a TKIP MSDU or SMK handshake failure. In case of a MIC failure, a Supplicant shall set this bit to 1 only when

the Request (bit 11) is 1. When the SMK Message bit is 1, Error shall be set to 1 to indicate the key data field contains an Error KDE.

9) Request (bit 11) is set to 1 by a Supplicant to request that the Authenticator initiate either a 4-way handshake or group key handshake, is set to 1 by a Supplicant in a Michael MIC Failure Report, and is set to 1 by the STSL peer STA to request initiator STA rekeying of the STK. The Supplicant shall not set this bit to 1 in on-going 4-way handshakes, i.e., the Key Ack bit (bit 7) shall not be set to 1 in any message in which the Request bit is 1. The Authenticator shall never set this bit to 1.

   In a Michael MIC Failure Report, setting the bit is not a request to initiate a new handshake. However, the recipient may initiate a new handshake on receiving such a message.

   If an EAPOL-Key frame in which the Request bit is 1 has a key type of Pairwise, the Authenticator shall initiate a 4-way handshake. If the EAPOL-Key frame in which the Request bit is 1 has a key type of Group/SMK, the Authenticator shall change the GTK, initiate a 4-way handshake with the Supplicant, and then execute the group key handshake to all Supplicants.

   If an EAPOL-Key frame in which the Request bit is 1 has the SMK Message bit equal to 1, the initiator STA shall take appropriate action to create a new STK (based on 12.7.8).

10) Encrypted Key Data (bit 12) is set to 1 if the Key Data field is encrypted and is set to 0 if the Key Data field is not encrypted. This subfield shall be set to 1, and the Key Data field shall be encrypted, if any key material (e.g., GTK or SMK) is included in the frame.

11) SMK Message (bit 13) specifies whether this EAPOL-Key frame is part of an SMK handshake. If the SMK handshake is not supported, the SMK Message subfield is reserved.

12) Reserved (bits 14–15).

c) **Key Length**. This field is 2 octets in length, represented as an unsigned integer. The value defines the length in octets of the pairwise temporal key. See Table 12-4.

Table 12-4—Cipher suite key lengths

Cipher suite	Key length (octets)	TK_bits (bits)
WEP-40	5	40
WEP-104	13	104
TKIP	32	256
CCMP-128	16	128
BIP-CMAC-128	16	128
GCMP-128	16	128
GCMP-256	32	256
CCMP-256	32	256
BIP-GMAC-128	16	128
BIP-GMAC-256	32	256
BIP-CMAC-256	32	256

d) **Key Replay Counter**. This field is 8 octets, represented as an unsigned integer, and is initialized to 0 when the PMK is established. The Supplicant shall use the key replay counter in the received

EAPOL-Key frame when responding to an EAPOL-Key frame. It carries a sequence number that the protocol uses to detect replayed EAPOL-Key frames.

The Supplicant and Authenticator shall track the key replay counter per security association. The key replay counter shall be initialized to 0 on (re)association. The Authenticator shall increment the key replay counter on each successive EAPOL-Key frame.

When replying to a message from the Authenticator, the Supplicant shall use the Key Replay Counter field value from the last valid EAPOL-Key frames received from the Authenticator. The Authenticator should use the key replay counter to identify invalid messages to silently discard. The Supplicant should also use the key replay counter and ignore EAPOL-Key frames with a Key Replay Counter field value smaller than or equal to any received in a valid message. The local Key Replay Counter field should not be updated until after the EAPOL-Key MIC is checked and is found to be valid. In other words, the Supplicant never updates the Key Replay Counter field for message 1 in the 4-way handshake, as it includes no MIC. This implies the Supplicant needs to allow for retransmission of message 1 when checking for the key replay counter of message 3.

The Supplicant shall maintain a separate key replay counter for sending EAPOL-Key request frames to the Authenticator; the Authenticator also shall maintain a separate replay counter to filter received EAPOL-Key request frames.

NOTE—The key replay counter does not play any role beyond a performance optimization in the 4-way handshake. In particular, replay protection is provided by selecting a never-before-used nonce value to incorporate into the PTK. It does, however, play a useful role in the group key handshake.

e) **Key Nonce**. This field is 32 octets. It conveys the ANonce from the Authenticator and the SNonce from the Supplicant. It may contain 0 if a nonce is not required to be sent.

f) **EAPOL-Key IV**. This field is 16 octets. It contains the IV used with the KEK. It shall contain 0 when an IV is not required. It should be initialized by taking the current value of the global key counter (see 12.7.11) and then incrementing the counter. Note that only the lower 16 octets of the counter value are used.

g) **Key RSC**. This field is 8 octets in length. It contains the receive sequence counter (RSC) for the GTK being installed. It is used in message 3 of the 4-way handshake and message 1 of the group key handshake, where it is used to synchronize the IEEE 802.11 replay state. It may also be used in the Michael MIC Failure Report frame, to report the TSC field value of the frame experiencing a MIC failure. It shall contain 0 in other messages. The Key RSC field gives the current message number for the GTK, to allow a STA to identify replayed MPDUs. If the Key RSC field value is less than 8 octets in length, the remaining octets shall be set to 0. The least significant octet of the TSC or PN should be in the first octet of the Key RSC field. The encoding of the Key RSC field is defined in Table 12-5.

Table 12-5—Key RSC field

KeyRSC 0	KeyRSC 1	KeyRSC 2	KeyRSC 3	KeyRSC 4	KeyRSC 5	KeyRSC 6	KeyRSC 7
TSC0	TSC1	TSC2	TSC3	TSC4	TSC5	0	0
PN0	PN1	PN2	PN3	PN4	PN5	0	0

For WEP, the Key RSC field is reserved.

h) **Key MIC**. The EAPOL Key MIC is a MIC of the EAPOL-Key frames, from and including the EAPOL protocol version field to and including the Key Data field, calculated with the Key MIC field set to 0. If the Encrypted Key Data subfield (of the Key Information field) is 1, the Key Data field is encrypted prior to computing the MIC. The length of this field depends on the negotiated AKM as defined in 12.7.3.

i) **Key Data Length**. This field is 2 octets in length, taken to represent an unsigned integer. This represents the length of the Key Data field in octets. If the Encrypted Key Data subfield (of the Key Information field) is 1, the length is the length of the Key Data field after encryption, including any padding.

j) **Key Data**. This field is a variable-length field that is used to include any additional data required for the key exchange that is not included in the fields of the EAPOL-Key frame. The additional data may be zero or more element(s) (such as the RSNE) and zero or more key data cryptographic encapsulation(s) (KDEs) (such as GTK(s) or PMKID(s)). Elements sent in the Key Data field include the Element ID and Length subfields. KDEs shall be encapsulated using the format in Figure 12-34.

	Type (0xdd)	Length	OUI	Data Type	Data
Octets:	1	1	3	1	(Length – 4)

Figure 12-34—KDE format

The Type field shall be set to 0xdd. The Length field specifies the number of octets in the OUI, Data Type, and Data fields. The OUI field contains either an OUI or CID. The order of the OUI field is described in 9.2.2.

Table 12-6 lists the KDE selectors defined by this standard.

Table 12-6—KDE

OUI	Data type	Meaning
00-0F-AC	0	Reserved
00-0F-AC	1	GTK KDE
00-0F-AC	2	Reserved
00-0F-AC	3	MAC address KDE
00-0F-AC	4	PMKID KDE
00-0F-AC	5	SMK KDE
00-0F-AC	6	Nonce KDE
00-0F-AC	7	Lifetime KDE
00-0F-AC	8	Error KDE
00-0F-AC	9	IGTK KDE
00-0F-AC	10	Key ID KDE
00-0F-AC	11	Multi-band GTK KDE
00-0F-AC	12	Multi-band Key ID KDE
00-0F-AC	13–255	Reserved
Other OUI or CID	Any	Vendor specific

A STA shall ignore any elements and KDEs it does not understand.

If the Encrypted Key Data subfield (of the Key Information field) is 1, the entire Key Data field shall be encrypted. If the Key Data field uses the NIST AES key wrap, then the Key Data field shall be padded before encrypting if the key data length is less than 16 octets or if it is not a multiple of 8.

The padding consists of appending a single octet 0xdd followed by zero or more 0x00 octets. When processing a received EAPOL-Key frame, the receiver shall ignore this trailing padding. Key Data fields that are encrypted, but do not contain the GroupKey or SMK KDE, shall be accepted.

If the GroupKey or SMK KDE is included in the Key Data field, but the Key Data field is not encrypted, the EAPOL-Key frames shall be ignored.

The format of the GTK KDE is shown in Figure 12-35

Key ID	Tx	Reserved	Reserved	GTK
bits 0–1	bit 2	bit 3–7	1 octet	(Length – 6) octets

Figure 12-35—GTK KDE format

If the value of the Tx field is 1, then the IEEE 802.1X component shall configure the temporal key derived from this KDE into its IEEE 802.11 STA for both transmission and reception.

If the value of the Tx field is 0, then the IEEE 802.1X component shall configure the temporal key derived from this KDE into its IEEE 802.11 STA for reception only.

The format of the MAC address KDE is shown in Figure 12-36.

Figure 12-36—MAC address KDE format

The format of the PMKID KDE is shown in Figure 12-37.

Figure 12-37—PMKID KDE format

The format of the SMK KDE is shown in Figure 12-38.

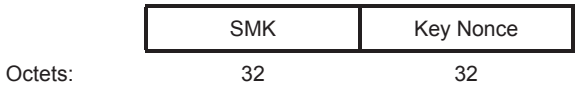

Figure 12-38—SMK KDE format

The format of the Nonce KDE is shown in Figure 12-39.

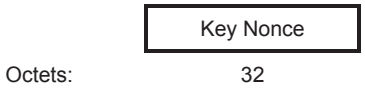

Figure 12-39—Nonce KDE format

The format of the Lifetime KDE is shown in Figure 12-40. The Key Lifetime value is expressed in seconds and uses big endian octet order.

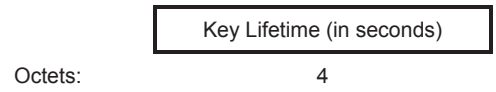

Figure 12-40—Lifetime KDE format

The format of the Error KDE is shown in Figure 12-41. The Error Type field is in big endian octet order. Table 12-7 shows different values of SMK error types.

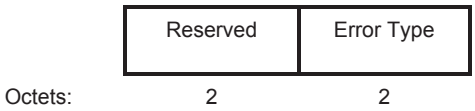

Figure 12-41—Error KDE format

Table 12-7—SMK error types

Error name	Error type	Error meaning
ERR_STA_NR	1	STA is not reachable from AP. See 12.7.8.5.2.
ERR_STA_NRSN	2	STA to AP secure network not present. See 12.7.8.5.3.
ERR_CPHR_NS	3	Cipher suites not supported. See 12.7.8.5.4.
ERR_NO_STSL	4	No STSL session present. See 12.7.8.5.5.

The format of the IGTK KDE is shown in Figure 12-42. The IPN corresponds to the last packet number used by the broadcast/multicast transmitter, and it is used by the receiver as the initial value for the BIP replay counter.

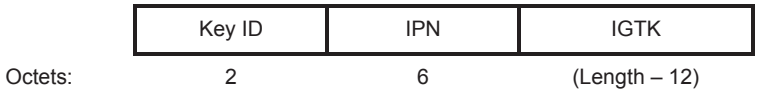

Figure 12-42—IGTK KDE format

The following EAPOL-Key frames are used to implement the three different exchanges:

— **4-way handshake message 1** is an EAPOL-Key frame with the Key Type subfield equal to 1. The Key Data field shall contain a PMKID for the PMK that is being used in this key derivation and need not be encrypted.

— **4-way handshake message 2** is an EAPOL-Key frame with the Key Type subfield equal to 1. The Key Data field shall contain an RSNE and need not be encrypted.

An ESS Supplicant's SME shall insert the RSNE it sent in its (Re)Association Request frame. The RSNE is included as transmitted in the Management frame. On receipt of message 2, the Authenticator's SME shall validate the selected security configuration against the RSNE received in the IEEE 802.11 (Re)Association Request.

An IBSS Supplicant's SME shall insert an RSNE containing a selected pairwise cipher suite. The Authenticator's SME shall validate that the pairwise cipher suite selected is one of its configured cipher suites and that the group cipher suite and AKM are consistent.

— **4-way handshake message 3** is an EAPOL-Key frame with the Key Type subfield equal to 1. The Key Data field shall contain one or two RSNEs. If a group cipher has been negotiated, this field shall also include a GTK. This field shall be encrypted if a GTK is included.

An Authenticator's SME shall insert the RSNE it sent in its Beacon or Probe Response frame. The Supplicant's SME shall validate the selected security configuration against the RSNE received in message 3. If the second optional RSNE is present, the STA shall either use that cipher suite with its pairwise key or deauthenticate. In either case, if the values do not match, then the receiver shall consider the RSNE modified and shall use the MLME-DEAUTHENTICATE.request primitive to break the association. A security error should be logged at this time.

It may happen, for example, that a Supplicant selects a pairwise cipher suite which is advertised by an AP, but which policy disallows for this particular STA. An Authenticator may, therefore, insert a second RSNE to overrule the STA's selection. An Authenticator's SME shall insert the second RSNE, after the first RSNE, only for this purpose. The pairwise cipher suite in the second RSNE included shall be one of the ciphers advertised by the Authenticator. All other fields in the second RSNE shall be identical to the first RSNE.

A GTK shall be included and the unencrypted length of the GTK is six less than the length of the GTK KDE in octets. The entire Key Data field shall be encrypted as specified by the Key Descriptor Version.

— **4-way handshake message 4** is an EAPOL-Key frame with the Key Type subfield equal to 1. The Key Data field can be empty.

— **Group key handshake message 1** is an EAPOL-Key frame with the Key Type subfield equal to 0. The Key Data field shall contain a GTK KDE and shall be encrypted.

— **Group key handshake message 2** is an EAPOL-Key frame with the Key Type subfield equal to 0. The Key Data field can be empty.

PeerKey handshake messages use EAPOL-Key frames as defined in 12.7.8.

The key wrap algorithm selected depends on the negotiated AKM as defined in 12.7.3.

The format of the Key ID KDE is shown in Figure 12-43.

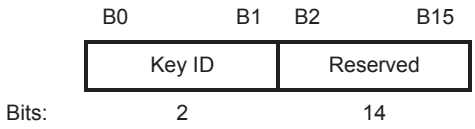

Figure 12-43—Key ID KDE

The Key ID field contains the Authenticator selected Key ID.

The format of the Multi-band GTK KDE is shown in Figure 12-44.

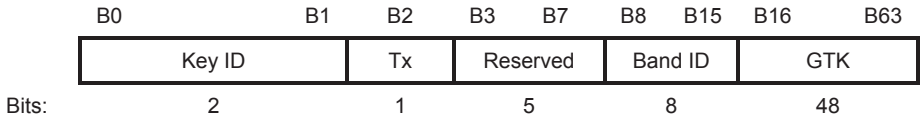

Figure 12-44—Multi-band GTK KDE

The definitions of the Key ID, Tx, and GTK fields are the same as in the GTK KDE described above.

The Band ID field contains the identification of the frequency band (see 9.4.1.46).

The format of the Multi-band Key ID KDE is shown in Figure 12-45.

Figure 12-45—Multi-band Key ID KDE

The definitions of the Key ID and Band ID fields are the same as in the Multi-band GTK KDE described above.

12.7.3 EAPOL-Key frame construction and processing

EAPOL-Key frames are constructed and processed according to the AKM negotiated at association time. The negotiated AKM determines what algorithm is used to construct and verify a MIC, the size of the MIC, and the algorithm used to wrap and unwrap the Key Data field.

Table 12-8 indicates the particular algorithms to use when constructing and processing EAPOL-Key frames. The AKM of "Deprecated" indicates an AKM of 00-0F-AC:1 or 00-0F-AC:2 when either TKIP or "Use group cipher suite" is the negotiated pairwise cipher. For all other AKMs the negotiated pairwise cipher suite does not influence the algorithms used to process EAPOL-Key frames.

Table 12-8—Integrity and key-wrap algorithms

AKM	Integrity algorithm	KCK_bits	Size of MIC	Key-wrap algorithm	KEK_bits
00-0F-AC:1	HMAC-SHA-1-128	128	16	NIST AES Key Wrap	128
00-0F-AC:2	HMAC-SHA-1-128	128	16	NIST AES Key Wrap	128
00-0F-AC:3	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:4	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:5	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:6	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:8	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:9	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:11	HMAC-SHA-256	128	16	NIST AES Key Wrap	128
00-0F-AC:12	HMAC-SHA-384	192	24	NIST AES Key Wrap	256
00-0F-AC:13	HMAC-SHA-384	192	24	NIST AES Key Wrap	256

12.7.4 EAPOL-Key frame notation

The following notation is used throughout the remainder of 12.7 and 13.4 to represent EAPOL-Key frames:

EAPOL-Key(S, M, A, I, K, SM, KeyRSC, ANonce/SNonce, MIC, DataKDs)

where

S	means the initial key exchange is complete. This is the Secure bit of the Key Information field.
M	means the MIC is available in message. This should be set in all messages except message 1 of a 4-way handshake. This is the Key MIC bit of the Key Information field.
A	means a response is required to this message. This is used when the receiver should respond to this message. This is the Key Ack bit of the Key Information field.
I	is the Install bit: Install/Not install for the pairwise key. This is the Install bit of the Key Information field.
K	is the key type: P (Pairwise), G (Group/SMK). This is the Key Type bit of the Key Information field.
SM	is the SMK Message bit: indicates that this message is part of SMK handshake.
KeyRSC	is the key RSC. This is the Key RSC field.
ANonce/SNonce	is the Authenticator/Supplicant nonce. This is the Key Nonce field.
MIC	is the integrity check, which is generated using the KCK. This is the Key MIC field.
DataKDs	is a sequence of zero or more elements and KDEs, contained in the Key Data field, which may contain the following:
RSNE	is described in 9.4.2.25.
RSNE[KeyName]	is the RSNE, with the PMKID field set to KeyName.
GTK[N]	is the GTK, with the key identifier field set to N. The key identifier specifies which index is used for this GTK. Index 0 shall not be used for GTKs, except in mixed environments, as described in 12.7.1.
FTE	is the Fast BSS Transition element, described in 9.4.2.48
MDE	is the Mobility Domain element, described in 9.4.2.47
TIE[IntervalType]	is a Timeout Interval element of type IntervalType, as described in 9.4.2.49, containing e.g., for type KeyLifetime, the lifetime of the FT key hierarchy.
IGTK[M]	is the IGTK, with key identifier field set to M.
IPN	is the current IGTK replay counter value provided by the IGTK KDE
PMKID	is of type PMKID KDE and is the key identifier used during 4-way PTK handshake for PMK identification and during 4-way STK handshake for SMK identification.
Lifetime	is the key lifetime KDE used for sending the expiration timeout value for SMK used during PeerKey handshake for SMK identification.
Initiator MAC	is the Initiator MAC KDE used during PeerKey handshake
Peer MAC	is the Peer MAC KDE used during PeerKey handshake
Initiator Nonce	is the Initiator Nonce KDE used during PeerKey handshake. This is used when multiple nonces need to be sent.
Peer Nonce	is the Peer Nonce KDE used during PeerKey handshake. This is used when multiple nonces need to be sent.
SMK KDE	is the SMK during SMK handshake.
Error KDE	is an error KDE used when error bit E is equal to 1 during PeerKey handshake.

12.7.5 Nonce generation

The following is an informative description of Nonce generation.

All STAs contain a global key counter, which is 256 bits in size. It should be initialized at system boot-up time to a fresh cryptographic-quality random number. Refer to J.5 on random number generation. It is recommended that the counter value is initialized to the following:

PRF-256(Random number, "Init Counter", Local MAC Address || Time)

The local MAC address should be AA on the Authenticator and SPA on the Supplicant.

The random number is 256 bits in size. Time should be the current time from Network Time Protocol (NTP) or another time in NTP format whenever possible. This initialization causes different initial key counter values to occur across system restarts regardless of whether a real-time clock is available. The key counter can be used as additional input data for nonce generation. A STA derives a random nonce for each new use.

12.7.6 4-way handshake

12.7.6.1 General

RSNA defines a protocol using EAPOL-Key frames called the *4-way handshake*. The handshake completes the IEEE 802.1X authentication process. The information flow of the 4-way handshake is as follows:

Message 1: Authenticator → Supplicant: EAPOL-Key(0,0,1,0,P,0,0,ANonce,0,DataKD_M1)
where DataKD_M1 = 0 or PMKID for PTK generation, or PMKID KDE (for sending SMKID) for STK generation

Message 2: Supplicant → Authenticator: EAPOL-Key(0,1,0,0,P,0,0,SNonce,MIC,DataKD_M2)
where DataKD_M2 = RSNE for creating PTK generation or peer RSNE, Lifetime KDE, SMKID KDE (for sending SMKID) for STK generation

Message 3: Authenticator → Supplicant:
EAPOL-Key(1,1,1,1,P,0,KeyRSC,ANonce,MIC,DataKD_M3)
where DataKD_M3 = RSNE,GTK[N] for creating PTK generation or initiator RSNE, Lifetime KDE for STK generation

Message 4: Supplicant → Authenticator: EAPOL-Key(1,1,0,0,P,0,0,0,MIC,DataKD_M4)
where DataKD_M4 = 0.

