IEEE P802.11  
Wireless LANs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Draft Text for PQC Protocols | | | | |
| Date: 2025-09-11 | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | HPE | San Jose, California,  United States of America |  |  |
| Michael Montemurro | Huawei | Toronto, Canada |  | montemurro.michael@gmail.com |
| Arik Klein | Huawei TRC |  |  | [arik.klein@huawei.com](mailto:arik.klein@huawei.com) |
| Stephen McCann | Huawei | Southampton, UK |  | [stephen.mccann@ieee.org](mailto:stephen.mccann@ieee.org) |

Abstract

This document proposes draft text to define PQC protocols for the PQC TG.

*Instruct the editor to modify section 2 as indicated:*

**2. Normative References**

FIPS 186-4, Digital Signature Standard.

FIPS 197, Advanced Encryption Standard (AES).

FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard

FIPS 204, Module-Lattice-Based Digital Signature Standard

IETF RFC 2104, HMAC: Keyed-Hashing for Message Authentication, Krawczyk, H., M. Bellare, andR. Canetti, Feb. 1997 (status: informational).

IETF RFC 2131, Dynamic Host Configuration Protocol, Mar. 1997.

IETF RFC 2315, PKCS#7: Cryptographic Message Syntax Version 1.5, Kaliski, B., Mar. 1998.

*Instruct the editor to modify section 6.5.5 as indicated:*

* Authenticate
* MLME-AUTHENTICATE.request
* Semantics of the service primitive

***Modify MLME-AUTHENTICATE.request and the table as follows (not all lines shown):***

The primitive parameters are as follows:

MLME-AUTHENTICATE.request(

...,

Content of Authentication frame fragment,

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| ..... | ..... | ..... | ..... |
| AuthenticationType | Enumeration | OPEN\_SYSTEM, SHARED\_KEY, FAST\_BSS\_TRANSITION, SAE, FILS\_SHARED\_KEY WITHOUT\_PFS, FILS\_SHARED KEY\_WITH\_PFS, FILS\_PUBLIC\_KEY, PQC\_NOSIG, PQC\_SIG, PQC\_PAKE, PQC\_UNAUTH | Specifies the type of authentication algorithm to use during the authentication process. |
| ..... | ..... | ...... | ....... |
| Content of Authentication frame fragment. | Octet String | A fragment of an Authentication frame. | An octet string containg a fragment of authentication frame for an authentication algorithm and authentication transaction sequence number where the Fragmentation column in Table 9-71 is Yes. |
| VendorSpecificInfo | A set of elements | As defined in 9.4.2.24 (Vendor Specific element) | Zero or more elements. |

* MLME-AUTHENTICATE.confirm
* Semantics of the service primitive

***Modify MLME-AUTHENTICATE.confirm and the table as follows (not all lines shown):***

The primitive parameters are as follows:

MLME-AUTHENTICATE.confirm(

....

Content of 802.1X Authentication frame,

Content of EPPKE Authentication frame

Content of Authentication frame fragment,

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| ..... | ..... | ..... | ..... |
| AuthenticationType | Enumeration | OPEN\_SYSTEM, SHARED\_KEY FAST\_BSS\_TRANSITION, SAE, FILS\_SHARED KEY\_WITHOUT\_PFS, FILS\_SHARED\_KEY\_WITH\_PFS, FILS\_PUBLIC\_KEY, PASN, PQC\_NOSIG, PQC\_SIG, PQC\_PAKE, PQC\_UNAUTH | Specifies the type of authentication algorithm that was used during the authentication process. This value matches the AuthenticationType parameter specified in the corresponding MLME-AUTHENTICATE.request primitive. |
| ..... | ..... | ..... | ..... |
| Content of Authentication frame fragment. | Octet String | A fragment of an Authentication frame. | An octet string containg a fragment of authentication frame for an authentication algorithm and authentication transaction sequence number where the Fragmentation column in Table 9-71 is Yes. |
| VendorSpecificInfo | A set of elements | As defined in 9.4.2.24 (Vendor Specific element) | Zero or more elements. |

* MLME-AUTHENTICATE.indication
* Semantics of the service primitive

***Modify MLME-AUTHENTICATE.indication and the table as follows (not all lines shown):***

The primitive parameters are as follows:

MLME-AUTHENTICATE.indication(

....

Content of Authentication frame fragment,

VendorSpecificInfo

)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| ..... | ..... | ..... | ..... |
| AuthenticationType | Enumeration | OPEN\_SYSTEM, SHARED\_KEY, FAST\_BSS\_ TRANSITION, SAE, FILS\_SHARED\_KEY\_WITHOUT\_PFS, FILS\_SHARED\_KEY\_WITH\_PFS, FILS\_PUBLIC\_KEY, PASN, PQC\_NOSIG, PQC\_SIG, PQC\_PAKE, PQC\_UNAUTH | Specifies the type of authentication algorithm that was used during the authentication process. |
| ..... | ..... | ..... | ..... |
| Content of Authentication frame fragment. | Octet String | A fragment of an Authentication frame. | An octet string containg a fragment of authentication frame for an authentication algorithm and authentication transaction sequence number where the Fragmentation column in Table 9-71 is Yes. |
| VendorSpecificInfo | A set of elements | As defined in 9.4.2.24 (Vendor Specific element) | Zero or more elements. |

* MLME-AUTHENTICATE.response
* Semantics of the service primitive

***Modify MLME-AUTHENTICATE.response and the table as follows (not all lines shown):***

The primitive parameters are as follows:

MLME-AUTHENTICATE.response(

....