The FT initial mobility domain association uses the FT 4-way handshake to establish an initial PTKSA, GTKSA and, if management frame protection is enabled, an IGTKSA, that is based on this protocol. The FT 4-way handshake protocol is described in 13.4.

Here, the following assumptions apply:

— EAPOL-Key(·) denotes an EAPOL-Key frame conveying the specified argument list, using the notation introduced in 12.7.4.
— ANonce is a nonce that the Authenticator contributes for PTK generation or that the initiator STA contributes for STK generation. ANonce has the same value in message 1 and message 3.
— SNonce is a nonce from the Supplicant for PTK generation or from the peer STA for STK generation.
— P means the pairwise bit is set.
— The MIC is computed over the body of the EAPOL-Key frame (with the Key MIC field first zeroed before the computation) using the KCK defined in 12.7.1.3 for PTK generation or SKCK defined in 12.7.1.6.
— RSNE represents the appropriate RSNEs.
— GTK[N] represents the GTK with its key identifier.
— SMKID represents the SMKID key identifier used during STK generation.
— Lifetime represents the expiration timeout used for exchanging SMK expiration value.

NOTE—While the MIC calculation is the same in each direction, the Key Ack bit is different in each direction. It is set in EAPOL-Key frames from the Authenticator and 0 in EAPOL-Key frames from the Supplicant. 4-way handshake requests from the Supplicant have the Request bit equal to 1. The Authenticator and Supplicant need to check these bits to stop reflection attacks. It is important that message 1 contents not be used to update state, in particular the keys in use, until the data are validated with message 3.

12.7.6.2 4-way handshake message 1

Message 1 uses the following values for each of the EAPOL-Key frame fields:

- Descriptor Type = N – see 12.7.2
- Key Information:
  - Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0
  - Key Type = 1 (Pairwise)
  - SMK Message = 0
  - Install = 0
  - Key Ack = 1
  - Key MIC = 0
  - Secure = 0
  - Error = 0
  - Request = 0
  - Encrypted Key Data = 0
  - Reserved = 0 – unused by this protocol version
- Key Length = Cipher-suite-specific; see Table 12-4
- Key Replay Counter = *n* – to allow Authenticator or initiator STA to match the right message 2 from Supplicant or peer STA
- Key Nonce = ANonce
- EAPOL-Key IV = 0
- Key RSC = 0
- Key MIC = 0
- Key Data Length = length of Key Data field in octets
- Key Data = PMKID for the PMK being used during PTK generation or SMKID for SMK being used during STK generation

Processing for PTK generation is as follows:

The Authenticator sends message 1 to the Supplicant at the end of a successful IEEE 802.1X authentication, after (re)association completes for a STA that has authenticated with SAE or PSK authentication is negotiated, when a cached PMKSA is used, or after a STA requests a new key. On reception of message 1, the Supplicant determines whether the Key Replay Counter field value has been used before with the current PMKSA. If the Key Replay Counter field value is less than or equal to the current local value, the Supplicant discards the message. Otherwise, the Supplicant:

a) Generates a new nonce SNonce.
b) Derives PTK.
c) Constructs message 2.

Processing for STK generation is as follows:

The initiator STA (STA_I) sends message 1 to the peer STA (STA_P) at the end of a successful SMK handshake, when SMKSA is created. On reception of message 1, the STA_P determines whether the Key Replay Counter field value has been used before with the current SMKSA. If the Key Replay Counter field value is less than or equal to the current local value, the STA_P discards the message. Otherwise, the STA_P

a) Generates a 256-bit random number following the recommendations of 12.7.5. That number is sent as a peer nonce as part of the Key Nonce field. This Nonce is different from the peer nonce generated as part of the SMK handshake message 3.
b) Derives STK.
c) Constructs message 2.

12.7.6.3 4-way handshake message 2

Message 2 uses the following values for each of the EAPOL-Key frame fields:

- Descriptor Type = N – see 12.7.2
- Key Information:
  - Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0 – same as message 1
  - Key Type = 1 (Pairwise) – same as message 1
  - SMK Message = 0 – same as message 1
  - Install = 0
  - Key Ack = 0
  - Key MIC = 1
  - Secure = 0 – same as message 1
  - Error = 0 – same as message 1
  - Request = 0 – same as message 1
  - Encrypted Key Data = 0
  - Reserved = 0 – unused by this protocol version
- Key Length = 0
- Key Replay Counter = *n* – to let the Authenticator or initiator STA know to which message 1 this corresponds
- Key Nonce = SNonce
- EAPOL-Key IV = 0
- Key RSC = 0
- Key MIC = MIC(KCK, EAPOL) – MIC computed over the body of this EAPOL-Key frame with the Key MIC field first initialized to 0
- Key Data Length = length of Key Data field in octets
- Key Data =
  - — included RSNE – the sending STA's RSNE for PTK generation or peer RSNE for the current operating band, and when this message 2 is part of a fast BSS transition initial mobility domain association or an association started through the FT protocol, the PMKR1Name calculated by the S1KH according to the procedures of 12.7.1.7.4 is included in the PMKID field of the RSNE and the FTE and MDE are also included, or;
  - — The sending STA's Multi-band element for PTK generation for a supported band other than the current operating band if dot11MultibandImplemented is true, or;
  - — The sending STA's RSNE and Multi-band element(s) for generating a single PTK for all involved bands, if dot11MultibandImplemented is true and both the Authenticator and the

Supplicant use the same MAC address in the current operating band and the other supported band(s); or;

— The sending STA's RSNE and Multi-band element(s) for generating a different PTK for each involved band, if dot11MultibandImplemented is true and the Joint Multi-band RSNA subfield of the RSN capabilities field is 1 for both the Authenticator and the Supplicant, and either the Authenticator or the Supplicant uses different MAC addresses for different bands.

— Lifetime of SMK and SMKID for STK generation.

Processing for PTK generation is as follows:

The Supplicant sends message 2 to the Authenticator.

On reception of message 2, the Authenticator checks that the key replay counter corresponds to the outstanding message 1. If not, it silently discards the message. Otherwise, the Authenticator:

a) Derives PTK.

b) Verifies the message 2 MIC.

1) If the calculated MIC does not match the MIC that the Supplicant included in the EAPOL-Key frame, the Authenticator silently discards message 2.

2) If the MIC is valid and this message 2 is part of a fast BSS transition initial mobility domain association or an association started through the FT protocol, the Authenticator checks that all fields of the RSNE other than the PMKID field bitwise matches the fields from the (Re)Association Request frame and that the FTE and MDE are the same as those provided in the AP's (Re)Association Response frame. If the MIC is valid and this message 2 is not part of a fast BSS transition initial mobility domain association and this message 2 is not part of an association started through the FT protocol, the Authenticator checks that the RSNE bitwise matches that from the (Re)Association Request frame.

i) If these are not exactly the same, the Authenticator uses MLME-DEAUTHENTICATE.request primitive to terminate the association.

ii) If they do match bitwise, the Authenticator constructs message 3.

c) If management frame protection is being negotiated, the AP initializes the SA Query Transaction Identifier to an implementation-specific non-negative integer value, valid for the current pairwise security association.

Processing for STK generation is as follows:

The STA_P sends message 2 to the STA_I. On reception of message 2, the STA_I checks that the key replay counter corresponds to message 1. If not, it silently discards the message. Otherwise, the STA_I

a) Derives the STK.

b) Verifies the message 2 MIC using the SKCK. If the calculated MIC does not match the MIC that the STA_P included in the EAPOL-Key frame, the STA_I silently discards message 2.

c) If the MIC is valid, the STA_I checks that the RSNE bitwise matches that from the SMK handshake message 5. If these are not exactly the same, STA_I silently discards the message and restarts the 4-way handshake after deleting the existing 4-way handshake states.

d) If they do match bitwise, the STA_I checks SMKID with the value of SMKID in the SMKSA. If these are not exactly the same, STA_I silently discards the message and restarts the 4-way handshake after deleting the existing 4-way handshake states.

e) If they do match, the STA_I constructs message 3. It also compares the Key Lifetime value from the KDE with value in its SMKSA. If value in its SMKSA is less, it discards the value received in message 2. Otherwise, it updates the value in the SMKSA with value in message 2.

12.7.6.4 4-way handshake message 3

Message 3 uses the following values for each of the EAPOL-Key frame fields:

- Descriptor Type = N – see 12.7.2
- Key Information:
  - Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0 – same as message 1
  - Key Type = 1 (Pairwise) – same as message 1
  - SMK Message = 0 - same as message 1
  - Install = 0/1 – For PTK generation, 0 only if the AP does not support key mapping keys, or if the STA has the No Pairwise bit (in the RSN Capabilities field) equal to 1and only the group key is used. For STK generation, this bit is set to 1.
  - Key Ack = 1
  - Key MIC = 1
  - Secure = 1 (keys installed)
  - Error = 0 – same as message 1
  - Request = 0 – same as message 1
  - Encrypted Key Data = 1
  - Reserved = 0 – unused by this protocol version
- Key Length = Cipher-suite-specific; see Table 12-4
- Key Replay Counter = *n*+*1*
- Key Nonce = ANonce – same as message 1
- EAPOL-Key IV = 0 (Version 2) or random (Version 1)
- Key RSC = For PTK generation, starting TSC or PN that the Authenticator's STA uses in MPDUs protected by GTK. For STK generation, this is set to 0.
- Key MIC = MIC(KCK, EAPOL) or MIC(SKCK, EAPOL) – MIC computed over the body of this EAPOL-Key frame with the Key MIC field first initialized to 0
- Key Data Length = length of Key Data field in octets of included RSNEs and GTK
- Key Data =
  - — For PTK generation for the current operating band, the AP's Beacon/Probe Response frame's RSNE for the current operating band, and, optionally, a second RSNE that is the Authenticator's pairwise cipher suite assignment for the current operating band, and, if a group cipher has been negotiated, the GTK and the GTK's key identifier (see 12.7.2) for the current operating band, and if management frame protection is negotiated, the IGTK KDE, and when this message 3 is part of a fast BSS transition initial mobility domain association or an association started through the FT protocol, the PMKR1Name calculated according to the procedures of 12.7.1.7.4 in the PMKID field of the RSNE and the FTE with the same contents as in the (Re)Association Response frame, the MDE with the same contents as in the (Re)Association Response frame, the reassociation deadline timeout set to the minimum of dot11FTReassociationDeadline and the key lifetime in the TIE[ReassociationDeadline], and the PTK lifetime in the TIE[KeyLifetime]; or
  - — For PTK generation for a supported band other than the current operating band, the Authenticator's Beacon/DMG Beacon/Announce/Probe Response/Information Response frame's Multi-band element associated with the supported band, and optionally a second Multi-band element that indicates the Authenticator's pairwise cipher suite assignment for the supported band, and, if group cipher for the supported band is negotiated, the Multi-band GTK KDE for the supported band if dot11MultibandImplemented is true, or;

— For generating a single PTK for all involved bands, the Authenticator's Beacon/DMG Beacon/Announce/Probe Response/Information Response frame's RSNE and Multi-band element(s), and optionally, additional RSNE and Multi-band element(s) that indicate the Authenticator's assignment of one pairwise cipher suite for all involved bands; if a group cipher for all involved bands is negotiated, the GTK and the GTK's key identifier for all involved bands, if dot11MultibandImplemented is true and both the Authenticator and the Supplicant use the same MAC address in the current operating band and the other supported band(s), or;

— For generating different PTKs for the current operating band and other supported band(s), the Authenticator's Beacon/DMG Beacon/Announce/Probe Response/Information Response frame's RSNE and Multi-band element(s), and optionally, additional RSNE and Multi-band elements that are the Authenticator's pairwise cipher suite assignments for one or more involved bands; if group ciphers for the involved bands are negotiated, the Multi-band GTK KDEs for the involved bands, if dot11MultibandImplemented is true and the Joint Multi-band RSNA subfield is 1 for both the Authenticator and Supplicant, and either the Authenticator or the Supplicant uses different MAC addresses for different bands.

— For STK generation Initiator RSNE, Lifetime of SMK is used.

— If the Extended Key ID for Individually Addressed Frames subfield of the RSN Capabilities field is 1 for both the Authenticator/STA_I and Supplicant/STA_P, then the Authenticator/STA_I includes the Key ID KDE with the assigned key identifier for the current operating band; or the Authenticator includes the Multi-band Key ID KDE(s) with the assigned key identifier(s) for one or more supported bands if dot11MultibandImplemented is true.

Processing for PTK generation is as follows:

If the Extended Key ID for Individually Addressed Frames subfield of the RSN Capabilities field is 1 for both the Authenticator and the Supplicant, then the Authenticator assigns a new Key ID for the PTKSA in the range 0 to 1 that is different from the Key ID assigned in the previous handshake and uses the MLME-SETKEYS.request primitive to install the new key to receive individually addressed MPDUs protected by the PTK with the assigned Key ID. Otherwise Key ID 0 is used and installation of the key is deferred until after message 4 has been received. The Authenticator sends message 3 to the Supplicant.

NOTE—If an existing PTK is still in effect, the Authenticator IEEE 802.11 MAC continues to transmit protected, individually addressed MPDUs (if any) using the existing key. With the installation of the new key for receive, the Authenticator is able to receive protected, individually addressed MPDUs using either the old key (if present) or the new key.

On reception of message 3, the Supplicant silently discards the message if the Key Replay Counter field value has already been used or if the ANonce value in message 3 differs from the ANonce value in message 1. The Supplicant also:

a) Verifies the RSNE. If this message 3 is part of a fast BSS transition initial mobility domain association or an association started through the FT protocol, the Supplicant verifies that the PMKR1Name in the PMKID field of the RSNE is identical to the value it sent in message 2 and verifies that all other fields of the RSNE are identical to the fields in the RSNE present in the Beacon or Probe Response frames and verifies that the FTE and MDE are the same as in the (Re)Association Response frame. Otherwise, the Supplicant verifies that the RSNE is identical to that the STA received in the Beacon or Probe Response frame. If any of these verification steps indicates a mismatch, the STA shall disassociate or deauthenticate. If a second RSNE is provided in the message, the Supplicant uses the pairwise cipher suite specified in the second RSNE or deauthenticates.

b) Verifies the message 3 MIC. If the calculated MIC does not match the MIC that the Authenticator included in the EAPOL-Key frame, the Supplicant silently discards message 3.

c) Updates the last-seen value of the Key Replay Counter field.
d) If the Extended Key ID for Individually Addressed Frames subfield of the RSN Capabilities field is 1 for both the Authenticator and Supplicant: Uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to receive individually addressed MPDUs protected by the PTK with the assigned Key ID.
e) Constructs message 4.
f) Sends message 4 to the Authenticator.
g) Uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to send and, if the receive key has not yet been installed, to receive individually addressed MPDUs protected by the PTK. The GTK is also configured by MLME-SETKEYS primitive.

Processing for STK generation is as follows:

If the Extended Key ID for Individually Addressed Frames subfield of the RSN Capabilities field is set to 1 for both the STA_I and the STA_P, then the Authenticator assigns a new Key ID for the STKSA in the range 0 to 1 that is different from the Key ID assigned in the previous handshake and uses the MLME-SETKEYS.request primitive to install the new key to receive individually addressed MPDUs protected by the STK with the assigned Key ID. Otherwise Key ID 0 is used and installation of the key is deferred until after message 4 has been received. The STA_I sends message 3 to the STA_P.

NOTE—If an existing STK is still in effect, the STA_I IEEE 802.11 MAC continues to transmit protected, individually addressed MPDUs (if any) using the existing key. With the installation of the new key for receive, the STA_I is able to receive protected, individually addressed MPDUs using both the old key (if present) or the new key.

On reception of message 3, the STA_P silently discards the message if the Key Replay Counter field value has already been used or if the INonce value in message 3 differs form the INonce value in message 1. Otherwise,

a) The STA_P verifies the message 3 MIC using the SKCK in the SMKSA. If the calculated MIC does not match the MIC that the STA_P included in the EAPOL-Key frame, the STA_I silently discards message 3.
b) If the MIC is valid, the STA_P checks that the RSNE bitwise matches that from the 4-way handshake message 2. If these are not exactly the same, STA_P silently discards the message and deletes existing 4-way handshake states.
c) If they do match, the STA_P constructs message 4. It also compares the Key Lifetime value from KDE with value in its SMKSA. If value in the SMKSA is less, it discards the value received in message 3. Otherwise, it updates the value in the SMKSA with value in message 3.
d) If the Extended Key ID for Individually Addressed Frames subfield of the RSN Capabilities field is 1 for both the Authenticator and Supplicant then prior to sending message 4, STA_P uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to receive individually addressed MPDUs protected by the STK with the assigned Key ID.
e) After sending message 4, STA_P uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to send and, if the receive key has not yet been installed, to receive individually addressed MPDUs protected by the STK with the assigned Key ID.

12.7.6.5 4-way handshake message 4

Message 4 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0 – same as message 1

Key Type = 1 (Pairwise) – same as message 1
SMK Message = 0 - same as message 1
Install = 0
Key Ack = 0 – this is the last message
Key MIC = 1
Secure = 1
Error = 0
Request = 0
Encrypted Key Data = 0
Reserved = 0 – unused by this protocol version
Key Length = 0
Key Replay Counter = $n+1$
Key Nonce = 0
EAPOL-Key IV = 0
Key RSC = 0
Key MIC = MIC(KCK, EAPOL) or MIC(SKCK, EAPOL) – MIC computed over the body of this EAPOL-Key frame with the Key MIC field first initialized to 0
Key Data Length = length of Key Data field in octets
Key Data = none required

Processing for PTK generation is as follows:

The Supplicant sends message 4 to the Authenticator. Note that when the 4-way handshake is first used, message 4 is sent in the clear.

On reception of message 4, the Authenticator verifies that the Key Replay Counter field value is one that it used on this 4-way handshake; if it is not, it silently discards the message. Otherwise:

a) The Authenticator checks the MIC. If the calculated MIC does not match the MIC that the Supplicant included in the EAPOL-Key frame, the Authenticator silently discards message 4.
b) If the MIC is valid, the Authenticator uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to send and, if the receive key has not yet been installed, to receive protected, individually addressed MPDUs using for the new PTK.
c) The Authenticator updates the Key Replay Counter field so that it uses a fresh value if a rekey becomes necessary.

Processing for STK generation is as follows:

The STA_P sends message 4 to the STA_I. On reception of message 4, the STA_I verifies that the Key Replay Counter field value is one that it used on this 4-way handshake; if it is not, it silently discards the message. Otherwise,

a) The STA_I checks the MIC. If the calculated MIC does not match the MIC that the STA_P included in the EAPOL-Key frame, the STA_I silently discards message 4.
b) If the MIC is valid, the STA_I uses the MLME-SETKEYS.request primitive to configure the IEEE 802.11 MAC to send and, if the receive key has not yet been installed, to receive protected, individually addressed MPDUs using for the new STK.
c) The STA_I updates the Key Replay Counter field so that it uses a fresh value if a rekey becomes necessary.

12.7.6.6 4-way handshake implementation considerations

When the 4-way handshake is used as part of the STK handshake, the initiator STA acts as Authenticator and peer STA acts as Supplicant.

If the Authenticator does not receive a reply to its messages, it shall attempt dot11RSNAConfigPairwiseUpdateCount transmits of the message, plus a final timeout. The retransmit timeout value shall be 100 ms for the first timeout, half the listen interval for the second timeout, and the listen interval for subsequent timeouts. If there is no listen interval or the listen interval is zero, then 100 ms shall be used for all timeout values. If it still has not received a response after these retries, then for PTK generation the Authenticator should deauthenticate the STA, and for STK generation the STAs should delete the SMKSA and initiate an STSL application teardown procedure.

For PTK generation, if the STA does not receive message 1 within the expected time interval (prior to IEEE 802.1X timeout), it should disassociate, deauthenticate, and try another AP/STA. For STK generation, if the peer STA does not receive message 1 or message 3 within the expected time interval (prior to dot11RSNAConfigSATimeout as specified in 12.7.8), it deletes the SMKSA and invokes an STSL application teardown procedure.

The Authenticator should ignore EAPOL-Key frames it is not expecting in reply to messages it has sent or EAPOL-Key frames in which the Ack bit is 1. This stops an attacker from sending the first message to the Supplicant who responds to the Authenticator.

An implementation should save the KCK and KEK beyond the 4-way handshake, as they are needed for group key handshakes, STK Rekeying, and recovery from TKIP MIC failures.

The Supplicant uses the MLME-SETKEYS.request primitive to configure the temporal key from 12.7.1 into its STA after sending message 4 to the Authenticator.

If the RSNE check for message 2 or message 3 fails, the SME should log an error and deauthenticate the peer.