Content of Authentication frame fragment,

VendorSpecificInfo  
)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| ..... | ..... | ..... | ..... |
| Content of Authentication frame fragment. | Octet String | A fragment of an Authentication frame. | An octet string containg a fragment of authentication frame for an authentication algorithm and authentication transaction sequence number where the Fragmentation column in Table 9-71 is Yes. |
| VendorSpecificInfo | A set of elements | As defined in 9.4.2.24 (Vendor Specific element) | Zero or more elements. |

*Instruct the editor to modify Tables 9-70 and 9-71 in 9.3.3.11 as indicated and renumber the* Order *column as necessary to accommodate the additional fields :*

**Table 9-70—Authentication frame body**

|  |  |  |
| --- | --- | --- |
| Order | Information | Notes |
| 3 | Status Code | The status code information is reserved in certain Authentication  frames as defined in Table 9-71. |
| 4 | MMPDU Fragmentation Information | The MMPDU Fragment Information field including the MMPDU Fragment Number field, the More MMPDU Fragments field, and a Requested MMPDU fragment field that is conditionally present. |
| 27 | Tunneled PASN | A Tunneled PASN element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| <ANA2> | PQC Key | A PQC Key element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| <ANA2>+1 | PQC Commit | A PQC Commit element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| <ANA2>+2 | PQC Ciphertext | A PQC Ciphertext element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| <ANA2>+3 | PQC Key Selector | A PQC Key Selector element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| <ANA2>+4 | PQC Signature | A PQC Signature element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |
| Last-1 | Vendor Specific | One or more Vendor Specific elements are optionally present |
| Last | MIC | A MIC element is present only in certain Authentication frames as defined in Table 9-71 (Presence of fields and elements in Authentication frames). |

*Instruct the editor to modify 9-71 by including a new column to the right with the title “Fragmentation”. Mark all rows of the column “No” with the exception of the following entries*

**Table 9-71—Presence of fields and elements in Authentication frames**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Authentication algorithm** | **Authentication transaction sequence number** | **Status code** | **Presence of fields and elements indicated as conditional in Table 9-70 (Authentication frame body)** | **Fragmentation** |
| PASN Authentication | 3 | Status | PASN Parameters element is present if Status Code  field is 0.  Wrapped data element is present if wrapped data  format in PASN Parameters element is nonzero and not  reserved; and Status Code field is 0.  MIC element is present.  Fragment element may be present if any of the  elements are fragmented and Status Code field is 0. |  |
| PQC No Signature Authentication | 1 | Any | The RSNE is present. The PQC Ciphertext element is present. The PQC Key Selector element is present. | Yes |
| PQC No Signature Authentication | 2 | Any | The RSNE is present. The PQC Ciphertext element is present if the Status Code field is zero. | Yes |
| PQC Digital Signature | 1 | Any | The RSNE is present. The PQC Key element is present. | Yes |
| PQC Digital Signature | 2 | Any | The RSNE is present. The PQC Ciphertext element is present. The Session element is present. | Yes |
| PQC Digital Signature | 3 | Any | The Public Key element is present. | Yes |
| PQC Digital Signature | 4 | Any | The Public Key element is present if the status is zero. | Yes |
| PQC Digital Signature | 5 | Any | The PQC Signature element is present. The MIC element is present. | Yes |
| PQC Digital Signature | 6 | Any | The PQC Signature element is present if the Status code is zero. The MIC element is present if the Status code is zero. | Yes |
| PQC PAKE | 1 | Any | The RSNE is present. The Password Identifier element is present. The PQC Commit element is present. | Yes |
| PQC PAKE | 2 | Any | The RSNE is present. The PQC Ciphertext element is present. The MIC element is present. | Yes |
| PQC PAKE | 3 | Any | The MIC element is present. | Yes |
| Opportunistic ML-KEM | 1 | Any | The RSNE is present. The PQC Key element is present. | Yes |
| Opportunistic ML-KEM | 2 | Any | The RSNE is present if the Status code is zero. The PQC Ciphertext element is present if the Status code is zero. | Yes |
| PQC PMK Caching | 1 | Any | The RSNE is present. The PQC Key element is present. | Yes |
| PQC PMK Caching | 2 | Any | The RSNE is present if the Status code is zero. The Ciphertext element is present if the Status code is zero. | Yes |