12.7.6.7 Sample 4-way handshake

The following is an informative sample of a 4-way handshake for illustration.

After IEEE 802.1X authentication completes by the AP sending an EAP-Success, the AP initiates the 4-way handshake. See Figure 12-46.

The 4-way handshake consists of the following steps:

a) The Authenticator sends an EAPOL-Key frame containing an ANonce.
b) The Supplicant derives a PTK from ANonce and SNonce.
c) The Supplicant sends an EAPOL-Key frame containing SNonce, the RSNE from the (Re)Association Request frame, and a MIC.
d) The Authenticator derives PTK from ANonce and SNonce and validates the MIC in the EAPOL-Key frame.
e) The Authenticator sends an EAPOL-Key frame containing ANonce, the RSNE from its Beacon or Probe Response frames, the MIC, the GTK, an indication of whether to install the temporal keys, and if management frame protection is negotiated, the IGTK.
f) The Supplicant sends an EAPOL-Key frame to confirm whether the temporal keys were installed.

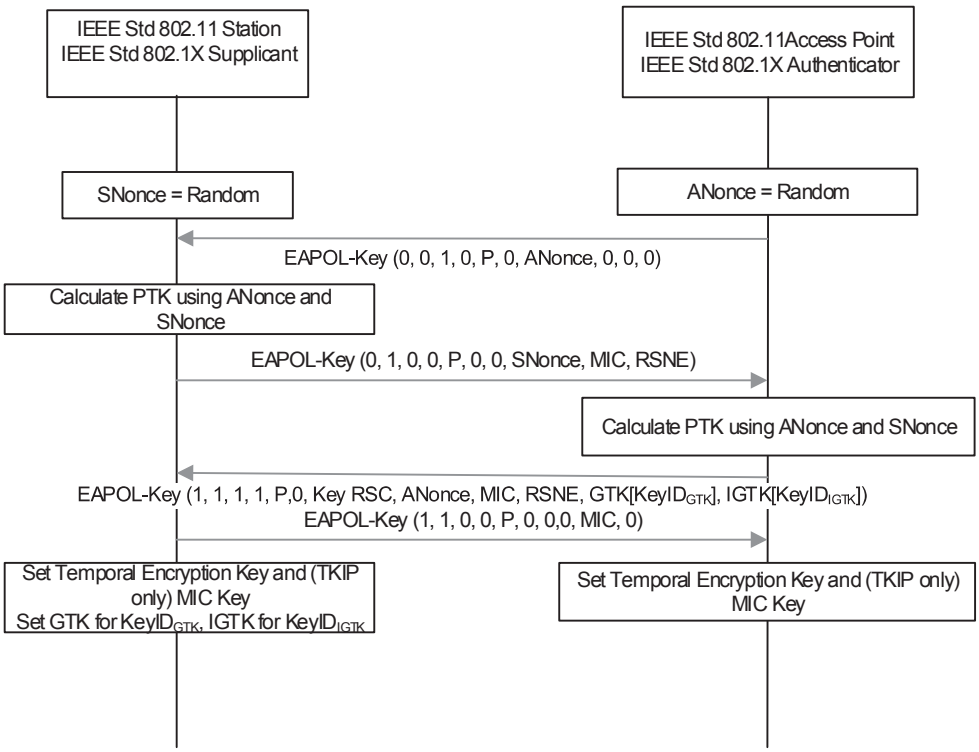

Figure 12-46—Sample 4-way handshake

12.7.6.8 4-way handshake analysis

The following is an informative analysis of the 4-way handshake.

This subclause makes the trust assumptions used in this protocol explicit. The protocol assumes the following:

— The PMK is known only by the Supplicant's STA and the Authenticator's STA.
— The Supplicant's STA uses IEEE 802 address SPA.
— The Authenticator's STA uses IEEE 802 address AA.

In many instantiations the RSNA architecture immediately breaks the first assumption because the IEEE 802.1X AS also knows the PMK. Therefore, additional assumptions are required:

— The AS does not expose the PMK to other parties.
— The AS does not masquerade as the Supplicant to the Authenticator.
— The AS does not masquerade as the Authenticator to the Supplicant.
— The AS does not masquerade as the Supplicant's STA.
— The AS does not masquerade as the Authenticator's STA.

The protocol also assumes this particular Supplicant-Authenticator pair is authorized to know this PMK and to use it in the 4-way handshake. If any of these assumptions are broken, then the protocol fails to provide any security guarantees.

The protocol also assumes that the AS delivers the correct PMK to the AP with IEEE 802 address AA and that the STA with IEEE 802 address SPA hosts the Supplicant that negotiated the PMK with the AS. None

of the protocols defined by this standard and IEEE Std 802.1X-2010 permit the AS, the Authenticator, the Supplicant, or either STA to verify these assumptions.

The PTK derivation step

PTK =PRF-Length(PMK, "Pairwise key expansion", Min(AA,SPA) || Max(AA,SPA) ||
Min(ANonce,SNonce) || Max(ANonce,SNonce))

performs a number of functions:

- Including the AA and SPA in the computation
  - Binds the PTK to the communicating STAs and
  - Prevents undetected man-in-the-middle attacks against 4-way handshake messages between the STAs with these two IEEE 802 addresses.
- If ANonce is randomly selected, including ANonce
  - Guarantees the STA at IEEE 802 address AA that PTK is fresh,
  - Guarantees that message 2 and message 4 are live, and
  - Uniquely identifies PTK as <AA, ANonce>.
- If SNonce is randomly selected, including SNonce
  - Guarantees the STA at IEEE 802 address SPA that PTK is fresh,
  - Guarantees that message 3 is live, and
  - Uniquely identifies PTK as <SPA, SNonce>.

Choosing the nonces randomly helps prevent precomputation attacks. With unpredictable nonces, a man-in-the-middle attack that uses the Supplicant to precompute messages to attack the Authenticator cannot progress beyond message 2, and a similar attack against the Supplicant cannot progress beyond message 3. The protocol might execute further before an error if predictable nonces are used.

Message 1 delivers ANonce to the Supplicant and initiates negotiation for a new PTK. It identifies AA as the peer STA to the Supplicant's STA. If an adversary modifies either of the addresses or ANonce, the Authenticator detects the result when validating the MIC in message 2. Message 1 does not carry a MIC, as it is impossible for the Supplicant to distinguish this message from a replay without maintaining state of all security associations through all time (PMK might be a static key).

Message 2 delivers SNonce to the Authenticator so it can derive the PTK. If the Authenticator selected ANonce randomly, message 2 also demonstrates to the Authenticator that the Supplicant is live, that the PTK is fresh, and that there is no man-in-the-middle attack, as the PTK includes the IEEE 802 MAC addresses of both. Inclusion of ANonce in the PTK derivation also protects against replay. The MIC prevents undetected modification of message 2 contents.

Message 3 confirms to the Supplicant that there is no man-in-the-middle attack. If the Supplicant selected SNonce randomly, it also demonstrates that the PTK is fresh and that the Authenticator is live. The MIC again prevents undetected modification of message 3.

While message 4 serves no cryptographic purpose, it serves as an acknowledgment to message 3. It is required to inform the Authenticator that the Supplicant has installed the PTK and GTK and hence can receive encrypted frames.

The PTK and GTK are installed by using MLME-SETKEYS.request primitive after message 4 is sent. The PTK is installed before the GTK.

Then the 4-way handshake uses a correct, but unusual, mechanism to guard against replay. As noted earlier in this subclause, ANonce provides replay protection to the Authenticator, and SNonce to the Supplicant. In most session initiation protocols, replay protection is accomplished explicitly by selecting a nonce randomly and requiring the peer to reflect the received nonce in a response message. The 4-way handshake instead mixes ANonce and SNonce into the PTK, and replays are detected implicitly by MIC failures. In particular, the Key Replay Counter field serves no cryptographic purpose in the 4-way handshake. Its presence is not detrimental, however, and it plays a useful role as a minor performance optimization for processing stale instances of message 2. This replay mechanism is correct, but its implicit nature makes the protocol harder to understand than an explicit approach.

It is critical to the correctness of the 4-way handshake that at least one bit differs in each message. Within the 4-way handshake, message 1 can be recognized as the only one in which the Key MIC bit is 0, meaning message 1 does not include the MIC, while message 2 to message 4 do. Message 3 differs from message 2 by not asserting the Ack bit and from message 4 by asserting the Ack Bit. Message 2 differs from message 4 by including the RSNE.

Request messages are distinct from 4-way handshake messages because the former asserts the Request bit and 4-way handshake messages do not. Group key handshake messages are distinct from 4-way handshake messages because they assert a different key type.

12.7.7 Group key handshake

12.7.7.1 General

The Authenticator uses the Group key handshake to send a new GTK and, if management frame protection is negotiated, a new IGTK to the Supplicant.

The Authenticator may initiate the exchange when a Supplicant is disassociated or deauthenticated.

Message 1: Authenticator → Supplicant:
EAPOL-Key(1,1,1,0,G,0,Key RSC,0, MIC,GTK[N],IGTK[M])

Message 2: Supplicant → Authenticator: EAPOL-Key(1,1,0,0,G,0,0,0,MIC,0)

Here, the following assumptions apply:

— Key RSC denotes the last TSC or PN sent using the GTK.
— GTK[N] denotes the GTK with its key identifier as defined in 12.7.2 using the KEK defined in 12.7.1.3 and associated IV.
— IGTK[M], when present, denotes the IGTK with its key identifier as defined in 12.7.2 using the KEK defined in 12.7.1.3 and associated IV.
— The MIC is computed over the body of the EAPOL-Key frame (with the MIC field zeroed for the computation) using the KCK defined in 12.7.1.3.

The Supplicant may trigger a group key handshake by sending an EAPOL-Key frame with the Request bit set to 1 and the type of the Group Key bit.

An Authenticator shall do a 4-way handshake before a group key handshake if both are required to be done.

NOTE—The Authenticator cannot initiate the group key handshake until the 4-way handshake completes successfully.

12.7.7.2 Group key handshake message 1

Message 1 uses the following values for each of the EAPOL-Key frame fields:

- Descriptor Type = N – see 12.7.2
- Key Information:
  - Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0
  - Key Type = 0 (Group/SMK)
  - SMK Message = 0
  - Install = 0
  - Key Ack = 1
  - Key MIC = 1
  - Secure = 1
  - Error = 0
  - Request = 0
  - Encrypted Key Data = 1
  - Reserved = 0
- Key Length = 0
- Key Replay Counter = *n*+2
- Key Nonce = 0
- EAPOL-Key IV = 0 (Version 2) or random (Version 1)
- Key RSC = last TSC or PN for the GTK
- Key MIC = MIC(KCK, EAPOL)
- Key Data Length = Cipher-suite-specific; see Table 12-4
- Key Data = encrypted, encapsulated
  - — GTK and the GTK's key identifier (see 12.7.2)
  - — When present, IGTK, IGTK's key identifier, and IPN (see 12.7.2)

The Authenticator sends message 1 to the Supplicant.

On reception of message 1, the Supplicant:

a) Verifies that the Key Replay Counter field value has not yet been seen before, i.e., its value is strictly larger than that in any other EAPOL-Key frame received thus far during this session.

b) Verifies that the MIC is valid, i.e., it uses the KCK that is part of the PTK to verify that there is no data integrity error.

c) Uses the MLME-SETKEYS.request primitive to configure the temporal GTK and, when present, IGTK into its IEEE 802.11 MAC.

d) Responds by creating and sending message 2 of the group key handshake to the Authenticator and incrementing the replay counter.

NOTE—The Authenticator increments and uses a new Key Replay Counter field value on every message 1 instance, even retries, because the message 2 responding to an earlier message 1 might have been lost. If the Authenticator did not increment the replay counter, the Supplicant discards the retry, and no responding message 2 ever arrives.

12.7.7.3 Group key handshake message 2

Message 2 uses the following values for each of the EAPOL-Key frame fields:

- Descriptor Type = N – see 12.7.2
- Key Information:
  - Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0 – same as message 1
  - Key Type = 0 (Group/SMK) – same as message 1
  - Install = 0
  - Key Ack = 0
  - Key MIC = 1
  - Secure = 1
  - Error = 0
  - Request = 0
  - Encrypted Key Data = 0
  - Reserved = 0
- Key Length = 0
- Key Replay Counter = *n*+2 – same as message 1
- Key Nonce = 0
- EAPOL-Key IV = 0
- Key RSC = 0
- Key MIC = MIC(KCK, EAPOL)
- Key Data Length = 0
- Key Data = none required

On reception of message 2, the Authenticator:

a) Verifies that the Key Replay Counter field value matches one it has used in the group key handshake.

b) Verifies that the MIC is valid, i.e., it uses the KCK that is part of the PTK to verify that there is no data integrity error.

12.7.7.4 Group key handshake implementation considerations

If the Authenticator does not receive a reply to its messages, its shall attempt dot11RSNAConfigGroupUpdateCount transmits of the message, plus a final timeout. The retransmit timeout value shall be 100 ms for the first timeout, half the listen interval for the second timeout, and the listen interval for subsequent timeouts. If there is no listen interval or the listen interval is zero, then 100 ms shall be used for all timeout values. If it still has not received a response after this, then the Authenticator's STA should use the MLME-DEAUTHENTICATE.request primitive to deauthenticate the STA.

12.7.7.5 Sample group key handshake

The following is an informative sample of a group key handshake for illustration.

The state machines in 12.7.10 and 12.7.11 change the GTK and, when present, IGTK in use by the network. See Figure 12-47.

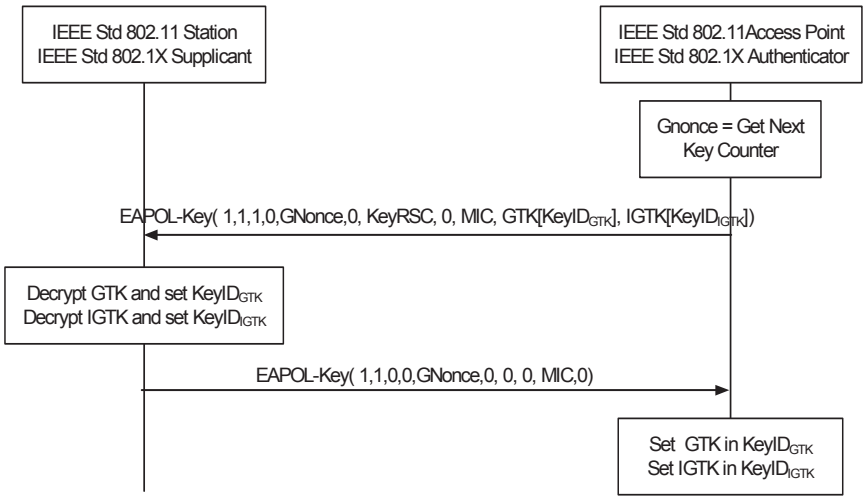

Figure 12-47—Sample group key handshake

The following steps occur:

a) The Authenticator generates a new GTK and when management frame protection has been negotiated, a new IGTK. It encapsulates the GTK and, as necessary, the IGTK, and sends an EAPOL-Key frame containing the GTK and IGTK (message 1), along with the last TSC or PN used with the GTK (RSC) and the last IPN used with the IGTK.

b) On receiving the EAPOL-Key frame, the Supplicant validates the MIC, decapsulates the GTK and, when present, the IGTK, and uses the MLME-SETKEYS.request primitive to configure the GTK, PN, IGTK, RSC, and IPN in its STA.

c) The Supplicant then constructs and sends an EAPOL-Key frame in acknowledgment to the Authenticator.

d) On receiving the EAPOL-Key frame, the Authenticator validates the MIC. If the GTK and, if present, the IGTK are is not already configured into IEEE 802.11 MAC, after the Authenticator has delivered the GTK and IGTK to all associated STAs, it uses the MLME-SETKEYS.request primitive to configure the GTK and IGTK.

12.7.8 PeerKey handshake

12.7.8.1 General

The PeerKey handshake occurs after any other STSL setup procedures and is used to create an STKSA providing data confidentiality between the two STAs. The AP establishes an RSNA with each STA prior to PeerKey setup. The initiator STA starts the PeerKey handshake, at the conclusion of which a key is established to secure the connection.

STSL security PeerKey handshake is used to establish security for Data frames passed directly between two STAs associated with the same AP. The AP establishes an RSNA with each STA prior to the PeerKey handshake. After the STAs establish the STSL, the initiator STA starts the PeerKey handshake, at the conclusion of which a key is established to secure the connection. The PeerKey handshake is used to create an STKSA between the two STAs.

The PeerKey EAPOL-Key exchange provides a mechanism for obtaining the keys to be used for direct station-to-station communication. The initiator STA shall start a timer when it sends the first EAPOL-Key

frame, and the peer STA shall do the same on receipt of the first EAPOL-Key frame. On expiration of this timer, the STA shall transition to the STKINIT state.

A STA should use the PeerKey handshake prior to transferring any direct station-to-station Data frames. The STKSA should be deleted when the station-to-station connection is terminated.

Here, the following assumptions apply:

- EAPOL-Key() denotes an EAPOL-Key frame conveying the specified argument list, using the notation introduced in 12.7.4.
- STA_I is the initiator STA.
- STA_P is the peer STA.
- AP is the access point with which both the STA_I and the STA_P are associated.
- MAC_I is the MAC address of the STA_I.
- MAC_P is the MAC address of the STA_I.
- INonce is the nonce generated by the STA_I.
- PNonce is the nonce generated by the STA_P.

The PeerKey handshake has two components:

a) **SMK handshake:** This handshake is initiated by the initiator STA, and as a result of this handshake, the SMKSA is installed in both the STAs. This message exchange goes through the AP and is protected using the PTK.

b) **4-way STK handshake:** Using the installed SMKSA, the initiator STA initiates the 4-way handshake (per 12.7.6), and as a result of this, the STKSA gets installed in both the STAs. The STKSA is used for securing data exchange between the initiator STA and the peer STA. The 4-way handshake analysis described in 12.7.6.8 applies to the 4-way STK handshake.

12.7.8.2 SMK handshake

12.7.8.2.1 General

The initiator STA initiates the SMK handshake by sending the first message to the AP to establish an SMKSA between itself and another STA associated with the same AP. Unlike the 4-way handshake and the group key handshake, the SMK handshake is initiated by the initiator STA.

Message 1: Initiator STA → AP: EAPOL-Key(1,1,0,0,0,1,0, INonce, MIC, RSNE_I, MAC_P KDE)

Message 2: AP → Peer STA: EAPOL-Key(1,1,1,0,0,1,0, INonce, MIC, RSNE_I, MAC_I KDE)

Message 3: Peer STA → AP: EAPOL-Key(1,1,0,0,0,1,0, PNonce, MIC, RSNE_P, MAC_I KDE, Initiator Nonce KDE)

Message 4: AP → Peer STA: EAPOL-Key(1,1,0,1,0,1,0, PNonce, MIC, MAC_I KDE, Initiator Nonce KDE, SMK KDE, Lifetime KDE)

Message 5: AP → Initiator STA: EAPOL-Key(1,1,0,0,0,1,0, INonce, MIC, RSNE_P, MAC_P KDE, Peer Nonce KDE, SMK KDE, Lifetime KDE)

12.7.8.2.2 SMK handshake message 1

The initiator STA creates the RSNE (see 9.4.2.25) by including the Element ID, length, version, and pairwise cipher suite list fields. Since the group cipher suit field is before the pairwise cipher suite list field (so the STA needs to include it), the STA includes any value in this field, and the receiving STA ignores it. The initiator STA also generates a 256-bit random number following the recommendations of 12.7.5. That number is sent in the Key Nonce field.

Message 1 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0

Key Type = 0 (Group/SMK)

SMK Message = 1 (SMK)

Install = 0

Key Ack = 0

Key MIC = 1

Secure = 1

Error = 0

Request = 1

Encrypted Key Data = 0

Reserved = 0

Key Length = 0

Key Replay Counter = request EAPOL replay counter of initiating STA

Key Nonce = INonce

EAPOL-Key IV = 0

Key RSC = 0

Key MIC = MIC (initiating STA's KCK, EAPOL)

Key Data Length = Length of Key Data field in octets

Key Data = Initiator RSNE, peer MAC address KDE

The STA_I sends message 1 to the AP.

On receipt of message 1, the AP checks that the key replay counter corresponds to message 1. If not, it silently discards the message. Otherwise:

a) The AP verifies the message 1 MIC using the STA_I PTKSA. If the calculated MIC does not match the MIC that the STA_I included in the EAPOL-Key frame, the AP silently discards message 1.

b) If the MIC is correct, the AP checks if the STA_P is reachable. If it is not reachable, the AP shall send an error EAPOL-Key frame to STA_I per 12.7.8.5.2. After sending the message, AP silently discards message 1.

c) The AP checks if the AP has a secure connection with STA_P. If not, the AP shall send an error EAPOL-Key frame to STA_I per 12.7.8.5.3. After sending the message, AP silently discards message 1.

d) If all checks succeed, the AP creates message 2 using the STA_P PTKSA. The AP copies the contents of message 1 to create message 2.

12.7.8.2.3 SMK handshake message 2

Message 2 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0

Key Type = 0 (Group/SMK)

SMK Message = 1 (SMK)

Install = 0

Key Ack = 1

Key MIC = 1

Secure = 1

Error = 0

Request = 0

Encrypted Key Data = 0

Reserved = 0

Key Length = 0

Key Replay Counter = request EAPOL replay counter of AP and STA_P

Key Nonce = INonce

EAPOL-Key IV = 0

Key RSC = 0

Key MIC = MIC (KCK of the STA_P, EAPOL)

Key Data Length = Length of Key Data field in octets

Key Data = Initiator RSNE, initiator MAC address KDE

The AP sends message 2 to the STA_P. On receipt of message 2, the STA_P checks that the key replay counter corresponds to message 2. If not, it silently discards the message. Otherwise,

a) The STA_P verifies the message 2 MIC using the STA_P PTKSA. If the calculated MIC does not match the MIC that the AP included in the EAPOL-Key frame, the STA_P silently discards message 2.
b) If the MIC is correct, the STA_P checks if it supports at least one cipher suites proposed by the STA_I. If it does not, the STA_P shall send an error EAPOL-Key frame to STA_I through the AP per 12.7.8.5.4. After sending the error message, the STA_P silently discards message 2.
c) STA_O checks if it has already created an STSL with STA_I. If it has not, STA_P shall send an error EAPOL-Key frame to STA_I through the AP per 12.7.8.5.5. After sending the error message, the STA_P silently discards message 2.
d) If all checks succeed, the STA_P creates the state of PeerKey handshake and stores the INonce and the RSNE received in message 2.
e) STA_P selects a support cipher suite from the cipher suite list proposed by the STA_I and creates the peer RSNE, which is sent with message 3.
f) STA_P generates a 256-bit random number following the recommendations of 12.7.5. That number is sent as the peer Nonce KDE with message 3.
g) Using all the information, STA_P creates message 3.

12.7.8.2.4 SMK handshake message 3

Message 3 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0

Key Type = 0 (Group/SMK)

SMK Message = 1 (SMK)

Install = 0

Key Ack = 0

Key MIC = 1

Secure = 1

Error = 0

Request = 0

Encrypted Key Data = 0

Reserved = 0

Key Length = 0

Key Replay Counter = request EAPOL replay counter of peer STA

Key Nonce = PNonce

EAPOL-Key IV = 0

Key RSC = 0

Key MIC = MIC (KCK of STA_I, EAPOL)

Key Data Length = Length of Key Data field in octets

Key Data = Peer RSNE, initiator MAC address KDE, initiator Nonce KDE

The STA_P sends message 3 to the AP. On receipt of message 3, the AP checks that the key replay counter corresponds to message 3. If not, it silently discards the message. Otherwise,

a) The AP verifies the message 1 MIC using the STA_I PTKSA. If the calculated MIC does not match the MIC that the STA_P included in the EAPOL-Key frame, the AP silently discards message 1.
b) If MIC is correct, the AP checks if the STA_I is reachable. If it is not reachable, the AP shall send an error EAPOL-Key frame to the STA_P per 12.7.8.5.2. After sending the message, the AP silently discards message 3.
c) The AP checks if the AP has secure connection with STA_I. If it does not, the AP shall send an error EAPOL-Key frame to STA_P per 12.7.8.5.3. After sending the message, the AP silently discards message 3.
d) If all checks succeed, the AP generates a 256-bit random number drawn from a uniform distribution that is used as the SMK, which is sent with message 4 and message 5 as SMK KDEs.
e) Depending on the strength of random number generator, the AP sets the lifetime of the SMK, which is sent with message 4 and message 5 as Lifetime KDEs.
f) Using all the information, the AP creates message 4 and message 5.

12.7.8.2.5 SMK handshake message 4

Message 4 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0

Key Type = 0 (Group/SMK)

SMK Message = 1 (SMK)

Install = 1

Key Ack = 0

Key MIC = 1

Secure = 1

Error = 0

Request = 0

Encrypted Key Data = 1

Reserved = 0

Key Length = Cipher-suite-specific; see Table 12-4

Key Replay Counter = request EAPOL replay counter of AP

Key Nonce = PNonce

EAPOL-Key IV = 0

Key RSC = 0

Key MIC = MIC (KCK of the STA_I, EAPOL)

Key Data Length = Length of Key Data field in octets

Key Data = Encrypted initiator MAC address KDE, Initiator Nonce KDE, SMK KDE (contains SMK and PNonce), Lifetime KDE

The AP sends message 4 to the STA_P. On receipt of message 4, the STA_P checks that the key replay counter corresponds to message 4. If it does not, STA_P silently discards the message. Otherwise,

a) The STA_P verifies the message 4 MIC using STA_P PTKSA. If the calculated MIC does not match the MIC that the AP included in the EAPOL-Key frame, the STA_P silently discards message 4.

b) If the MIC is correct, STA_P identifies the PeerKey session using the PNonce sent as part of the Key Nonce field of message 4. If STA_P has an existing PeerKey state for this session, i.e., STA_P has received message 2 and this message is a follow-up to that. If STA_P has an existing PeerKey state for this session, STA_P silently discards message 4.

c) If all checks succeed, STA_P decrypts the Key Data field of message 4 and extracts the MAC_I, the INonce, the PNonce, the SMK, and the lifetime from message 4. The STA_P verifies the extracted INonce against the INonce originally received as part of message 2.

d) The STA_P calculates the SMKID per 12.7.1.6.

e) The STA_P checks the value of the lifetime with the maximum value it can support. If the lifetime suggested by the AP is too long, the STA_P selects a lower value that it can support.

f) Using all the information, the STA_P creates the SMKSA for this PeerKey session.

12.7.8.2.6 SMK handshake message 5

Message 5 uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2

Key Information:

Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0

Key Type = 0 (Group/SMK)

SMK Message = 1 (SMK)

Install = 0

Key Ack = 0

Key MIC = 1

Secure = 1

Error = 0

Request = 0

Encrypted Key Data = 1

Reserved = 0

Key Length = Cipher-suite-specific; see Table 12-4

Key Replay Counter = request EAPOL replay counter of AP

Key Nonce = INonce

EAPOL-Key IV = 0

Key RSC = 0

Key MIC = MIC (KCK of the STA_I, EAPOL)

Key Data Length = Length of Key Data field in octets

Key Data = Encrypted peer RSNE, peer MAC address KDE, peer Nonce KDE, SMK KDE (contains SMK and INonce), Lifetime KDE

The AP sends message 5 to the STA_I. On receipt of message 5, the STA_I checks that the key replay counter corresponds to message 5. If it does not, the STA_I silently discards the message. Otherwise,

a) STA_I verifies the message 4 MIC using STA P PTKSA. If the calculated MIC does not match the MIC that the AP included in the EAPOL-Key frame, the STA_I silently discards message 5.

b) If the MIC is correct, the STA_I identifies the PeerKey session using the INonce sent as part of the Key Nonce field of message 5. If STA_I has an existing PeerKey state for this session, i.e., STA_I has initiated this message exchange using message 1 and this message is a follow-up to that. If STA_I has an existing PeerKey state for this session, STA_I shall silently discard message 5.

c) If all checks succeed, STA_I decrypts the Key Data field of message 5 and extracts the peer RSNE, the MAC_P, the INonce, the PNonce, the SMK, and the lifetime from message 5.

d) The STA_I verifies that the peer RSNE includes a valid cipher (i.e., one that was included in an initiator RSNE). If not, STA_I discards the message and sends an Error KDE ERR_CPHR_NS.

e) The STA_I calculates SMKID per 12.7.1.6.

f) The STA_I checks the value of the lifetime with the maximum value it can support. If the lifetime suggested by the AP is too long, STA_I selects a lower value that can support.

g) Using all the information, the STA_I creates the SMKSA for this PeerKey session.

12.7.8.3 PeerKey setup and handshake error conditions

If the STA_P does not receive a valid SMK message 2 or a 4-way STK message 1 after sending the EAPOL-Key request frame to initiate the PeerKey rekey within a 200 ms timeout, the STA_P shall invoke an STSL application teardown procedure.

If the STA_I does not receive an SMK message 5 from the AP, the STA_I shall attempt dot11RSNAConfigSMKUpdateCount transmits of the SMK handshake message 1 plus a final timeout. If the STA_I still has not received a response after these retries, it shall invoke an STSL application teardown procedure. The retransmit timeout value shall be 200 ms for the first timeout, the listen interval for the second timeout, and twice the listen interval for subsequent timeouts. If there is no listen interval or the listen interval is zero, then 200 ms shall be used for all timeout values.

There is no specific recovery mechanism at the AP if the SMK message 3 is dropped. This results in a timeout by the STA_I after nonreceipt of SMK message 5, as described in the preceding paragraph.

If the SMK message 4 is not received by the STA_P, a failure is detected during the 4-way STK handshake. In this case, the STA_P discards the EAPOL-Key frames without the proper key. This failure is covered by behavior described in 12.7.6.6 and results in teardown of the STSL.

Upon receipt of the SMK message 5, the STA_I transmits message 1 of the 4-way STK handshake to the STA_P. If the STA_I does not receive message 2 of the 4-way STK handshake from the STA_P, it shall attempt dot11RSNAConfigSMKUpdateCount transmits of 4-way STK handshake message 1, plus a final timeout. If STA_I still has not received a response after these retries, it shall invoke an STSL application teardown procedure. The retransmit timeout value shall be 100 ms for the first timeout, half the listen interval for the second timeout, and the listen interval for subsequent timeouts. If there is no listen interval or the listen interval is zero, then 100 ms shall be used for all timeout values.

There is no specific recovery mechanism at the STA_P if the SMK message 3 is lost. This results in a timeout on the STA_I, as described in the preceding paragraph, and a subsequent reinitiation of the SMK handshake. The STA_P shall allow reinitiation of the SMK handshake at any point prior to receipt of SMK message 4.

12.7.8.4 STKSA rekeying

Rekeying is always initiated by the STA_I. When needed, the STA_P sends an EAPOL-Key request frame to the STA_I to request rekeying. The STA_P shall wait a minimum of one half the IEEE 802.1X timeout after the STSL setup before initiating a PeerKey rekey procedure. To perform rekeying, there are two cases:

a) If SMK timer has not expired, the STAs initiate a 4-way handshake to create a new STK. The 4-way handshake is always initiated by the STA_I. In this case, the STA_P should not delete any existing STKSA prior to verifying message 3 of the 4-way handshake with STA_I for this session.

b) If the SMK has expired, the STA_I shall not use an existing STKSA and shall start the SMK handshake followed by a 4-way handshake to create new keys.

The format of the EAPOL-Key request frame in case a) from STA_P to STA_I is as follows:

Request message: STA_P → STA_I: EAPOL-Key(1,1,0,0,1,0,0,0, MIC, PMKID KDE)

The request message uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2
Key Information:
Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0
Key Type = 1 (PTK)
SMK Message = 0
Install = 0
Key Ack = 0
Key MIC = 1
Secure = 1
Error = 0
Request = 1
Encrypted Key Data = 0

Reserved = 0
Key Length = 0
Key Replay Counter = request replay counter of peer STA
Key Nonce = 0
EAPOL-Key IV = 0
Key RSC = 0
Key MIC = MIC computed over the body of this EAPOL-Key frame
Key Data Length = Length of Key Data field in octets
Key Data = SMKID in SMKID KDE

12.7.8.5 Error reporting

12.7.8.5.1 General

Error reporting messages are defined in this subclause and used to report errors whenever STAs or an AP detect an error during the SMK handshake.

The AP, upon receiving the error messages defined in this subclause or upon generating the error messages defined in this subclause, should log the error. The STA, upon receipt of the error messages defined in this subclause, shall tear down the STSL with the other STA and clear all of the PeerKey states.

The format of EAPOL-Key request frame for reporting an error message is as follows:

Error message: EAPOL-Key(1,1,0,0,0,1,0, 0, MIC, Error KDE, MAC Address KDE).

The request message uses the following values for each of the EAPOL-Key frame fields:

Descriptor Type = N – see 12.7.2
Key Information:
Key Descriptor Version = 1 (ARC4 encryption with HMAC-MD5) or 2 (NIST AES key wrap with HMAC-SHA-1-128) or 3 (NIST AES key wrap with AES-128-CMAC), in all other cases 0
Key Type = 0 (Group/SMK)
SMK Message = 1 (SMK)
Install = 0
Key Ack = 0
Key MIC = 1
Secure = 1
Error = 1
Request = 1 when the message is going from the STA to an AP or 0 when the message is going from an AP to the STA
Encrypted Key Data = 0
Reserved = 0
Key Length = 0
Key Replay Counter = request EAPOL replay counter
Key Nonce = 0
EAPOL-Key IV = 0
Key RSC = 0

Key MIC = MIC computed over the body of this EAPOL-Key frame

Key Data Length = Length of Key Data field in octets

Key Data = Error KDE (different types defined in Table 12-7), MAC Address KDE

12.7.8.5.2 Error ERR_STA_NR

This error message is sent whenever an AP finds that the STA to which it needs to send a message is not reachable. In response to this error, the AP creates an Error KDE with error type ERR_STA_NR and sends the message back to the other STA involved in the handshake. The MAC address KDE contains the MAC address of the unreachable STA.

12.7.8.5.3 Error ERR_STA_NRSN

This error message is sent whenever the AP finds that the STA to which it needs to send the message does not have a secure RSNA connection. In response of this error, the AP creates an Error KDE with error type ERR_STA_NRSN and sends the message back to the STA from which it received the last message. The MAC address KDE contains the MAC address of the STA with which the AP does not have a secure RSNA connection.

12.7.8.5.4 Error ERR_CPHR_NS

This error message is sent whenever a STA finds that it does not support any of the cipher suites proposed by the other STA. In response to this error, the STA creates an Error KDE with error type ERR_CPHR_NS and sends the message back to the other STA. The MAC address KDE contains the MAC address of the other STA.

12.7.8.5.5 Error ERR_NO_STSL

This error message is sent whenever a STA finds that it does not have an existing STSL with the other STA. In response of this error, the STA creates an Error KDE with error type ERR_NO_STSL and sends the message back to the other STA. The MAC address KDE contains the MAC address of the other STA.

12.7.9 TDLS PeerKey (TPK) security protocol

12.7.9.1 General

The TPK security protocol is executed between the two non-AP STAs that intend to establish an RSNA for direct-link communication. If any security method (pre-RSNA or RSNA) is enabled on the connection between a STA and the AP, the STA shall require that the TPK security protocol complete successfully before using a direct link. If no security method is enabled on the connection between a STA and the AP, the STA shall not use the TPK security protocol on the direct link. A STA may refuse to set up a TDLS link when the protection on the STA link to the AP is secured with a weak algorithm or when the link between the STA and the AP is not using any security.

12.7.9.2 TPK handshake

The TPK handshake occurs as part of the TDLS direct-link setup procedure. The TPKSA is the result of the successful completion of the TPK handshake protocol, which derives keys for providing confidentiality and data origin authentication.

In order to maintain TPK confidentiality, both the TDLS initiator STA and the TDLS responder STAs establish an RSNA with their common AP prior to executing the TPK handshake. To meet this criterion, a STA may refuse to initiate the TDLS direct link if:

a) The AP does not include an RSNE in its Beacon and Probe Response frames to advertise the availability of security;
b) The AP's RSNE indicates that WEP-40 (OUI 00-0F-AC:1) or WEP-104 (OUI 00-0F-AC:5) are enabled as either pairwise or group cipher suites; or
c) The AP's RSNE indicates that Use group cipher suite (00-0F-AC:0) is used as the pairwise cipher suite.

Violation of any of these cases would cause the TPK handshake to leak the TPK.

The TDLS initiator STA and the TDLS responder STA perform the following exchange to set up a TPK:

TDLS PMK handshake message 1: TDLS initiator STA → TDLS responder STA:
Link Identifier element, RSNE, Timeout Interval element, FTE
TDLS PMK handshake message 2: TDLS responder STA → TDLS initiator STA:
Link Identifier element, RSNE, Timeout Interval element, FTE
TDLS PMK handshake message 3: TDLS initiator STA → TDLS responder STA:
Link Identifier element, RSNE, Timeout Interval element, FTE

where

The TDLS initiator STA Address field of the Link Identifier element is the MAC address of the TDLS initiator STA
The TDLS responder STA Address field of the Link Identifier element is the MAC address of the TDLS responder STA
The PairwiseCipherSuite field of the RSNE identifies the cipher suite used to protect the Data frames sent over the direct link
The AKM suite list of the RSNE identifies which Authentication Method was used
The TimeoutIntervalType field of the Timeout Interval element is the key lifetime
The SNonce field of the FTE is a 256 bit value randomly generated by the TDLS initiator STA
The ANonce field of the FTE is a 256 bit value randomly generated by the TDLS responder STA (set to 0 in message 1)
The MIC field of the FTE is 0 for message 1 and computed as described in 12.7.9.4.3 and 12.7.9.4.4 for messages 2 and 3 respectively

The TDLS PMK handshake message 1 shall be transmitted in the TDLS Setup Request frame.

TDLS PMK handshake message 2 shall be transmitted in the TDLS Setup Response frame.

TDLS PMK handshake message 3 shall be transmitted in the TDLS Setup Confirm frame.

The TPK shall be derived as follows:

TPK-Key-Input = Hash(min (SNonce, ANonce) || max (SNonce, ANonce))
TPK = KDF-Hash-Length(TPK-Key-Input, "TDLS PMK", min (MAC_I, MAC_R)
|| max (MAC_I, MAC_R) || BSSID)

where

Hash is the hash algorithm identified by the negotiated AKM suite selector specified in Table 9-133
KDF-Hash-Length is the key derivation function defined in 12.7.1.7.2 that uses Hash to generate a key whose length is TK_bits + 128
TK_bits is cipher-suite specific and specified in Table 12-4

MAC_I and MAC_R are the MAC addresses of the TDLS initiator STA and the TDLS responder STA, respectively

SNonce and ANonce are the nonces generated by the TDLS initiator STA and TDLS responder STA, respectively, for this instance of the TPK handshake. The BSSID is set to the BSSID of the current association of the TDLS initiator STA.

Each TPK has two component keys—TPK-KCK and TPK-TK, defined as follows:

The Key Confirmation Key (KCK) shall be computed as the first 128 bits (bits 0–127) of the TPK

TPK-KCK = L(TPK, 0, 128)

The KCK is used to provide data origin authenticity in TDLS Setup Response and TDLS Setup Confirm frames.

The temporal key (TK) shall be computed as the remaining bits (for CCMP-128, the second 128 bits, i.e., bits 128–255) of the TPK

TPK-TK = L(TPK, 128, Length – 128)

The TPK-TK is used to provide confidentiality for direct-link data.

The temporal key is configured into the STA by the SME through the use of the MLME-SETKEYS.request primitive.

12.7.9.3 TPK handshake security assumptions

The security of the TDLS PMK handshake depends on the following:

a) The TDLS initiator STA and the TDLS peer STA each have an RSNA established with the AP that is being used for TDLS Setup.
b) The AP does not expose the nonces exchanged by the TDLS initiator STA and the TDLS responder STA to any external party.
c) The AP does not use these nonces to derive the TPK and attack the TDLS direct-link instance.
d) TDLS message security (encryption and integrity computations) processing at the AP is protected from illegal eavesdropping, alterations, insertions and substitutions.
e) The TDLS initiator STA and TDLS responder STAs do not expose SNonce, ANonce, or the derived key to a third party.
f) The TDLS initiator STA and the TDLS peer STA are associated to the same AP.

12.7.9.4 TPK Security Protocol handshake messages

12.7.9.4.1 Overview

The TPK handshake consists of three messages. Each message comprises a number of elements and is included in the TDLS Setup Request, TDLS Setup Response, and TDLS Setup Confirm.

In an RSN, these handshake messages serve to provide a session identifier, are identified by the nonces, and are used as association instance identifiers. These nonces are chosen randomly or pseudorandomly, and are used to generate the TPK.

12.7.9.4.2 TPK handshake message 1

If the TDLS initiator STA has security enabled on the link with the AP, it shall add an RSNE, FTE, and Timeout Interval element to its TDLS Setup Request frame. The elements shall be formatted as follows:

The RSNE, if present, shall be set as follows:

Version shall be set to 1.

The pairwise cipher suite list field indicating the pairwise cipher suites the TDLS initiator STA is willing to use with the TPKSA. WEP-40, WEP-104, and TKIP shall not be included in this list.

The group cipher suite shall be set to 00-0F-AC:7.

The AKM suite count field shall be set to 1.

The AKM suite list field shall be set to indicate TPK handshake (00-0F-AC:7).

In the RSN Capabilities field, the No Pairwise subfield shall be set to 0 and the PeerKey Enabled subfield shall be set to 1.

PMKID Count subfield, if present, shall be set to 0.

PMKID list shall not be present.

The Group Management Cipher Suite subfield, if present, shall be set to 00-0F-AC:7.