*Instruct editor to modify section 9.4.1.1 as indicated:*

**9.4.1.1 Authentication Algorithm Number field**

The Authentication Algorithm Number field indicates a single authentication algorithm. (M118)(#4170)The

Authentication Algorithm Number field is shown in Figure 9-137 (Authentication Algorithm Number field

format). The following values are defined for authentication algorithm number:

Authentication algorithm number = 0: Open System

Authentication algorithm number = 2: Fast BSS Transition

Authentication algorithm number = 3: Simultaneous Authentication of Equals (SAE)

Authentication algorithm number = 4: FILS Shared Key authentication without PFS

Authentication algorithm number = 5: FILS Shared Key authentication with PFS

Authentication algorithm number = 6: FILS Public Key authentication

Authentication algorithm number = 7: PASN authentication

Authentication algorithm number = <ANA3>: PQC signature authentication

Authentication algorithm number = <ANA4>; PQC non-signature authentication

Authentication algoirithm number = <ANA5>: PQC PAKE authentication

Authentication algorithm number = <ANA6>: PQC unauthenticated

Authentication algorithm number = <ANA7>: PQC PMK Caching

Authentication algorithm number = 65 535: vendor specific use

*Instruct the editor to modify Table 9-130 in section 9.4.2.1 as indicated:*

**Table 9-130—Element IDs**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element** | **Element ID** | **Element ID Extension** | **Extensible** | **Fragmentable** |
| Tunneled PASN (see 9.4.2.315 (Tunneled PASN element)) | 255 | 143 | Yes | No |
| PQC Key Selector | 255 | <ANA8> | Yes | Yes |
| PQC Key | 255 | <ANA9> | Yes | Yes |
| PQC Commit | 255 | <ANA10> | Yes | Yes |
| PQC Ciphertext | 255 | <ANA11> | Yes | Yes |
| PQC Signature | 255 | <ANA12> | Yes | Yes |
| Reserved | 255 | <ANA12>+1-255 |  |  |

*Instruct the editor to insert the following subclause at the end of 9.4.1*

**9.4.1.71a MMPDU Fragmentation Information field**

The MMPDU Fragmentation Information field is used with authentication protocols as specified in 12.4 which produce frames that exceed the maximum transmission unit. See Figure 9-181.

The MMPDU Fragment Number field indicates the fragment number of the fragmented MMPDU.

The More MMPDU Fragments field indicates that there are more MMPDU fragments which are part of the Authentication frame to be received.

The Requested MMPDU Fragment field indicates a request for an MMPDU fragment carried in an Authentication frame to be retransmitted.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **B0 B3** | **B4** | **B5** | **B6 B7** |
|  | MMPDU Fragment Number | More  MMPDU Fragments | Requested MMPDU Fragment | Reserved |
| Bits | 4 | 1 | 1 | 2 |

**Figure 9-207x—Fragmentation Information field format**

*Instruct the editor to modify Table 9-190 in section 9.4.2.23.3 as indicated:*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **OUI** | **Suite Type** | **Authentication Type** | **Key management type** | **Key derivation type** | **Authentication algorithm numbers (see 9.4.1.1 (Authentication Algorithm Number field))** | **Cipher suite selector restriction** |
| 00-0F-AC | 25 | FT authentication over SAE | FT key management defined in 12.7.1.6 (FT key hierarchy) | Defined in 12.7.1.6.2 (Key derivation function (KDF)) using the hash algorithm specified in 12.4.2 (Assumptions on SAE) | 3 (SAE) for FT Initial Mobility Domain Association 2 (FT) for FT protocol reassociation as defined in 13.5 (FT protocol)  0 (open) for FT Initial Mobility Domain Association over PMKSA caching | none |
| 00-0F-AC | <ANA13> | PQC Auth With No Signature | PQC Mutual Auth defined in 12.7.1.3 (Pairwise Key Heirachy) | Defined in 12.7.1.6.2 (Key Derivation function (KDF)) using HKDF | <ANA4> for PQC mutual authentication  without signatures | None |
| 00-0F-AC | <ANA14> | PQC Digital Signature | PQC Mutual Auth defined in 12.7.1.3 (Pairwise Key Hierarchy) | Defined in 12.7.1.6.2 (Key Derivation (KDF)) using HKDF | <ANA3> for PQC mutual authentication using ML-DSA | None |
| 00-0F-AC | <ANA15> | PQC PAKE | PQC Mutual Auth defined in 12.7.1.3 (Pairwise Key Hierarchy) | Defined in 12.7.1.6.2 (Key Derivation (KDF)) using HKDF | <ANA5> for PQC mutual authentication using a password | None |
| 00-0F-AC | <ANA16> | Opportunistic ML-KEM | None | Defined in 12.7.1.6.2 (Key Derivation (KDF) using HKDF) | <ANA6> for PQC Unauthenticated | None |
| 00-0F-AC | <ANA17> | Authentication negotiated over IEEE Std 802.1X using a CNSA 2.0-compliant EAP method | RSNA Key management as defined in 12.7 | Defined in 12.7.1.6.2 (Key Derivation (KDF)) using HKDF | 0 (Open) | Used only  with cipher  suite selector  values 00-0FAC:  9 (GCMP-  256), 00-0FAC:  10 (CCMP-256),  00-0F-AC:13  (BIP-CMAC-  256), and 00-  0F-AC:12  (BIP-GMAC-256 |
| 00-0F-AC | <ANA17>+1-255 | Reserved | Reserved | Reserved | Reserved | Reserved |
| Other OUI or CID | Any | Vendor-specific | Vendor-Specific | Vendor-Specific | Vendor-Specific | Vendor-Specific |

*Instruct the editor to modify the following subclause to add a status code to Table 9-80*

**9.4.1.9 Status Codes**

Table 9-80—Status codes

|  |  |  |
| --- | --- | --- |
| Status Code | Name | Meaning |
| 0 | SUCCESS | Successful. |
| 1 | REFUSED\_REASON\_UNSPECIFIED | Unspecified failure. |
|  | … |  |
| 143 | GAS\_QUERY\_REQUEST\_  TOO\_ LARGE | GAS query request is larger than the  dot11GASQueryRequestLengthLimit value. |
| <ANA> | MMPDU\_FRAGMENT\_NOT\_AVAILABLE | The Authentication frame with MMPDU fragment (as derived by the SME) is not available for retransmission. |
| 144-65 535 |  | Reserved |

*Instruct the editor to modify section 9.4.2.178 as indicated:*

**9.4.2.178 Session element**

The Session element is used to convey the (unique) identifier of an in-progress authentication protocol session. The format of the Session element is shown in Figure 9-749 ( Session element format).

|  |  |  |  |
| --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Session |

Octets 1 1 1 variable

**Figure 9-749— Session element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1. For FILS, the Session field is 8 octets and chosen randomly by the non-AP STA. For PQC protocols, the Session field is a variable length.

*Instruct editor to modify section 9.4.2.179 as indicated*

**9.4.2.179 Public Key element**

The Public Key element is used to communicate the device’s (certified) public key for use with signature-based authentication exchanges. The format of the Public Key element is shown in Figure 9-750 (FILS Public Key element format).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Key Type | Public Key |

Octets: 1 1 1 1 variable

Figure 9-750—Public Key element format

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1.

The Key Type field values are as follows:

0: Reserved.

1: Public Key field contains an X.509v3 certificate encoded according to IETF RFC 5280.

2: Public Key field contains an uncertified public key encoded according to IETF RFC 5480.

3: Public Key field contains an uncertified public key encoded according to IETF RFC 3279.

4: Public Key field contains a PKCS#7 envelope of certificates encoded according to IETF RFC 2315.

5: Public Key field contains a DER-encoded X.509v3 certificate per draft-ietf-lamps-dilithium-certificates (soon to be an RFC)

6: Public Key field contains an uncertified public key encoded according to section 4 of draft-ietf-lamps-dilithium-certificates

7–255: Reserved.

*Instruct editor to modify section 9.4.2.188 as indicated:*

**9.4.2.188 Nonce element**

The Nonce element is used for exchanging an additional source of randomness in authentication exchanges. The format of the Nonce element is shown in Figure 9-769 (Nonce element format).

|  |  |  |  |
| --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Nonce |

Octets 1 1 1 16

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1. The Nonce field contains randomly generated data.

*Instruct the editor to create new sections 9.4.2.2.V, 9.4.2.W, 9.4.2.X, 9.4.2.Y, and 9.4.2.Z including new figures, as indicated:*

**9.4.2.V PQC Key Selector element**

The PQC Key Selector element contains a hash of a public key used in a PQC algorithm. The algorithm definition determines the hash function used to compute the hash. See figure 9-ABCD (PQC Key Selector element format).

|  |  |  |  |
| --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Key Selector |

Octets: 1 1 1 variable

**Figure 9-ABCD—Key Selector element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The Key Selector contains a hash of a PQC Key. The definition of the PQC algorithm determines the hash function used to compute the hash and therefore the length of the Key Selector field.

**9.4.2.W PQC Key element**

The PQC Key element contains a public key used in a PQC key exchange algorithm. See Figure 9-LMNO (PQC Key element format). This element contains public keys for key establishment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | KEM Parameter Set | Length of Public Key | KEM Key |

Octets: 1 1 1 1 2 variable

**Figure 9-LMNO—PQC Key element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The KEM Parameter Set is an 8-bit unsigned integer that maps an identifying number to a parameter set defining the algorithm. The following values are defined for KEM Parameter Set:

KEM Parameter Set = 0: Reserved

KEM Parameter Set = 1: ML-KEM-512

KEM Parameter Set = 2: ML-KEM-768

KEM Parameter Set = 3: ML-KEM-1024

KEM Parameter Set 4-254: Reserved

KEM Parameter Set 255: Vendor Specific

The Length of Public Key indicates the length in octets of the public key that follows. Some KEM public keys can be too large to fit in a single element and in many cases too large to fit in a single frame. Therefore, these elements will necessarily require fragmentation and reassembly.