The Timeout Interval element indicates the lifetime of the TPKSA. The Lifetime Interval Type shall be set to ‘2’ (Key Lifetime Interval). The minimum lifetime shall be 300 seconds.

The FTE shall be set as follows:

SNonce shall be set to a value chosen randomly by the TDLS initiator STA, following the recommendations of 12.7.5.

All other fields shall be set to 0.

The TDLS initiator STA sends message 1 to the TDLS responder STA.

On reception of message 1, the TDLS responder STA checks whether the RSNE is present.

If the TDLS responder STA does not have security enabled on the link with the AP, it shall reject the request with status code SECURITY_DISABLED.

If the TDLS responder STA has security enabled on the link with the AP, it checks whether the request includes an RSNE and FTE. If not, the TDLS responder STA shall reject the request with status code INVALID_PARAMETERS.

If the version field of the RSNE is 0, then the TDLS responder STA shall reject the request with status code UNSUPPORTED_RSNE_VERSION.

Otherwise, the TDLS responder STA processes the message as follows:

If the contents of the RSNE do not indicate AKM of TPK handshake (suite type 00-0F-AC:7), the TDLS responder STA shall reject the request with status code STATUS_INVALID_AKMP.

If none of the pairwise cipher suites are acceptable, or pairwise ciphers include WEP-40, WEP-104, or TKIP, then the TDLS responder STA shall reject the TDLS Setup Request frame with status code STATUS_INVALID_PAIRWISE_CIPHER.

If the RSN Capabilities field has not set the subfields according to the described rules for this message, then the TDLS responder STA rejects with status code INVALID_RSNE_CAPABILITIES.

If the suggested lifetime is unacceptable or below the default value, the TDLS responder STA shall reject the TDLS Setup Request frame with status code UNACCEPTABLE_LIFETIME.

If the contents of the FTE are not as per specified for this message, then the TDLS responder STA shall reject the TDLS Setup Request frame with status code STATUS_INVALID_FTE.

The TDLS responder STA shall ignore the remaining fields in the RSNE, FTE, and Timeout Interval element.

Otherwise, the TDLS responder STA shall respond as specified in 11.23.4.

12.7.9.4.3 TPK handshake message 2

If the TDLS responder STA validates the TPK handshake message 1 for this TDLS instance, the TDLS responder STA may respond with TPK handshake message 2. To do so, the TDLS responder STA shall add an RSNE, FTE, and Timeout Interval element to its TDLS Setup Response frame. The elements shall be formatted as follows:

The RSNE shall include the following:

Include a pairwise cipher suite from one of those presented in RSNE of message 1 of this sequence in the pairwise cipher suite list, and set the pairwise cipher suite count to 1.

The version number shall be the minimum of the maximum version supported by the TDLS responder STA and the version number received in the RSNE of message 1.

All other RSNE fields shall be same as those received in message 1.

The Timeout Interval element shall be the same as that received in the TPK handshake message 1.

The FTE shall include the following:

ANonce shall be set to a value chosen randomly by the TDLS responder STA, following the recommendations of 12.7.5.

SNonce shall be same as that received in message 1 of this sequence

The MIC shall be calculated on the concatenation, in the following order, of:

TDLS initiator STA MAC address (6 octets)

TDLS responder STA MAC address (6 octets)

Transaction Sequence number (1 octet) which shall be set to the value 2

Link Identifier element

RSNE

Timeout Interval element

FTE, with the MIC field of the FTE set to 0.

The MIC shall be calculated using the TPK-KCK and the AES-128-CMAC algorithm. The output of the AES-128-CMAC shall be 128 bits.

All other fields shall be set to 0.

The TDLS responder STA shall use the MLME-SETKEYS.request primitive to configure the Temporal Key into its STA prior to sending message 2.

The TDLS responder STA sends message 2 to the TDLS initiator STA. The TDLS initiator STA shall process message 2 as follows:

If the TDLS initiator STA Address and TDLS responder STA Address of the Link Identifier element do not match those for an outstanding TDLS Setup Request, the TDLS initiator STA shall silently discard the received TDLS Setup Response frame.

If the SNonce field of the FTE does not match that of an outstanding request to the TDLS responder STA, then the TDLS initiator STA shall silently discard the received TDLS Setup Response frame.

Otherwise, the TDLS initiator STA shall compute the TPK and then validate the MIC in the FTE as specified in MIC calculation procedure for TPK handshake message 2. If invalid, the TDLS initiator STA shall discard the message.

If the version of the RSNE is 0 or is greater than the version of the RSNE sent in message 1, then the TDLS initiator STA shall reject the response with status code UNSUPPORTED_RSNE_VERSION. Otherwise,

If the contents of the RSNE, with the exception of the pairwise cipher suite count and pairwise cipher suite list are not the same as those sent by the TDLS initiator STA in message 1 of this sequence, then the TDLS initiator STA shall reject the response with status code INVALID_RSNE.

If the pairwise cipher suite count is other than 1, then the TDLS initiator STA shall reject the response with status code STATUS_INVALID_PAIRWISE_CIPHER.

If the selected pairwise cipher suite was not included in the Initiator's request, then the TDLS initiator STA shall reject the TDLS Setup Response frame with status code STATUS_INVALID_PAIRWISE_CIPHER.

If the Timeout Interval element is not the same as that sent in message 1, the TDLS initiator STA shall reject the TDLS Setup Response frame with status code UNACCEPTABLE_LIFETIME.

If the BSSID in the Link Identifier element is different from the one sent in message 1, then the TDLS initiator STA shall reject the response with status code NOT_IN_SAME_BSS.

If the TDLS initiator STA validates TDLS message 2, the TDLS initiator STA shall create a TPKSA and respond with message 3 as defined in 11.23.4. The TDLS initiator STA shall use the MLME-SETKEYS.request primitive to configure the Temporal Key into its STA prior to sending message 3.

12.7.9.4.4 TPK handshake message 3

If the TDLS initiator STA responds to message 2 for this TDLS instance, the TDLS initiator STA shall add an RSNE, FTE, and Timeout Interval element to its TDLS Setup Confirm frame. The elements shall be formatted as follows:

The RSNE shall be the same as the RSNE received in message 2.

The Timeout Interval element shall be the same as that received in the TPK handshake message 2.

With the exception of the MIC field, the contents of the FTE shall be the same as the FTE received in message 2.

The MIC shall be calculated on the concatenation, in the following order, of:

TDLS initiator STA MAC address (6 octets)

TDLS responder STA MAC address (6 octets)

Transaction Sequence number (1 octet), which shall be set to the value 3

Link Identifier element

RSNE

Timeout Interval element

FTE, with the MIC field of the FTE set to 0.

The MIC shall be calculated using the TPK-KCK and the AES-128-CMAC algorithm. The output of the AES-128-CMAC shall be 128 bits.

All other fields shall be set to 0.

The TDLS initiator STA sends message 3 to the TDLS responder STA. The TDLS responder STA shall process message 3 as follows:

If the Source and Destination Addresses of the Link Identifier element do not match those for an outstanding TDLS Setup Request, the TDLS responder STA shall discard the message.

If the ANonce and SNonce fields of the FTE do not match that of an outstanding request to the TDLS initiator STA, then the TDLS responder STA shall discard the message.

Otherwise, the TDLS responder STA shall validate the MIC in the FTE as specified in the MIC calculation procedure for TPK handshake message 3. If invalid, the TDLS responder STA shall discard the message.

The TDLS responder STA shall discard the message, the TDLS responder STA shall abandon the TPK handshake identified by the <ANonce, SNonce> combination, and delete existing TPK handshake key state for this sequence if any of the following checks fail:

The contents of the RSNE are not the same as that sent by the TDLS responder STA in message 2

The Timeout Interval element is not the same as that sent in message 2

The BSSID from the Link Identifier element is not the same as that sent in message 2

On successful processing of message 3, the TPK handshake is considered successful.

The TPKSA shall be deleted by the TDLS responder STA if it does not receive a valid TPK handshake message 3 from the TDLS Initiator STA within dot11TDLSResponseTimeout.

12.7.9.5 Supplicant state machine procedures

The following list summarizes the procedures used by the Supplicant state machine:

— **STADisconnect** – The Supplicant invokes this procedure to disassociate and deauthenticate its STA from the AP.

— **MIC**(*x*) – The Supplicant invokes this procedure to compute a MIC of the data *x*.

— **CheckMIC**() – The Supplicant invokes this procedure to verify a MIC computed by the MIC() function.

— **StaProcessEAPOL-Key** – The Supplicant invokes this procedure to process a received EAPOL-Key frame. The pseudocode for this procedure is as follows:

```
StaProcessEAPOL-Key (S, M, A, I, K, RSC, ANonce, RSC, MIC, RSNE, GTK[N], IGTK[M], IPN)
    TPTK ← PTK
    TSNonce ← 0
    PRSC ← 0
    UpdatePTK ← 0
    State ← UNKNOWN
    if M = 1 then
        if Check MIC(PTK, EAPOL-Key frame) fails then
            State ← FAILED
        else
            State ← MICOK
        endif
    endif
    if K = P then
        if State ≠ FAILED then
            if PSK exists then – PSK is a preshared key
                PMK ← PSK
            else
                PMK ← L(MSK, 0, PMK_bits)
            endif
            TSNonce ← SNonce
            if ANonce ≠ PreANonce then
                TPTK ← Calc PTK(PMK, ANonce, TSNonce)
                PreANonce ← ANonce
            endif
```

```
if State = MICOK then
    PTK ← TPTK
    UpdatePTK ← I
    if UpdatePTK = 1 then
        if no GTK then
            PRSC ← RSC
        endif
        if MLME-SETKEYS.request(0, true, PRSC, PTK) fails then
            invoke MLME-
            DEAUTHENTICATE.request
        endif
        MLME-SETPROTECTION.request(TA, Rx)
    endif
    if GTK then
        if (GTK[N] ← Decrypt GTK) succeeds then
            if MLME-SETKEYS.request(N, 0, RSC, GTK[N]) fails then
                invoke MLME-DEAUTHENTICATE.request
            endif
        else
            State ← FAILED
        endif
    endif
    if IGTK then
        if (IGTK[M] ← Decrypt IGTK) succeeds then
            if MLME-SETKEYS.request(M, 0, IPN, IGTK[M]) fails then
                invoke MLME-DEAUTHENTICATE.request
            endif
        else
            State ← FAILED
        endif
    endif
  endif
 endif
else if KeyData = GTK then
    if State = MICOK then
        if (GTK[N] ← Decrypt GTK) succeeds then
            if MLME-SETKEYS.request(N, T, RSC, GTK[N]) fails then
                invoke MLME-DEAUTHENTICATE.request
            endif
        else
            State ← FAILED
        endif
        if (IGTK[M] ← Decrypt IGTK) succeeds then
            if MLME-SETKEYS.request(M, T, IPN, IGTK[M]) fails then
                invoke MLME-DEAUTHENTICATE request
            endif
        else
            State ← FAILED
        endif
    else
        State ← FAILED
    endif
endif
if A = 1 && State ≠ Failed then
```

```
        Send EAPOL-Key(0,1,0,0,K,0,0,TSNonce,MIC(TPTK),RSNE)
    endif
    if UpdatePTK = 1 then
        MLME-SETPROTECTION.request(TA, Rx_Tx)
    endif
    if State = MICOK && S = 1 then
        MLME-SETPROTECTION.request(TA, Rx_Tx)
        if IBSS then
            keycount++
            if keycount = 2 then
                802.1X::portValid← true
            endif
        else
            802.1X::portValid← true
        endif
    endif
```

Here UNKNOWN, MICOK, and FAILED are values of the variable State used in the Supplicant pseudocode. State is used to decide how to do the key processing. MICOK is set to 1 when the MIC of the EAPOL-Key has been checked and is valid. FAILED is used when a failure has occurred in processing the EAPOL-Key frame. UNKNOWN is the initial value of the variable State.

When processing 4-way handshake message 3, the GTK and IGTK are decrypted from the EAPOL-Key frame and installed. The PTK shall be installed before the GTK and IGTK.

The Key Replay Counter field used by the Supplicant for EAPOL-Key frames that are sent in response to a received EAPOL-Key frame shall be the received Key Replay Counter field. Invalid EAPOL-Key frames such as invalid MIC, GTK without a MIC, etc., shall be ignored.

NOTE 1—TPTK is used to stop attackers changing the PTK on the Supplicant by sending the first message of the 4-way handshake. An attacker can still affect the 4-way handshake while the 4-way handshake is being carried out.

NOTE 2—The PMK is supplied by the authentication method used with IEEE Std 802.1X-2010 if preshared mode is not used.

NOTE 3—A PTK is configured into the encryption/integrity engine depending on the Tx/Rx bit, but if configured, is always a transmit key. A GTK is configured into the encryption/integrity engine independent of the state of the Tx/Rx bit, but whether the GTK is used as a transmit key is dependent on the state of the Tx/Rx bit.

— **CalcGTK**(x) – Generates the GTK.
— **DecryptGTK(x)** – Decrypt the GTK from the EAPOL-Key frame.
— **DecryptIGTK(x)** – Decrypt the IGTK from the EAPOL-Key frame.

12.7.9.6 Supplicant PeerKey state machine states

Figure 12-48 depicts the PeerKey handshake Supplicant key management state machine. The following list summarizes the states the Supplicant state machine uses to support the PeerKey handshake:

— **STKINIT:** This state is the idle state and is entered when the IEEE 802.1X Supplicant completes successful Authentication.
— **SMKNEGOTIATING1:** This state is entered when the MLME-STKSTART.request primitive is received for the SMK handshake by the initiator STA.
— **SMKNEGOTIATING2:** This state is entered when the first EAPOL-Key frame of the SMK handshake is received by the peer STA.
— **SMKNEGOTIATING3:** This state is entered when the fifth EAPOL-Key frame of the SMK handshake is received by the initiator STA.
— **SMKNEGOTIATING4:** This state is entered when the fourth EAPOL-Key frame of the SMK handshake is received by the peer STA.

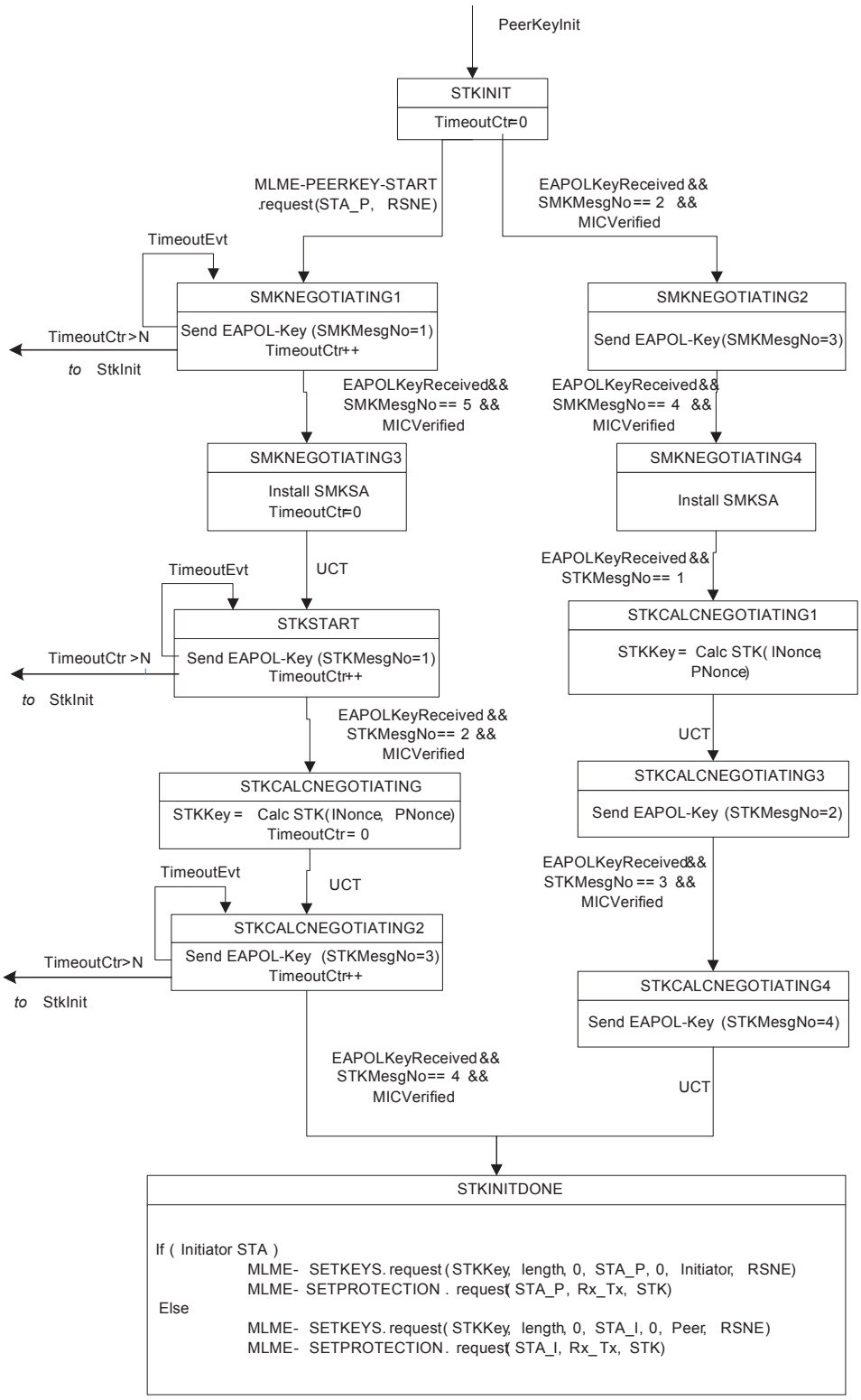

Figure 12-48—PeerKey handshake Supplicant key management state machine

— **STKSTART:** Once the SMKSA is created, the initiator STA enters this state. This is the start of 4-Way STK handshake.
— **STKCALCNEGOTIATING:** This state is entered when the second EAPOL-Key frame of the 4-Way STK handshake is received by the initiator STA and the MIC is verified.
— **STKCALCNEGOTIATING1:** This state is entered when the first EAPOL-Key frame of the 4-Way STK handshake is received by the peer STA and the MIC is verified.
— **STKCALCNEGOTIATING2:** This state is entered unconditionally by the initiator STA.
— **STKCALCNEGOTIATING3:** This state is entered unconditionally by the peer STA.
— **STKCALCNEGOTIATING4:** This state is entered when the third EAPOL-Key frame of the 4-Way STK handshake is received by the peer STA and the MIC is verified.
— **STKINITDONE:** This state is entered by the initiator STA when the fourth EAPOL-Key frame of the 4-way STK handshake is received. This state is entered by the peer STA when the fourth EAPOL-Key frame of the 4-way STK handshake is sent.

12.7.9.7 Supplicant PeerKey state machine variables

The following list summarizes the variables used by the Supplicant state machine:

— *PeerKeyInit* – This variable is used to initialize the PeerKey state machine.
— *TimeoutEvt* – This variable is set to true if the EAPOL-Key frame sent fails to obtain a response from the Supplicant. The variable may be set by management action or set by the operation of a timeout while in the different states.
— *TimeoutCtr* – This variable maintains the count of EAPOL-Key receive timeouts. It is incremented each time a timeout occurs on EAPOL-Key receive event and is initialized to 0. The Key Replay Counter field value for the EAPOL-Key frame shall be incremented on each transmission of the EAPOL-Key frame.
— *MICVerified* – This variable is set to true if the MIC on the received EAPOL-Key frame is verified and is correct. Any EAPOL-Key frames with an invalid MIC are dropped and ignored.
— *SMKMesgNo* – This variable indicates SMK handshake EAPOL-Key frame types. Details for each message type (1-5) are provided in 12.7.8.
— *STKMesgNo* – This variable indicates 4-way STK handshake EAPOL-Key frame types. Details for each message type (1-4) are provided in 12.7.6.
— *STA_P* – This variable indicates the MAC address of the peer STA participating in the PeerKey handshake.
— *STA_I* – This variable indicates the MAC address of the initiator STA participating in the PeerKey handshake.
— *STKKey* – The STK generated as a result of the 4-way STK handshake.
— *EAPOLKeyReceived* – The Supplicant sets this variable to true when it receives an EAPOL-Key frame.

12.7.10 RSNA Supplicant key management state machine

12.7.10.1 General

The Supplicant shall reinitialize the Supplicant state machine whenever its system initializes. A Supplicant enters the AUTHENTICATION state on an event from the MAC that requests another STA to be authenticated. A Supplicant enters the STAKEYSTART state on receiving an EAPOL-Key frame from the Authenticator. If the MIC or any of the EAPOL-Key frames fails, the Supplicant silently discards the frame. Figure 12-49 depicts the RSNA Supplicant state machine.

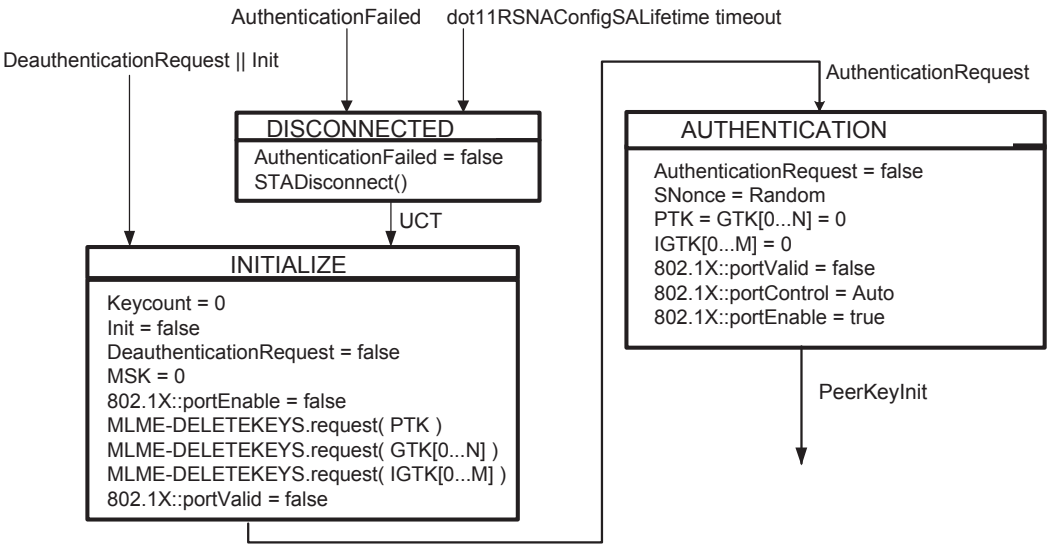

Figure 12-49—RSNA Supplicant key management state machine

Unconditional transfer (UCT) means the event triggers an immediate transition.

This state machine does not use timeouts or retries. The IEEE 802.1X state machine has timeouts that recover from authentication failures, etc.

In order to authenticate an Authenticator, the management entity sends an authentication request event. This might be before or after the STA associates to the AP. In an IBSS environment, the event is generated when a Probe Response frame is received.

12.7.10.2 Supplicant state machine states

The following list summarizes the states of the Supplicant state machine:

— **AUTHENTICATION**: A Supplicant enters this state when it sends an IEEE 802.1X AuthenticationRequest to authenticate to an SSID.

— **DISCONNECTED**: A Supplicant enters this state when IEEE 802.1X authentication fails. The Supplicant executes StaDisconnect and enters the INITIALIZE state.

— **INITIALIZE**: A Supplicant enters this state from the DISCONNECTED state, when it receives Disassociation or Deauthentication messages or when the STA initializes, causing the Supplicant to initialize the key state variables.

— **STAKEYSTART:** A Supplicant enters this state when it receives an EAPOL-Key frame. All the information to process the EAPOL-Key frame is in the message and is described in the StaProcessEAPOL-Key procedure.

12.7.10.3 Supplicant state machine variables

The following list summarizes the variables used by the Supplicant state machine:

— *DeauthenticationRequest* – The Supplicant sets this variable to true if the Supplicant's STA reports it has received Disassociation or Deauthentication messages.

— *AuthenticationRequest* – The Supplicant sets this variable to true if its STA's IEEE 802.11 management entity reports that an SSID is to be authenticated. This might be on association or at other times.

— *AuthenticationFailed* – The Supplicant sets this variable to true if the IEEE 802.1X authentication failed. The Supplicant uses the MLME-DISASSOCIATE.request primitive to cause its STA to disassociate from the Authenticator's STA.

— *EAPOLKeyReceived* – The Supplicant sets this variable to true when it receives an EAPOL-Key frame.

— *IntegrityFailed* – The Supplicant sets this variable to true when its STA reports that a fatal data integrity error (e.g., michael failure) has occurred.

NOTE—A michael failure is not the same as MICVerified because IntegrityFailed is generated if the michael integrity check fails; MICVerified is generated from validating the EAPOL-Key integrity check. Note also the STA does not generate this event for ciphers other than TKIP because countermeasures are not required.

— *MICVerified* – The Supplicant sets this variable to true if the MIC on the received EAPOL-Key frame verifies as correct. The Supplicant silently discards any EAPOL-Key frame received with an invalid MIC.

— *SNonce* – This variable represents the Supplicant's nonce.

— *PTK* – This variable represents the current PTK.

— *TPTK* – This variable represents the current PTK until message 3 of the 4-way handshake arrives and is verified.

— *GTK[]* – This variable represents the current GTKs for each group key index.

— *IGTK[]* – This variable represents the current IGTKs for each group management key index.

— *PMK* – This variable represents the current PMK.

— *keycount* – This variable is used in IBSS mode to decide when all of the keys have been delivered and an IBSS link is secure.

— *802.1X::XXX* – This variable denotes another IEEE 802.1X state variable *XXX* not specified in this standard.

12.7.11 RSNA Authenticator key management state machine

12.7.11.1 General

There is one state diagram for the Authenticator. In an infrastructure BSS, the Authenticator is always on the AP; and in an IBSS environment, the Authenticator is on every STA.

The state diagram shown in parts in Figure 12-50 to Figure 12-53 consists of the following states:

a) The AUTHENTICATION, AUTHENTICATION2, INITPMK, INITPSK, PTKSTART, PTKCALCNEGOTIATING, PTKCALCNEGOTIATING2, PTKINITNEGOTIATING, PTKINIT-DONE, DISCONNECT, DISCONNECTED, and INITIALIZE states. These states handle the initialization, 4-way handshake, teardown, and general clean-up. These states are per associated STA.

b) The IDLE, REKEYNEGOTIATING, KEYERROR, and REKEYESTABLISHED states. These states handle the transfer of the GTK to the associated client. These states are per associated STA.

c) The GTK_INIT, SETKEYS, and SETKEYSDONE states. These states change the GTK when required, trigger all of the PTK group key state machines, and update the IEEE 802.11 MAC in the Authenticator's AP when all STAs have the updated GTK. These states are global to the Authenticator.

Because there are two GTKs, responsibility for updating these keys is given to the group key state machine (see Figure 12-52). In other words, this state machine determines which GTK is in use at any time.

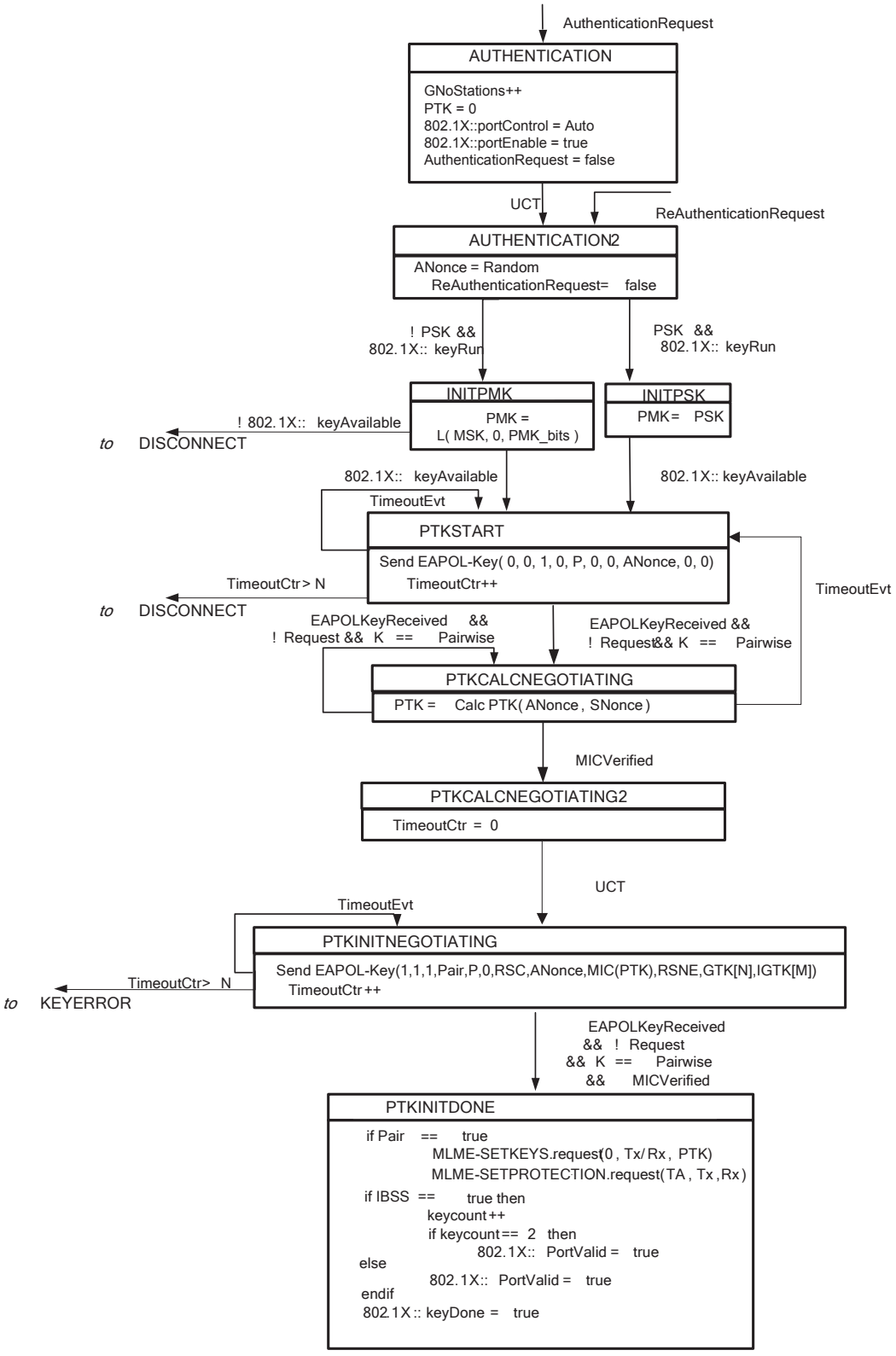

Figure 12-50—Authenticator state machines, part 1

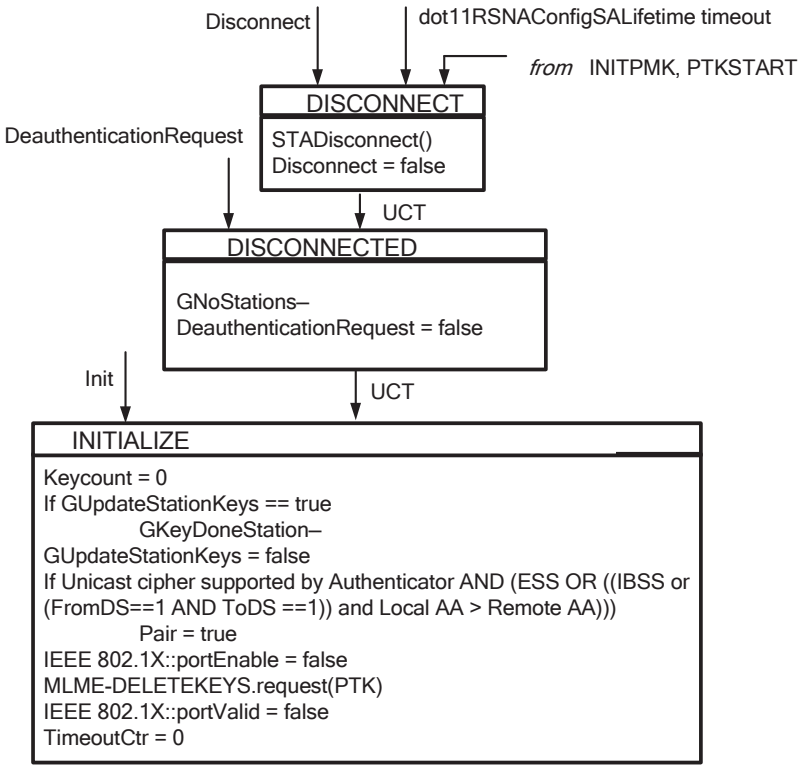

Figure 12-51—Authenticator state machines, part 2

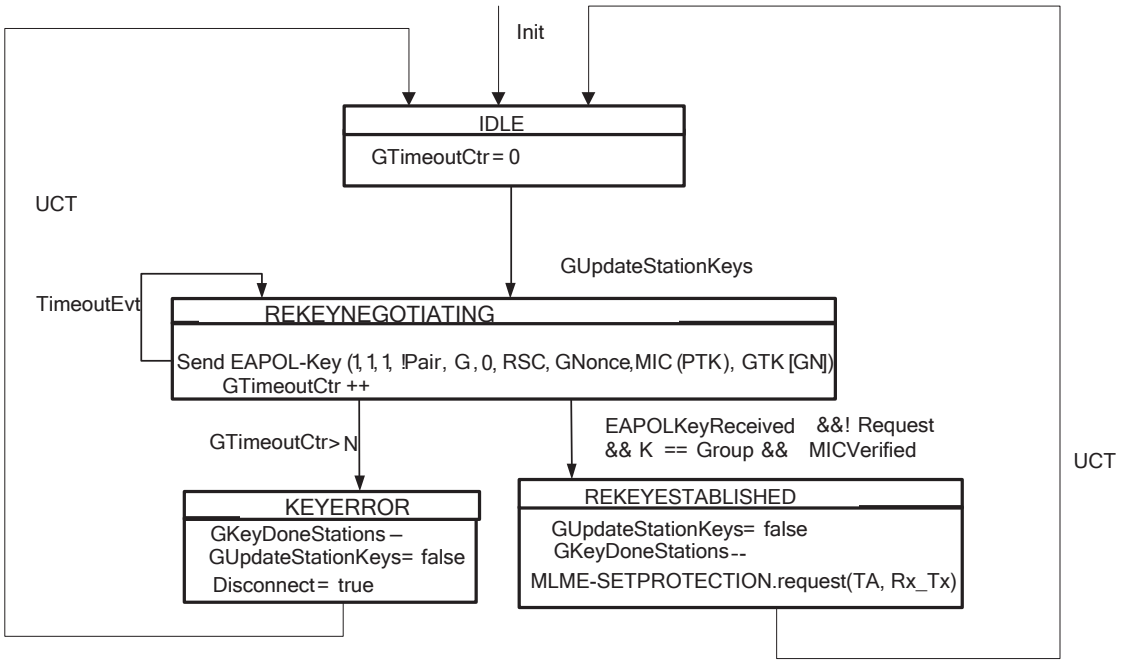

Figure 12-52—Authenticator state machines, part 3

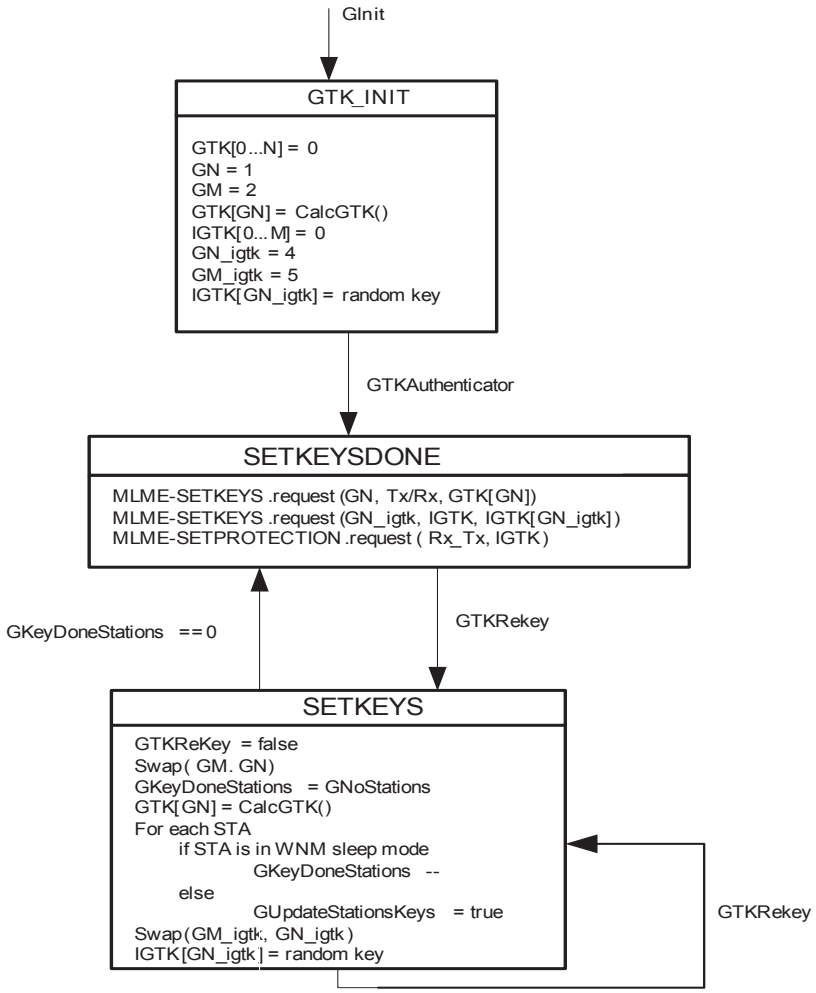

Figure 12-53—Authenticator state machines, part 4

When a second STA associates, the group key state machine is already initialized, and a GTK is already available and in use.

When the GTK is to be updated the variable GTKReKey is set to 1. The SETKEYS state updates the GTK and triggers all of the PTK group key state machines that currently exist—one per associated STA. Each PTK group key state machine sends the GTK to its STA. When all of the STAs have received the GTK (or failed to receive the key), the SETKEYSDONE state is executed which updates the APs encryption/integrity engine with the new key.

Both the PTK state machine and the PTK group key state machine use received EAPOL-Key frames as an event to change states. The PTK state machine only uses EAPOL-Key frames with the Key Type field equal to Pairwise, and the PTK group key state machine only uses EAPOL-Key frames with the Key Type field equal to Group.

12.7.11.2 Authenticator state machine states

12.7.11.2.1 Authenticator state machine: 4-way handshake (per STA)

The following list summarizes the states the Authenticator state machine uses to support the 4-way handshake:

— **AUTHENTICATION**: This state is entered when an AuthenticationRequest is sent from the management entity to authenticate a BSSID.

— **AUTHENTICATION2**: This state is entered from the AUTHENTICATION state or from the PTKINITDONE state.

— **DISCONNECT**: This state is entered if an EAPOL-Key frame is received and fails its MIC check. It sends a Deauthentication message to the STA and enters the INITIALIZE state.

— **DISCONNECTED**: This state is entered when Disassociation or Deauthentication messages are received.

— **INITIALIZE**: This state is entered from the DISCONNECTED state, when a deauthentication request event occurs, or when the STA initializes. The state initializes the key state variables.

— **INITPMK**: This state is entered when the IEEE 802.1X backend AS completes successfully. If a PMK is supplied, it goes to the PTKSTART state; otherwise, it goes to the DISCONNECTED state.

— **INITPSK:** This state is entered when a PSK is configured.

— **PTKCALCNEGOTIATING:** This state is entered when the second EAPOL-Key frame for the 4-way handshake is received with the key type of Pairwise.

— **PTKCALNEGOTIATING2:** This state is entered when the MIC for the second EAPOL-Key frame of the 4-way handshake is verified.

— **PTKINITNEGOTIATING:** This state is entered when the MIC for the second EAPOL-Key frame for the 4-way handshake is verified. When message 3 of the 4-way handshake is sent in state PTKINITNEGOTIATING, the encrypted GTK shall be sent at the end of the data field, and the GTK length is put in the GTK Length field.

— **PTKINITDONE:** This state is entered when the last EAPOL-Key frame for the 4-way handshake is received with the key type of Pairwise. This state may call SetPTK; if this call fails, the AP should detect and recover from the situation, for example, by doing a disconnect event for this association.

— **PTKSTART:** This state is entered from INITPMK or INITPSK to start the 4-way handshake or if no response to the 4-way handshake occurs.

12.7.11.2.2 Authenticator state machine: group key handshake (per STA)

The following list summarizes the states the Authenticator state machine uses to support the group key handshake:

— **IDLE:** This state is entered when no group key handshake is occurring.

— **KEYERROR:** This state is entered if the EAPOL-Key acknowledgment for the group key handshake is not received.

— **REKEYESTABLISHED:** This state is entered when an EAPOL-Key frame is received from the Supplicant with the Key Type subfield equal to Group.

— **REKEYNEGOTIATING:** This state is entered when the GTK is to be sent to the Supplicant.

NOTE—The Rx_Tx flag for sending a GTK is always the opposite of whether the pairwise key is used for data encryption/integrity or not. If a pairwise key is used for encryption/integrity, then the STA never transmits with the GTK; otherwise, the STA uses the GTK for transmit.

12.7.11.2.3 Authenticator state machine: group key handshake (global)

The following list summarizes the states the Authenticator state machine uses to coordinate a group key update of all STAs:

— **GTK_INIT:** This state is entered on system initialization.

— **SETKEYS:** This state is entered if the GTK is to be updated on all Supplicants.

— **SETKEYSDONE:** This state is entered if the GTK has been updated on all Supplicants.