The KEM key field is a public key from the indicated KEM Parameter Set whose length depends on the parameter set.

**9.4.2.X PQC Commit**

The PQC Commit element contains the information a STA sends to an AP when committing to a guess of a password in the PQC PAKE exchange. See Figure 9-PQRS (PQC Commit).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | KEM Parameter Set | Random Commit | Encoded public key |

Octets: 1 1 1 1 96 variable

**Figure 9-PQRS—PQC Commit element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The KEM Parameter set indicates the parameter set of the encoded public key. The Random Commit is always 96 octets and the length of the encoded public key depends on the KEM Parameter Set of the encoded public key (see Table 12.Y in section 12.X.4).

**9.4.2.Y PQC Ciphertext element**

The PQC Ciphertext element contains an encapsulated secret encrypted using the public key from a PQC key establishment algorithm. See Figure 9-WXYZ (Ciphertext element).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Length of Ciphertext | Ciphertext |

Octets: 1 1 1 2 variable

**Figure 9-WXYZ—Ciphertext element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The Length of Ciphertext indicates the length in octets of the ciphertext that follows. PQC ciphertexts are too large to fit in a single element and in some cases too large to fit in a single frame. Therefore, these elements will necessarily require fragmentation and reassembly (see 10.28.11 (Element fragmentation) and 10.28.12 (Element defragmentation)) and possibly encompass multiple frames.

The Ciphertext field contains a secret encrypted using a public key from a PQC key establishment algorithm. Its length depends on the KEM Parameter Set of the public key used for encapsulation.

**9.4.2.Z PQC Signature element**

The PQC Signature element contains a digital signature from a PQC digital signature algorithm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | DSA Parameter Set | Length of Signature | Signature |

Octets: 1 1 1 1 2 variable

**Figure 9-TUVW—PQC Signature element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The DSA Parameter Set is an 8-bit unsigned integer that maps an identifying number to a parameter set defining the algorithm. The following values are defined for DSA Parameter Set:

DSA Parameter Set = 0: Reserved

DSA Parameter Set = 1: ML-DSA-44

DSA Parameter Set = 2: ML-DSA-65

DSA Parameter Set = 3: ML-DSA-87

DSA Parameter Set 4-254: Reserved

DSA Parameter Set 255: Vendor Specific

The Length of Signature indicates the length in octets of the signature that follows. PQC signatures are too large to fit in a single element and in many cases too large to fit in a single frame. Therefore, these elements will necessarily require fragmentation and reassembly.

The Signature field is a signature generated by the indicated DSA Parameter Set whose length depends on the parameter set.

***Instruct the editor to insert the following subclause to 11.3.4.2***

**11.3.4.2.a Fragmentation of Authentication frames by the SME – orginating STA**

The SME may fragment an MMPDU that is carried in Authentication frame with a specific Authentication algorithm number and Transaction Sequence Number as indicated with a Yes in the MMPDU Fragmentation column of Table 9-71.

The SME shall fragment the contents of the authentication frame body on octet boundaries. All fragments except the last shall be the same size.The Authentication frame shall include the MMPDU Fragmentation Information field as follows:

* The STA SME sets the MMPDU Fragment Number field to 0 for the first fragment and increments the field by 1 for each additional fragment.
* The STA SME sets the More MMPDU Fragments field to 1 for all fragments except for the last fragment, where the More MMPDU Fragments field is set to 0.
* The Requested MMPDU Fragment field is set to 0.

All Authentication frames for protocols listed as “Fragmentable” in Table 9-71 shall include the MMPDU Fragmentation Information field. If the length of the Authentication frame will be less than the maximum transmission unit the frame shall be treated as the “last fragment” and the More MMPDU Fragments field will be set to 0.

If the initiating STA SME receives an MLME-AUTHENTICATE.indication.indication from the MAC for a reception of an Authentication frame with the transaction sequence number which the initiating STA sent and that includes a MMPDU Fragmentation Information field where the Requested MMPDU Fragment field is set to 1, and the MMPDU fragment indicated in the MMPDU Fragment Number field is available, the STA SME may retransmit the requested MMPDU fragment in an Authentication frame as requested by the peer STA.

If the STA SME requests the retransmssion of the MMPDU fragment, it invokes an MLME-AUTHENTICATE.request to the MAC to retransmit the fragemented MMPDU in an Authentication frame. The Authentication frame shall include the Fragmentation Information field contains the following following fields:

* The Authentication Algorithm Number and Transaction Sequence Number fields set to the values in the received Authentication frame.
* The MMPDU Fragment Number field set to the value received in the Authentication frame requesting the retransmission.
* If the requested fragment is available, the STA sets the Status Code field to SUCCESS and includes the MMPDU fragment in the payload of the Authentication frame.
* If the requested fragment is not available, the STA sets the Status Code field to FRAGMENT\_NOT\_AVAILABLE.