NOTE—SETKEYSDONE calls SetGTK to set the GTK for all associated STAs that are not in WNM sleep mode. If this fails, all communication via this key fails, and the AP needs to detect and recover from this situation. A STA that is in WNM sleep mode will not have the current GTK installed when it wakes up and will need to get new GTK as described in the WNM sleep mode procedures in 11.2.3.18.

12.7.11.3 Authenticator state machine variables

The following list summarizes the variables used by the Authenticator state machine:

— *AuthenticationRequest* – This variable is set to true by the STA's IEEE 802.11 management entity in order to authenticate an association. This can be set to true when the STA associates or at other times.

— *ReAuthenticationRequest* – This variable is set to true if the IEEE 802.1X Authenticator received an eapStart or 802.1X::reAuthenticate is 1.

— *DeauthenticationRequest* – This variable is set to true if a Disassociation or Deauthentication message is received.

— *Disconnect* – This variable is set to true when the STA should initiate a deauthentication.

— *EAPOLKeyReceived* – This variable is set to true when an EAPOL-Key frame is received. EAPOL-Key frames that are received in response to an EAPOL-Key frame sent by the Authenticator shall contain the same Key Replay Counter field value as the Key Replay Counter field in the transmitted message. EAPOL-Key frames that contain different Key Replay Counter field values should be discarded. An EAPOL-Key frame that is sent by the Supplicant in response to an EAPOL-Key frame from the Authenticator shall not have the Ack bit set to 1. EAPOL-Key frames sent by the Supplicant not in response to an EAPOL-Key frame from the Authenticator shall have the Request bit set to 1.

  EAPOL-Key frames with a key type of Pairwise and a nonzero key index should be ignored.

  EAPOL-Key frames with a key type of Group and an invalid key index should be ignored.

NOTE—When an EAPOL-Key frame in which the Ack bit is 0 is received, then it is expected as a reply to a message that the Authenticator sent, and the replay counter is checked against the replay counter used in the sent EAPOL-Key frame. When an EAPOL-Key frame in which the Request bit is 1 is received, then a replay counter for these messages is used that is a different replay counter than the replay counter used for sending messages to the Supplicant.

— *GTimeoutCtr* – This variable maintains the count of EAPOL-Key receive timeouts for the group key handshake. It is incremented each time a timeout occurs on EAPOL-Key receive event and is initialized to 0. Annex C details the timeout values. The Key Replay Counter field value for the EAPOL-Key frame shall be incremented on each transmission of the EAPOL-Key frame.

— *GInit* – This variable is used to initialize the group key state machine. This is a group variable.

— *Init* – This variable is used to initialize per-STA state machine

— *TimeoutEvt* – This variable is set to true if the EAPOL-Key frame sent out fails to obtain a response from the Supplicant. The variable may be set to 1 by management action or set to 1 by the operation of a timeout while in the PTKSTART and REKEYNEGOTIATING states.

— *TimeoutCtr* – This variable maintains the count of EAPOL-Key receive timeouts. It is incremented each time a timeout occurs on EAPOL-Key receive event and is initialized to 0. Annex C contains details of the timeout values. The Key Replay Counter field value for the EAPOL-Key frame shall be incremented on each transmission of the EAPOL-Key frame.

— *MICVerified* – This variable is set to true if the MIC on the received EAPOL-Key frame is verified and is correct. Any EAPOL-Key frames with an invalid MIC are dropped and ignored.
— *GTKAuthenticator* – This variable is set to true if the Authenticator is on an AP or it is the designated Authenticator for an IBSS.
— *GKeyDoneStations* – Count of number of STAs left to have their GTK updated. This is a global variable.
— *GTKRekey* – This variable is set to true when a group key handshake is required. This is a global variable.
— *GUpdateStationKeys* – This variable is set to true when a new GTK is available to be sent to Supplicants.
— *GNoStations* – This variable counts the number of Authenticators so it is known how many Supplicants need to be sent the GTK. This is a global variable.
— *ANonce* – This variable holds the current nonce to be used if the STA is an Authenticator.
— *GN*, *GM* – These are the current key indices for GTKs. Swap(GM, GN) means that the global key index in GN is swapped with the global key index in GM, so now GM and GN are reversed.
— *PTK* – This variable is the current PTK.
— *GTK[]* – This variable is the current GTKs for each GTK index.
— *PMK* – This variable is the buffer holding the current PMK.
— *802.1X::XXX* – This variable is the IEEE 802.1X state variable *XXX*.
— *keycount* – This variable is used in IBSS mode to decide when all of the keys have been delivered and an IBSS link is secure.
— *WNM sleep mode* – This variable is true when the non-AP STA is in the WNM sleep mode, as described in 11.2.3.18. Otherwise, it is false.

12.7.11.4 Authenticator state machine procedures

The following list summarizes the procedures used by the Authenticator state machine:

— **STADisconnect**() – Execution of this procedure deauthenticates the STA.
— **CalcGTK**(x) – Generates the GTK.
— **MIC**(x) – Computes a MIC over the plaintext data.

12.8 Mapping EAPOL keys to IEEE 802.11 keys

12.8.1 Mapping PTK to TKIP keys

See 12.7.1.3 for the definition of the EAPOL temporal key derived from PTK.

A STA shall use bits 0–127 of the temporal key as its input to the TKIP Phase 1 and Phase 2 mixing functions.

A STA shall use bits 128–191 of the temporal key as the michael key for MSDUs from the Authenticator's STA to the Supplicant's STA.

A STA shall use bits 192–255 of the temporal key as the michael key for MSDUs from the Supplicant's STA to the Authenticator's STA.

12.8.2 Mapping GTK to TKIP keys

See 12.7.1.4 for the definition of the EAPOL temporal key derived from GTK.

A STA shall use bits 0–127 of the temporal key as the input to the TKIP Phase 1 and Phase 2 mixing functions.

A STA shall use bits 128–191 of the temporal key as the michael key for MSDUs from the Authenticator's STA to the Supplicant's STA.

A STA shall use bits 192–255 of the temporal key as the michael key for MSDUs from the Supplicant's STA to the Authenticator's STA.

12.8.3 Mapping PTK to CCMP keys

See 12.7.1.3 for the definition of the EAPOL temporal key derived from PTK.

A STA shall use the temporal key as the CCMP key for MPDUs between the two communicating STAs.

12.8.4 Mapping GTK to CCMP keys

See 12.7.1.4 for the definition of the EAPOL temporal key derived from GTK.

A STA shall use the temporal key as the CCMP key.

12.8.5 Mapping GTK to WEP-40 keys

See 12.7.1.4 for the definition of the EAPOL temporal key derived from GTK.

A STA shall use bits 0–39 of the temporal key as the WEP-40 key.

12.8.6 Mapping GTK to WEP-104 keys

See 12.7.1.4 for the definition of the EAPOL temporal key derived from GTK.

A STA shall use bits 0–103 of the temporal key as the WEP-104 key.

12.8.7 Mapping IGTK to BIP keys

See 12.7.1.5 for the definition of the IGTK. A STA shall use bits 0–127 of the IGTK as the AES-128-CMAC key, bits 0–127 of the IGTK as the AES-128-GMAC key, and bits 0–255 of the IGTK as the AES-256-GMAC key.

12.8.8 Mapping PTK to GCMP keys

See 12.7.1.3 for the definition of the EAPOL temporal key derived from PTK.

A STA shall use the temporal key as the GCMP key for MPDUs between the two communicating STAs.

12.8.9 Mapping GTK to GCMP keys

See 12.7.1.4 for the definition of the EAPOL temporal key derived from GTK.

A STA shall use the temporal key as the GCMP key.

12.9 Per-frame pseudocode

12.9.1 WEP frame pseudocode

A WEP-encapsulated MPDU is called a *WEP MPDU*. All other MPDUs are called *non-WEP MPDUs*.

A STA shall not transmit WEP MPDUs when dot11PrivacyInvoked is false. This MIB variable does not affect the reception of frames containing all or part of an MSDU or MMPDU.

if dot11PrivacyInvoked is false **then**
    the MPDU is transmitted without WEP cryptographic encapsulation
else
    if (the MPDU has an individual RA and there is an entry in dot11WEPKeyMappings for that RA) **then**
        if that entry has WEPOn equal to false **then**
            the MPDU is transmitted without WEP cryptographic encapsulation
        else
            if that entry contains a key that is null **then**
                discard the MPDU's entire MSDU and generate an MA-UNITDATA-STATUS.indication primitive to notify LLC that the MSDU was undeliverable due to a null WEP key
            else
                encrypt the MPDU using that entry's key, setting the Key ID subfield of the IV field to 0
            endif
        endif
    else
        if (the MPDU has a group RA and the Privacy subfield of the Capability Information field in this BSS is 0) **then**
            the MPDU is transmitted without WEP cryptographic encapsulation
        else
            if dot11WEPDefaultKeys[dot11WEPDefaultKeyID] is null **then**
                discard the MPDU's entire MSDU and generate an MA-UNITDATA-STATUS.indication primitive to notify LLC that the MSDU was undeliverable due to a null WEP key
            else
                WEP-encapsulate the MPDU using the key dot11WEPDefaultKeys-[dot11WEPDefaultKeyID], setting the Key ID subfield of the IV field to dot11WEPDefaultKeyID
            endif
        endif
    endif
endif

When the boolean dot11ExcludeUnencrypted is true, MPDUs with the Type subfield equal to Data and the Protected Frame subfield of the Frame Control field equal to 0 shall not be indicated at the MAC service interface, and only MSDUs successfully reassembled from successfully decrypted MPDUs shall be indicated at the MAC service interface. When receiving a frame with the Type subfield equal to Data, the values of dot11PrivacyOptionImplemented, dot11WEPKeyMappings, dot11WEPDefaultKeys, dot11WEPDefaultKeyID, and dot11ExcludeUnencrypted in effect at the time the PHY-RXSTART.indication primitive is received by the MAC shall be used according to the following decision tree:

if the Protected Frame subfield of the Frame Control field is 0 **then**
    if `dot11ExcludeUnencrypted` is true **then**

```
            discard the frame body without indication to LLC and increment
              dot11WEPExcludedCount
        else
            receive the frame without WEP decapsulation
        endif
    else
        if dot11PrivacyOptionImplemented is true then
            if (the MPDU has individual RA and there is an entry in dot11WEPKeyMappings
              matching the MPDU's TA) then
                if that entry has WEPOn equal to false then
                    discard the frame body and increment dot11WEPUndecryptableCount
                else
                    if that entry contains a key that is null then
                        discard the frame body and increment dot11WEPUndecryptableCount
                    else
                        WEP-decapsulate with that key, incrementing dot11WEPICVErrorCount if
                          the ICV check fails
                    endif
                endif
            else
                if dot11WEPDefaultKeys[Key ID] is null then
                    discard the frame body and increment dot11WEPUndecryptableCount
                else
                    WEP-decapsulate with dot11WEPDefaultKeys[Key ID], incrementing
                      dot11WEPICVErrorCount if the ICV check fails
                endif
            endif
        else
            discard the frame body and increment dot11WEPUndecryptableCount
        endif
    endif
```

12.9.2 RSNA frame pseudocode

12.9.2.1 General

STAs transmit protected MSDUs, A-MSDUs, and robust Management frames to an RA when a temporal key has been configured with a MLME-SETKEYS.request primitive and an MLME-SETPROTECTION.request primitive has been invoked with ProtectType parameter Tx or Rx_Tx to that RA. STAs expect to receive protected MSDUs, A-MSDUs, and robust Management frames from a TA when a temporal key has been configured with a MLME-SETKEYS.request primitive and an MLME-SET-PROTECTION.request primitive has been invoked with ProtectType parameter Rx or Rx_Tx from that TA. MSDUs, A-MSDUs, and robust Management frames that do not match these conditions are sent in the clear and are received in the clear.

12.9.2.2 Per-MSDU/Per-A-MSDU Tx pseudocode

```
if dot11RSNAActivated = true then
    if MSDU or A-MSDU has an individual RA and Protection for RA is off for Tx then
        transmit the MSDU or A-MSDU without protections
    else if (MPDU has individual RA and Pairwise key exists for the MPDU's RA) or (MPDU has
      a group addressed RA and network type is IBSS/PBSS and IBSS/PBSS GTK exists for
      MPDU's TA) then
        // If we find a suitable Pairwise or GTK for the mode we are in…
```

```
            if key is a null key then
                discard the entire MSDU or A-MSDU and generate one or more MA-UNITDATA-
                  STATUS.indication primitives to notify LLC that the MSDUs were undeliverable
                  due to a null key
            else
                // Note that it is assumed that no entry in the key
                // mapping table is of an unsupported cipher type
                Set the Key ID subfield of the IV field to 0.
                if cipher type of entry is AES-CCM then
                    Transmit the MSDU or A-MSDU, to be protected after fragmentation using
                    AES-CCM
                else if cipher type of entry is AES-GCM then
                    Transmit the MSDU or A-MSDU, to be protected after fragmentation using
                    AES-GCM
                else if cipher type of entry is TKIP then
                    Compute MIC using michael algorithm and entry's Tx MIC key.
                    Append MIC to MSDU
                    Transmit the MSDU, to be protected with TKIP
                else if cipher type of entry is WEP then
                    Transmit the MSDU, to be protected with WEP
                endif
            endif
        else // Else we did not find a key but we are protected, so handle the default key case or discard
            if GTK entry for Key ID contains null then
                discard the MSDU or A-MSDU and generate one or more MA-UNITDATA-
                  STATUS.indication primitives to notify the LLC that the MSDUs were
                  undeliverable due to a null GTK
            else if GTK entry for Key ID is not null then
                Set the Key ID subfield of the IV field to the Key ID.
                if MPDU has an individual RA and cipher type of entry is not TKIP then
                    discard the entire MSDU or A-MSDU and generate one or more MA-
                      UNITDATA-STATUS.indication primitives to notify the LLC that the
                      MSDUs were undeliverable due to a null key
                else if cipher type of entry is AES-CCM then
                    Transmit the MSDU or A-MSDU, to be protected after fragmentation using
                    AES-CCM
                else if cipher type of entry is AES-GCM then
                    Transmit the MSDU or A-MSDU, to be protected after fragmentation using
                    AES-GCM
                else if cipher type of entry is TKIP then
                    Compute MIC using michael algorithm and entry's Tx MIC key.
                    Append MIC to MSDU
                    Transmit the MSDU, to be protected with TKIP
                else if cipher type of entry is WEP then
                    Transmit the MSDU, to be protected with WEP
                endif
            endif
        endif
    endif
```

12.9.2.3 Per-MMPDU Tx pseudocode

```
if ((dot11RSNAActivated = true) and (frame is a robust Management frame)) then
```

if ((*dot11RSNAProtectedManagementFramesActivated* = false) **then**
Transmit the MMPDU without protection
else // *dot11RSNAProtectedManagementFramesActivated* = true
if (*dot11RSNAUnprotectedManagementFramesAllowed* = true) **then**
if (MMPDU has an individual RA) **then**
if (peer STA advertised MFPC = 1) **then**
if (Pairwise key exists for the MMPDU's RA) **then**
// Note that it is assumed that no entry in the key
// mapping table is of an unsupported cipher.
Transmit the MMPDU, to be protected after fragmentation
// see 12.9.2.5
else if (robust Action frame) then
// pairwise key was not found
Discard the MMPDU and generate an MLME.confirm primitive to notify the SME that the MMPDU was not delivered
else // Disassociation or Deauthentication
Transmit the MMPDU without protection
endif
else // (peer STA didn't advertised MFPC = 1)
Transmit the MMPDU without protection
endif
else // MMPDU has a group RA
if (IGTK exists) **then**
// if we find a suitable IGTK
Transmit the MMPDU with protection
// See 12.9.2.5
else if (MMPDU is Disassociate ||Deauthenticate ||(not a robust Action frame)) **then**
Transmit the MMPDU without protection
else
Discard the MMPDU and generate an MLME.confirm primitive to notify the SME that the MMPDU was undeliverable
endif
endif
else // *dot11RSNAUnprotectedManagementFramesAllowed* = false
if (MMPDU has an individual RA) **then**
if (peer STA advertised MFPC = 1) **then**
if (Pairwise key exists for the MMPDU's RA) **then**
// Note that it is assumed that no entry in the key
// mapping table is of an unsupported cipher.
Transmit the MMPDU, to be protected after fragmentation
// see 12.9.2.5
else if (robust Action frame) then
// pairwise key was not found

```
                        Discard the MMPDU and generate an MLME.confirm primitive to
                        notify the SME that the MMPDU was not delivered
                    else // FrameControlSubType is Disassociation or Deauthentication
                        Transmit the MMPDU without protection
                    endif
                else // peer STA didn't advertise MFPC = 1
                    Discard the MMPDU and generate an MLME.confirm primitive to notify
                    the SME that the MMPDU was not delivered
                endif
            else // MMPDU has a group RA
                if (IGTK exists) then
                    // if we find a suitable IGTK
                    Transmit the MMPDU with protection
                    // See 12.9.2.5
                else if (MMPDU is Disassociate || Deauthenticate || (not a robust Action frame))
                then
                    Transmit the MMPDU without protection
                else
                    Discard the MMPDU and generate an MLME.confirm primitive to notify
                    the SME that the MMPDU was undeliverable
                endif
            endif
        endif
    endif
else // (dot11RSNAActivated = false) or (not a robust Management frame)
    Use 12.9.2.2 to transmit the frame
endif
```

12.9.2.4 Per-MPDU Tx pseudocode

```
if dot11RSNAActivated = true then
    if MPDU is member of an MSDU that is to be transmitted without protections
        transmit the MPDU without protections
    else if MSDU or A-MSDU that MPDU is a member of is to be protected using AES-CCM
        Protect the MPDU using entry's key and AES-CCM
        Transmit the MPDU
    else if MSDU or A-MSDU that MPDU is a member of is to be protected using AES-GCM
        Protect the MPDU using entry's key and AES-GCM
        Transmit the MPDU
    else if MSDU that MPDU is a member of is to be protected using TKIP
        Protect the MPDU using TKIP encryption
        Transmit the MPDU
    else if MSDU that MPDU is a member of is to be protected using WEP
        Encrypt the MPDU using entry's key and WEP
        Transmit the MPDU
    else
        // should not arrive here
    endif
```

endif

12.9.2.5 Per-MPDU Tx pseudocode for MMPDU

if ((*dot11RSNAActivated* = true) **then**
if (MPDU is member of an MMPDU that is to be transmitted without protection) **then**
Transmit the MPDU without protection
else if (MPDU has an individual RA) **then**
Protect the MPDU using entry's TK and selected cipher from RSNE
Transmit the MPDU
else
// MPDU has a group RA
Protect the MPDU using IGTK and BIP
Transmit the MPDU
endif
endif

12.9.2.6 Per-MPDU Rx pseudocode

if *dot11RSNAActivated* = true **then**
if the Protected Frame subfield of the Frame Control field is 0 **then**
if *Protection for TA is off for Rx* **then**
Receive the unencrypted MPDU without protections
else
discard the frame body without indication to LLC **and** increment dot11WEPExcludedCount
endif
else if Protection is true for TA **then**
if ((MPDU has individual RA **and** Pairwise key exists for the MPDU's TA) **or** (MPDU has a group addressed RA **and** network type is IBSS/PBSS **and** IBSS/PBSS GTK exists for MPDU's RA)) **then**
if MPDU has individual RA **then**
lookup pairwise key using Key ID from MPDU
else
lookup group key using Key ID from MPDU
endif
if key is null **then**
discard the frame body and increment dot11WEPUndecryptableCount
else if entry has an AES-CCM key **then**
decrypt frame using AES-CCM key
discard the frame if the integrity check fails and increment dot11RSNAStats-CCMPDecryptErrors
else if entry has an AES-GCM key **then**
decrypt frame using AES-GCM key
discard the frame if the integrity check fails
else if entry has a TKIP key **then**
prepare a temporal key from the TA, TKIP key and PN
decrypt the frame using ARC4
discard the frame if the ICV fails and increment dot11RSNAStatsTKIPLocal-MicFailures
else if entry has a WEP key **then**

```
                    decrypt the frame using WEP decryption
                    discard the frame if the ICV fails and increment dot11WEPICVErrorCount
                else
                    discard the frame body and increment dot11WEPUndecryptableCount
                endif
            else if GTK for the Key ID does not exist then
                discard the frame body and increment dot11WEPUndecryptableCount
            else if GTK for the Key ID is null then
                discard the frame body and increment dot11WEPUndecryptableCount
            else if the GTK for the Key ID is a CCM key then
                decrypt frame using AES-CCM key
                discard the frame if the integrity check fails and increment dot11RSNAStatsCCMP-
                  DecryptErrors
            else if the GTK for the Key ID is a GCM key then
                decrypt frame using AES-GCM key
            else if the GTK for the Key ID is a TKIP key then
                prepare a temporal key from the TA, TKIP key and PN
                decrypt the frame using ARC4
                discard the frame if the ICV fails and increment dot11RSNAStatsTKIPICVErrors
            else if the GTK for the Key ID is a WEP key then
                decrypt the frame using WEP decryption
                discard the frame if the ICV fails and increment dot11WEPICVErrorCount
            endif
        else
            MLME-PROTECTEDFRAMEDROPPED.indication
            discard the frame body and increment dot11WEPUndecryptableCount
        endif
    endif
```