*Instruct the editor to insert the following subclause to 11.3.4.3*

**11.3.4.3.a Fragmentation of Authentication frames by the SME – destination STA**

The STA SME may receive and reassemble a MMPDU fragment that is carried in an Authentication frame with a specific Authentication Algorithm Number and Transaction Sequence Number as indicated with a Yes in the MMPDU Fragmentation column of Table 9-71.

A STA that receives an Authentication frame with MMPDU Fragment Information field shall verify that the Authentication Algorithm Number and Transaction Sequence number values indicate that the Authentication frame may contain a MMPDU fragment. Otherwise, the STA shall discard the received MPDU.

If the Authentication frame with the MMPDU fragment is correctly received by the MAC, it shall invoke the MLME-MMPDU-FRAGMENT.indication to deliver the contents of the Authentication frame to the SME.

A STA SME that receives a MLME-AUTHENTICATE.indication shall process received MMPDU fragments using the MMPDU Fragmentation Information field in the Authentication frame by processing the Fragment Number and More Fragments fields to reassemble the frame body of the Authentication frame.

If the STA SME determines that it did not receive anMMPDU fragment, it may invoke an MLME-AUTHETNICATE.response primitive to request the MAC to request the peer STA to retransmit the missing MMPDU fragment. The MMPDU Fragmentation Information field in the Authentication frame is set as follows:

* The Authentication Algorithm Number and Transaction Sequence Number fields are set to the values from the missing fragment.
* The More MMPDU Fragments field is set to 0.
* The MMPDU Fragment Number field shall be set to the value of the requested MMPDU fragment.
* The Requested MMPDU Fragment field shall be set to 1 to indicate a request to retransmit the MMPDU fragment.

If the STA SME receives a MLME-AUTHENTICATE.indication that contains a retransmitted frame fragment,

it shall process received MMPDU fragment using the MMPDU Fragmentation Information based on the Status Code field received in the Authentication frame as follows:

* When the STA SME receives a Status Code field set to FRAGMENT\_NOT\_AVAILABLE, it shall abandon the instance of the authentication protocol
* If the Status Code field is set to SUCCESS and More Fragments field is set to 0, the STA SME shall process the MMPDU fragment according to the value of the MMPDU Fragmentation Information field.

*Instruct the editor to create section 12.X*

**12.X PQC Key Exchanges**

**12.X.1 General**

The PQC Key Exchanges are performed between a non-AP STA and an AP to establish a PMKSA using ML-KEM. Five PQC exchanges are defined: one in which authentication is performed with a shared password, another in which authentication is performed with signatures, another employing a signature-less form of authentication using trusted ML-KEM keys, an unauthenticated key establishment exchange, and a PMK Caching indication. In addition, an exchange used to obtain a PQC key from a peer is defined.

There are trust assumptions made about the PQC exchanges which perform authentication. In the signature-less exchange it is assumed that the two peers possess each other’s trusted ML-KEM public encapsulation key. In the password authenticated key exchange (PAKE) it is assumed that the peers share a password or passphrase and an identifier. In the signature exchange it is assumed that the two peers have a way of trusting each other’s signature public key (through, for instance, a certificate issued by a trusted third party).

The PQC protocol definitions in section 12.X use Authentication frames. Each protocol definition defines how to construct an Authentication exchange to effect the protocol. In addition to the components of the Authentication frames defined by the PQC exchanges, PQC exchanges that generate a PMK shall also include:

* in the first Authentication frame, an RSNE to indicate the AKM and the pairwise cipher suite. The Version field shall be set to 1. The Pairwise Cipher Suite Count field shall be set to 1. The AKM Suite Count field shall be set to 1. The PMKID count field shall be set to 0 and the PMKID List field is empty. All other fields shall be as specified in 9.4.2.5 (TIM element) and 12.6.3 (RSNA policy selection in an infrastructure BSS).
* in the second Authentication frame an RSNE to acknowledge the AKM and the pairwise cipher suite. The AKM Suite Count field shall be set to 1, the Pairwise Ciphersuite Count field shall be set to 1, and the PMKID count shall be set to zero with the PMKID List field empty.

Every PQC exchange that generates, or identifies, a PMK shall also generate a digest, which is a hashed transcript of the messages sent that comprise the exchange. This digest is computed by hashing a portion of each of the Authentication frames sent, in order, by each side. The portion that is hashed is defined as the data from the Authentication status code (exclusive) to the end of the frame. If an Authentication frame is fragmented (see section 11.3.4.2.a) the data hashed for a particular message (defined as all frames with a singular authentication algorithm number) shall be the portion of each fragment of the message in the order of their fragment number.

Following PMK identification or establishement, PQC exchanges generate a PTK and use the resulting secrets to encrypt a (Re)Association per 12.16.6. PQC exchanges that generate a PMK use a salt of 32 octets of zero in PTK derivation while the PQC PMK Caching exchange uses a salt consisting of the secret derived by the ML-KEM APIs.