12.9.2.7 Per-MPDU Rx pseudocode for an MMPDU

```
if ((dot11RSNAActivated = true) and (frame is a robust Management frame)) then
    if ((dot11RSNAProtectedManagementFramesActivated = false) then
        if (Protected Frame subfield of the Frame Control field is equal to 1) then
            Discard the frame
        else
            Receive the MMPDU
        endif
    else // dot11RSNAProtectedManagementFramesActivated = true
        if (dot11RSNAUnprotectedManagementFramesAllowed = true) then
            if (STA with frame TA advertised MFPC = 0) then
                if (Protected Frame subfield of the Frame Control field is equal to 1) then
                    Discard the frame
                else
                    Make frame available for further processing
                endif
            else // STA with frame TA advertised MFPC = 1
                if (MMPDU has an individual RA) then
                    if (Pairwise key does not exist) then
```

```
        if (frame is a Disassociation or Deauthentication) then
            if (Protected Frame subfield of the Frame Control field is equal to 0) then
                Make the MPDU available for further processing
            else // encrypted
                Discard the frame
            endif
        else // frame is not a Disassociation or Deauthenticate
            Discard the frame
        endif
    else if (security association has an AES-CCM key) then
        if (Protected Frame subfield of the Frame Control field is equal to 0) then
            //unprotected frame
            Discard the frame
        else // frame is encrypted
            if (PN is not sequential) then
                Discard the MPDU as a replay
                Increment dot11RSNAStatsCCMPReplays
            else
                Decrypt frame using AES-CCM key
                if (the integrity check fails) then
                    Discard the frame
                    Increment dot11RSNAStatsCCMPDecryptErrors
                else
                    Make the MPDU available for further processing
                endif
            endif
        endif
    else if (security association has AES-GCM key) then
        if (Protected Frame subfield of the Frame Control field is equal to 0) then
            //unprotected frame
            Discard the frame
        else //frame is encrypted
            if (PN is not sequential) then
                Discard the MPDU as a replay
                Increment dot11RSNAStatsGCMPReplays
            else
                Decrypt frame using AES-GCM key
                If (the integrity check fails) then
                    Discard the frame
                else
                    Make the MPDU available for further processing
                endif
            endif
```

```
                    endif
                else // key for some other cipher—for future expansion
                endif
            else // MMPDU has a group RA
                if (IGTK does not exist) then
                    if (Disassociation or Deauthentication) then
                        Make frame available for further processing
                    else
                        Discard the frame
                    endif
                else // IGTK exists
                    if (MME is not present) then
                        Discard the frame
                    else // MME is present
                        if (AES-128-CMAC IGTK) then
                            if (IPN is not valid) then
                                Discard the frame as a replay
                                Increment dot11RSNAStatsCMACReplay
                            else if (integrity check fails) then
                                Discard the frame
                                Increment dot11RSNAStatsCMACICVError
                            else
                                Make frame available for further processing
                            endif
                        else // some other kind of key—for the future
                        endif
                    endif
                endif
            endif
        endif
    else // dot11RSNAUnprotectedManagementFramesAllowed = false
        if (MMPDU has an individual RA) then
            if (peer STA advertised MFPC = 1) then
                if (Pairwise key exists for the MMPDU's RA) then
                    if (security association has an AES-CCM key) then
                        if (Protected Frame subfield of the Frame Control field is equal to 0) then
                            Discard the frame
                        else // frame is encrypted
                            if (PN is not sequential) then
                                Discard the MPDU as a replay
                                Increment dot11RSNAStatsCCMPReplays
                            else
                                Decrypt frame using AES-CCM key
```

```
                        if (the integrity check fails) then
                            Discard the frame
                            Increment dot11RSNAStatsCCMPDecryptErrors
                        else
                            Make the MPDU available for further processing
                        endif
                    endif
                endif
            else // key for some other cipher—for future expansion
            endif
        else if (Protected Frame subfield of the Frame Control field is set to 1) then
            Discard the frame
        else if (Deauthenticate || Disassociate) then
            Make frame available for processing
        else
            Discard the frame
        endif
    else // peer STA didn't advertise MFPC = 1
        Discard the frame
    endif
else // MMPDU has a group RA
    if (IGTK exists) then
        if (MME is not present) then
            Discard the frame
        else // MME is present
            if (AES-128-CMAC IGTK) then
                if (PN is not valid) then
                    Discard the frame as a replay
                    Increment dot11RSNAStatsCMACReplay
                else if (security association has an AES-128-CMAC IGTK) then
                    Discard the frame
                    Increment dot11RSNAStatsCMACICVError
                else
                    Make frame available for further processing
                endif
            else // some other kind of key—for the future
            endif
        endif
    else // IGTK does not exist
        if (Disassociation or Deauthentication) then
            Make frame available for further processing
        else
            Discard the frame
```

endif
endif
endif
endif
endif
else // (*dot11RSNAActivated* = false) **or** (not a robust Management frame)
Use 12.9.2.6 to receive the frame
endif

12.9.2.8 Per-MSDU/Per-A-MSDU Rx pseudocode

if *dot11RSNAActivated* = true **then**
if the frame was not protected **then**
Receive the MSDU or A-MSDU unprotected
Make MSDU(s) available to higher layers
else if Address 1 has an individual RA **then** // Have a protected MSDU or A-MSDU
if Pairwise key is an AES-CCM key **then**
Accept the MSDU or A-MSDU if its MPDUs had sequential PNs (or if it consists of only one MPDU), otherwise discard the MSDU or A-MSDU as a replay attack and increment dot11RSNAStatsCCMPReplays
Make MSDU(s) available to higher layers
else if Pairwise key is an AES-GCM key **then**
Accept the MSDU or A-MSDU if its MPDUs had sequential PNs (or if it consists of only one MPDU), otherwise discard the MSDU or A-MSDU as a replay attack and increment dot11RSNAStatsGCMPReplays
else if Pairwise key is a TKIP key **then**
Compute the MIC using the michael algorithm
Compare the received MIC to the computed MIC
discard the frame if the MIC fails increment dot11RSNAStatsTKIPLocalMIC-Failures and invoke countermeasures if appropriate
compare TSC to replay counter, if replay check fails increment dot11RSNA-StatsTKIPReplays
otherwise accept the MSDU
Make MSDU available to higher layers
else if dot11WEPKeyMappings has a WEP key **then**
Accept the MSDU since the decryption took place at the MPDU
Make MSDU available to higher layers
endif
else // Have a group addressed MSDU or A-MSDU
if GTK for the Key ID does not exist **then**
discard the frame body and increment dot11WEPUndecryptableCount
else if GTK for the Key ID is null **then**
discard the frame body and increment dot11WEPUndecryptableCount
else if GTK for the Key ID is a CCM key **then**
Accept the MSDU or A-MSDU if its MPDUs had sequential PNs (or if it consists of only one MPDU), otherwise discard the MSDU or A-MSDU as a replay attack and increment dot11RSNAStatsCCMPReplays
Make MSDU(s) available to higher layers
else if GTK for the Key ID is a GCM key **then**
Accept the MSDU or A-MSDU if its MPDUs have sequential PNs (or if it consists of only one MPDU), otherwise discard the MSDU or A-MSDU as a replay attack and increment dot11RSNAStatsGCMPReplays

else if GTK for the Key ID is a TKIP key **then**
Compute the MIC using the michael algorithm
Compare the received MIC to the computed MIC
discard the frame if the MIC fails increment
dot11RSNAStatsTKIPLocalMICFailures and invoke countermeasures if appropriate
compare TSC to replay counter, if replay check fails increment
dot11RSNAStatsTKIPReplays
otherwise accept the MSDU
Make MSDU available to higher layers
else if GTK for the Key ID is a WEP key **then**
Accept the MSDU since the decryption took place at the MPDU
Make MSDU available to higher layers
endif
endif
endif

12.9.2.9 Per-MMPDU Rx pseudocode

if (*dot11RSNAActivated* = true) **then**
if (dot11RSNAProtectedManagementFramesActivated = true) **then**
if (the MPDU was not protected) **then**
Receive the MMPDU unprotected
Make the MMPDU available to higher layers
else //Have a protected MMPDU
if ((MMPDU has individual RA) **and** (security association has an AES-CCM key)) **then**
if (the MPDU has only one MPDU or multiple MPDUs with sequential PNs) **then**
Receive the MMPDU protected
Make the MMPDU available to higher layers
else
Discard the MMPDU as a replay
Increment *dot11RSNAStatsRobustMgmtCCMPReplays*
endif
else if ((MMPDU has individual RA) and (security association has an AES-GCM key)) **then**
if (the MPDU has only one MPDU or multiple MPDUs with sequential PNs) **then**
Receive the MMPDU protected
Make the MMPDU available to higher layers
else
Discard the MMPDU as a replay
Increment dot11RSNAStatsRobustMgmtGCMPReplays
else if ((MPDU has group addressed RA) **and** (security association has an AES-128-CMAC IGTK)) **then**
Receive the MMPDU
Make the MMPDU available to higher layers

```
                else
                    if (any other cipher exists) then
                        Process the frame using other cipher
                    else
                        Discard the frame
                    endif
                endif
            endif
        endif
    endif
```

12.10 Authenticated mesh peering exchange (AMPE)

The authenticated mesh peering exchange is defined in 14.5.

12.11 AP PeerKey support

12.11.1 AP PeerKey overview

The AP PeerKey protocol provides session identification and creation of an AP PeerKey association to provide for security of OBSS management communication between two APs. The result of a successful run of the AP PeerKey protocol is an AP PeerKey association. An AP PeerKey association is composed of a mesh PMKSA and a mesh TKSA.

Two APs perform the AP PeerKey protocol in order to protect HCCA TXOP Advertisement frames in an OBSS. The AP PeerKey protocol is unauthenticated (neither peer has a verified identity of the other peer) but an AP knows that only the peer AP that completed the AP PeerKey protocol is able to send protected HCCA TXOP Advertisement frames protected by the resulting AP PeerKey association. This allows an AP to determine whether a peer AP honors the HCCA TXOP avoidance schedule that is negotiated. In this manner, an AP is able to honor the negotiated schedule of trusted peer APs and ignore peer APs that are not trustworthy. This allows trustworthy APs to negotiate mutually beneficial schedules while allowing an AP to not disadvantage itself in an OBSS in the presence of untrustworthy APs.

AP PeerKey uses various functions and data to accomplish its task and assumes certain properties about each function 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, pseudorandom key. This function takes as input a non-secret “salt” and a secret input and produces a fixed-length output.

— A finite cyclic group is negotiated for which solving the discrete logarithm problem is computationally infeasible.

When using AKM suite selector 00-0F-AC:10 to indicate AP PeerKey, H shall be instantiated as HMAC-SHA-256, taking as input a non-secret “salt” and a secret input keying material “ikm”:

H(salt, ikm) = HMAC-SHA-256(salt, ikm)

Other instantiations of function H require creation of a new AKM identifier.

12.11.2 AP PeerKey protocol

AP PeerKey uses the same discrete logarithm cryptography as SAE (as described in 12.4) to achieve key agreement. Each party to the exchange has a public and private key with respect to a particular set of domain parameters that define a finite cyclic group. Groups may be based on elliptic curve cryptography (ECC) or finite field cryptography (FFC). Each component of a group is referred to as an *element*. Groups are negotiated using an identifying number from a repository maintained by IANA as "Group Description" attributes for IETF RFC 2409 (IKE) [B17][B31]. The repository maps an identifying number to a complete set of domain parameters for the particular group. For the purpose of interoperability, APs that have dot11ProtectedHCCATXOPNegotiationImplemented true or dot11ProtectedQLoadReportImplemented true shall support group 19, an ECC group defined over a 256-bit prime order field.

AP PeerKey uses one arithmetic operator that takes one element and one scalar value to produce another element (called the *scalar operation*). The convention used here is to represent group elements in uppercase bold italic and scalar values in lowercase italic. The scalar operation takes an element and a scalar and is denoted scalar-op(x,$\boldsymbol{Y}$).

The private key d shall be chosen randomly so that $1 < d < r$, where r is the order of the group. The public key $\boldsymbol{Q}$ shall be produced using Equation (12-1).

$$\boldsymbol{Q} = \text{scalar-op}(d,\boldsymbol{G}) \tag{12-1}$$

where

$\boldsymbol{G}$ is the generator (also known as the base point) of the group

An AP for which dot11ProtectedTXOPNegotiationActivated is true or dot11ProtectedQLoadReportActivated is true shall support at least one public key from cyclic group nineteen and may support multiple public keys from multiple cyclic groups. An AP that supports the Multiple BSSID capability and has dot11ProtectedTXOPNegotiationActivated true or dot11ProtectedQLoadReportActivated true may use one public key across multiple BSSIDs, or it may choose to generate a public key for each supported BSSID.

The AP PeerKey protocol consists of an exchange of public keys from an AP and a peer AP. An AP requests the public key of a peer AP by sending a Public Key frame with the Public Key Frame Usage field set to "Request." This frame contains the public key of the initiating AP. The initiating AP awaits a response to its request. The peer AP responds to the "Request" from the initiating AP with a Public Key frame with the Public Key Frame Usage field set to "Response", indicating an acceptable group was sent by the initiating AP, or set to "NAK" indicating that the group sent by the initiating AP was unacceptable.

It is possible for multiple runs of the AP PeerKey protocol to be performed by an AP but there shall be only one instance at a time of the AP PeerKey protocol to a peer AP for a particular group. The state associated with each run of the AP PeerKey protocol is the group and the two public keys of the participating APs.

An AP for which dot11ProtectedTXOPNegotiationActivated is true or dot11ProtectedQLoadReportActivated is true shall reply to a Public Key frame. An AP that has both dot11ProtectedTXOPNegotiationActivated is false and dot11ProtectedQLoadReportActivated is false shall drop all received Public Key frames. If the responding AP refuses to perform the AP PeerKey protocol with the initiating AP it shall respond with a Public Key frame, setting the Public Key Frame Usage field to "Refused". Otherwise, if the Group field in the public key request is a group that is supported by the responding AP, the AP shall reply with a public key of the same group as the request, generating such a key pair if required, and setting the Public Key Frame Usage field to "Response". The receiving AP shall

generate a PMK and a mesh PMKSA, see below, and terminate the PeerKey protocol. If the group field in the public key request is not supported by the responding AP, the responding AP shall respond with a Public Key frame, setting the Public Key Frame Usage field to "NAK" and the group set to a group acceptable by the receiving AP.

If the initiating AP does not receive a response to its request after five seconds, it should retransmit its request. The initiating AP should attempt such retransmission a minimum of five times.

If the initiating AP receives a Public Key frame from a peer AP with the Public Key Frame Usage field set to "NAK", the initiating AP inspects the group field in the received "NAK". If the group is supported, the initiating AP should initiate to the peer AP using the indicated group. If the group is not supported, the AP may choose to initiate to the peer AP using a different group from the dot11RSNAConfigDLCGroup table. Receipt of a Public Key frame from a peer AP with the Public Key Frame Usage field set to "NAK" terminates the PeerKey protocol.

If the initiating AP receives a Public Key frame from a peer AP with the Public Key Frame Usage field set to "Refused", the initiating AP shall terminate the PeerKey protocol.

An initiating AP that receives a refusal should delay sending another request for at least 5 seconds, and should perform exponential back-off on all subsequent requests by doubling the delay with each subsequent refusal.

NOTE—This standard does not specify a limit on the number of attempts an initiating AP makes to communicate with another AP.

If the initiating AP receives a Public Key frame from a peer AP with the Public Key Frame Usage field set to "Response" and the group set to the value in the initiating AP's "Request", the initiating AP shall generate a PMK and a mesh PMKSA. This terminates the PeerKey protocol.

If the initiating AP receives a Public Key frame from a peer AP with the Public Key Frame Usage field set to "Request" prior to receiving a "Response", it checks the Group field in the request. If the Group in the received "Request" is the same as the Group the initiating AP sent it its request, the AP shall construct a Public Key frame with the Public Key Frame Usage field set to "Response" containing the public key it sent in its "Request" and transmit it to the peer. The AP shall then generate a PMK and a mesh PMKSA, see below, and terminate the PeerKey protocol. If the Group field differs and the group indicated is supported, the AP shall respond to the peer AP by replying with a public key from that group, generating a key pair if required, and setting the Public Key Frame Usage field to "Response". The AP shall generate a PMK and a mesh PMKSA, see below, and terminate this run of the PeerKey protocol.

Once the AP and peer AP have exchanged public keys from the same finite cyclic group they can compute the Diffie-Hellman shared secret using scalar-op() and function F from 12.4.4:

$$k = \text{F}(\text{scalor-op}(d, \boldsymbol{Q}_p)) \tag{12-2}$$

where

d is the private key of the AP that is calculating k
$\boldsymbol{Q}_p$ is the public key of the peer AP

Entropy of the shared secret shall then be extracted using function H to produce *keyseed* using Equation (12-3).

$$keyseed = \text{H}(<0>32, k) \tag{12-3}$$

where

<0>32 denotes 32 octets of the value zero

The PMK shall be derived according to Equation (12-4), and the PMKID shall be derived according to Equation (12-5).

PMK = KDF-Hash-256(*keyseed*, "AP Peerkey Protocol",
0x00 || Max(LOCAL-MAC, PEER-MAC) || Min(LOCAL-MAC, PEER-MAC)) (12-4)

PMKID = Truncate-128(SHA-256(Q_1 || Q_2 ||
Max(LOCAL-MAC, PEER-MAC) ||
Min(LOCAL-MAC, PEER-MAC)) (12-5)

where

KDF-Hash-256	is the key derivation function defined in 12.7.1.7.2 using the hash algorithm identified by the AKM suite selector (see Table 9-133)
0x00	is a single octet with a value of zero
LOCAL-MAC	is the AP's MAC address
PEER-MAC	is the peer AP's MAC address
Q_1	is the public key used by AP1 in the AP PeerKey protocol encoded as an octet stream using the Element to Octet string conversion from 12.4.7.2.3.
Q_2	is the public key used by AP2 in the AP PeerKey protocol encoded as an octet stream using the Element to Octet string conversion from 12.4.7.2.3.
AP1	is the AP whose MAC address equals Max(LOCAL-MAC, PEER-MAC)
AP2	is the AP whose MAC address equals Min(LOCAL-MAC, PEER-MAC)

The Max and Min operations for IEEE 802 addresses are with the address converted to a positive integer treating the first octet as the most significant octet of the integer.

keyseed shall be irretrievably deleted after the PMK is generated.

The lifetime of the mesh PMKSA shall be set to the value dot11RSNAConfigPMKLifetime.

Upon creation of the PMK, an AEK shall be created per 14.5.7. The mesh PMKSA for this instance of the AP PeerKey protocol shall then be created using the AP's MAC address as the STA's MAC address, the peer AP's MAC address as the peer STA's MAC address, the AEK, the lifetime, and the PMKID. If a mesh PMKSA created by a prior run of the AP PeerKey protocol exists it shall be deleted upon creation of the new PMKSA. All mesh TKSAs created by the old mesh PMKSA shall also be deleted.

Upon creation of the mesh PMKSA, the APME protocol (as defined in 14.5 shall be used to prove possession of the PMK (and implicitly the private key that corresponds to the peer's public key) and generate the mesh TKSA.

Note that it is possible for two APs which simultaneously initiated to each other with different, but acceptable, groups to end up with two mesh PMKSAs. In this unlikely case, the mesh PMKSA from a group with the largest prime in its domain parameter set shall be used with the AMPE protocol. The other mesh PMKSA shall be deleted.

If the AMPE protocol completes successfully, Protected HCCA TXOP Advertisement frames and Protected HCCA TXOP Response frames may be used in the HCCA TXOP negotiation procedures, as defined in 11.28.3 using the MTK from the mesh TKSA. If the AMPE procedure completes successfully, Protected QLoad Request frames and Protected QLoad Report frames may be used in the QLoad report procedures, as defined in 11.28.2 using the MTK from the mesh TKSA. If the AMPE protocol fails, the peer's public key, PMK, and PMKSA shall be deleted.

NOTE—The PMK, as well as any key derived from it, is not authenticated in any way nor is it bound to any identity. This protocol protects an AP that cooperates in scheduling its HCCA TXOPs using Protected HCCA TXOP Advertisement frames because no other entity knows its private key and cannot forge Protected HCCA TXOP Advertisement frames. An AP that receives a Protected HCCA TXOP Advertisement frame is assured it was sent by the holder of a particular private key, and no one else, and can, therefore, establish which APs are cooperating in their HCCA TXOP scheduling. An AP that receives a Protected QLoad Report frame is assured it was sent by the holder of a particular private key, and no other STA, and can, therefore, establish which APs are correctly reporting their QoS load.