The hash functions used in PQC exchanges depends on the parameter set used for key establishment as shown in table 12.X.

|  |  |
| --- | --- |
| **Parameter Set** | **Hash Algorithm** |
| ML-KEM-512 | SHA256 |
| ML-KEM-768 | SHA384 |
| ML-KEM-1024 | SHA512 |

**Table 12.X PQC Hash algorithms**

**12.X.2 PQC Key Exchange without Signature**

The two peers in this exchange have obtained each other’s trusted public KEM encapsulation keys a priori in a manner outside the scope of this standard. The encapsulation keys used in this exchange do not need to be from the same parameter set. In the event that the encapsulation keys differ, the AP’s key will be used to determine the appropriate hash function to use. A non-AP STA determines which KEM encapsulation key to use with this exchange by checking which of its known and trusted KEM keys hash to the same value as that indicated in the PQC Key Selector element. A match identifies the AP’s encapsulation key.

**12.X.2.1 Signatureless Request**

The non-AP STA initiates this exchange by calling the KEM.Encaps() routine with the AP’s encapsulation key, *pkap*, generating a key, K1, and a ciphertext, c1 and then derives a secret:

(K1, c1) = KEM.Encaps(pkap)

ss = HKDF-Expand(HKDF-Extract(c1,K1), “IEEE 802.11 PQC NoSig Handshake Key”, 64)

It then computes a hash of its KEM encapsulation key and encrypts the resulting digest with AES-SIV-512 using ss as the key. It then generates an Authentication frame with the Authentication algorithm set to <ANA4> and with the transaction sequence number set to 1. It places c1 in the body of a PQC Ciphertext element (see 9.4.2.Y (PQC Ciphertext element)), and the ciphertext output of AES-SIV-512 into the PQC Key Selector field of a PQC Key Selector element. Finally, it transmits the frame to the AP.

**STA --> AP: c1, {H(pksta)}ss**

Upon receipt of the Authentication frame with the Authentication algorithm equal to <ANA4> and the transaction sequence number equal to 1, the AP extracts the ciphertext, c1, from the PQC Ciphertext element and calls the KEM.Decaps() routine with its decapsulation key, skap, to generate a key, K1, and it a secret, ss:

K1 = KEM.Decaps(c1, skap)

ss = HKDF-Expand(HKDF-Extract(c1, K1), “IEEE 802.11 PQC NoSig Handshake Key”, 64)

It then passes the data in the PQC Key Selector field of the PQC Key Selector element to AES-SIV-512 with ss as the key. If decryption fails, the exchange fails. Otherwise, the AP determines which KEM encapsulation key in its database of trusted KEM encapsulation keys has a hash digest that matches the plaintext output from AES-SIV-512. If none is found, the exchange fails. Otherwise, the AP then calls the KEM.Encaps() routine with the non-AP STA’s encapsulation key, *pksta*, generating a key, K2, and a ciphertext, c2.

(K2, c2) = KEM.Encaps(pksta)

**12.X.2.2 Signatureless Response**

This is transmitted to the non-AP STA in an Authentication frame with the transaction sequence number set to 2 and c2 in the body of a PQC Ciphertext element (9.4.2.Y (PQC Ciphertext element)).

**STA <-- AP: c2**

The AP then generates shared secret PMK and a PMKID using HKDF and H with a hash algorithm from table 12.X for its public key:

PMK = HKDF-expand(HKDF-extract(c1|c2, K1|K2|pksta|pkap), “IEEE 802.11 PQC NoSig Secret”, 32)

PMKID = Truncate-128(H(c1 | c2))

It then generates a transcript digest, D, using the two Authentication frames per 12.X.1 with the hash algorithm from table 12.X with the AP’s public key.

Upon receipt of an Authentication frame with the transaction sequence number equal to 2, the non-AP STA extracts the ciphertext, c2, from the PQC Ciphertext element and calls the KEM.Decaps() with its decapsulation key, sksta, and c2, generating a key K2:

K2 = KEM.Decaps(c2, sksta)

The non-AP STA then generates shared secret PMK and a PMKID using HKDF and H with a hash algorithm from table 12.X for the AP’s public key:

PMK = HKDF-expand(HKDF-extract(c1|c2, K1|K2|pksta|pkap), “IEEE 802.11 PQC NoSig Secret”, 32)

PMKID = Truncate-128(H(c1 | c2))

It then generates a transcript digest, D, in the same fashion as above.

**12.X.3 PQC Key Exchange with Digital Signature**

Authentication using digital signatures is accomplished by doing a SIGMA-like exchange using a KEM to derive a shared secret and each peer signing context of the exchange with PQC digital signature algorithm to authenticate. The AP and non-AP STA have static keypairs suitable for post-quantum digital signatures, ska/pka and sks/pks respectively.

It is assumed that the peers have a priori trust of each other’s public digital signature key or have a way of obtaining such trust, such as use of a trusted third party which has signed certificates containing the peer’s public key. In addition to the static keypair, the non-AP STA also generates an ephemeral key pair using a post-quantum key exchange algorithm.

When ML-DSA keys are used the algorithms Sign() and Verify() are the pure versions from FIPS 204 with the optional context being the empty string. The pre-hashed signature variant MUST NOT be used.

**12.X.3.1 Signature Initial Request**

A non-AP STA initiates the PQC Digital Signature exchange by first generating an ephemeral keypair using a post-quantum key exchange algorithm:

(esk, epk) = KEM.KeyGen()

Then, it constructs an Authentication frame with the Authentication algorithm set to <ANA3>, the transaction sequence number set to 1, and a PQC Key element in the body of the frame. The parameter set of the key pair indicated in the KEM Parameter Set of the PQC Key Element, and the public key, epk, is the PQC Key field. The frame is transmitted to the AP.

**STA --> AP: epk**

Upon receipt of an Authentication frame with the transaction sequence number equal to 1, the AP checks the KEM Parameter set of the PQC Key element. If the parameter set not supported, the AP constructs an Authentication frame with the Authentication algorithm set to <ANA3>, and the status set to INVALID\_PUBLIC\_KEY. The frame is transmitted to the non-AP STA and the exchange fails. If the parameter set is supported, the AP extracts the public key, pk, from the PQC Key field generates a key, ciphertext, and a session identifier, sid:

(K, c) = KEM.Encaps(epk)

sid = rand()

The AP then derives secrets as:

bk = HKDF-Extract(c, K))

ke = HKDF-Expand(bk, “IEEE 802.11 PQC Sig Handshake Key”, 64)

km = HKDF-Expand(bk, “IEEE 802.11 PQC Sig Mac Key”, n)

where n is the length of the digest output by the hash algorithm from table 12-X.

**12.X.3.2 Signature Initial Response**

The AP then constructs an Authentication frame with the Authentication algorithm set to <ANA3>, the transaction sequence number set to 2, and a Session element and a PQC Ciphertext element in the body of the frame. The ciphertext c is copied into the Ciphertext field of the PQC Ciphertext element and the sid is encrypted using AES-SIV-512 with ke as the key and the entire PQC Ciphertext element as the sole component of AAD. The output of AES-SIV-512 encryption is copied to the Session field of the Session element. The frame is transmitted to the non-AP STA.

**STA <-- AP: c, session({sid}ke)**

Upon receipt of an Authentication frame with the transaction sequence number equal to 2, the non-AP STA extracts the ciphertext, c, from the Ciphertext field of the PQC Ciphertext element and generates a key using its ephemeral private key:

K = KEM.Decaps(c, esk)

The non-AP STA then derives secrets as:

bk = HKDF-Extract(c, K))

ke = HKDF-Expand(bk, “IEEE 802.11 PQC Sig Handshake Key”, 64)

km = HKDF-Expand(bk, “IEEE 802.11 PQC Sig Mac Key”, n)

where n is the length of the digest output by the hash algorithm from table 12-X. The non-AP STA then decrypts the Session field of the Session element using AES-SIV-512 with ke as the key and the entire PQC Ciphertext element as a single component of AAD. If decryption is unsuccessful, the exchange fails, otherwise the non-AP STA retains sid as the session identifier.

**12.X.3.3 Signature Certificate Request**

The non-AP STA then constructs an Authentication frame with Authentication algorithm set to <ANA3>, the transaction sequence number set to 3, and a Public Key element with a Key Type of 4 in the body of the frame. It then encrypts its certificate using AES-SIV-512 with ke and no AAD and places the resulting output into the Public Key field and sets the length to be the length of the ciphertext received from AES-SIV-512. The non-AP STA sends the frame to the AP.

**STA --> AP: publickey({certsta}ke)**

Upon receipt of an Authentication frame with transaction sequence number equal to 3, the AP constructs an Authentication frame with Authentication algorithm set to <ANA3> and the transaction sequence number set to 4. It then extracts the encrypted certificate from the Public Key field of the Public Key element and passes it to AES-SIV-512 with ke. If decryption fails, the AP sets the status of the Authentication frame to REQUEST\_DECLINED and transmits the frame to the non-AP STA, the exchange fails. If decryption succeeds, the AP validates the non-AP STA’s certificate. If validation fails, the AP sets the status of the Authentication frame to UNSUPPORTED\_AUTH\_ALGORITHM and transmits the frame to the non-AP STA, the exchange fails.

**12.X.3.4 Signature Certificate Response**

If validation is successful, the AP adds a Public Key element with Key Type set to 4 to the body of the Authentication frame and then encrypts its own certificate using AES-SIV-512 with ke placing the output in the Public Key field. The AP transmits the frame to the non-AP STA.

**STA <-- AP: publickey({certap}ke)**

Upon receipt of an Authentication frame with the transaction sequence number equal to 4, the non-AP STA extracts the encrypted certificate from the Public Key field of the Public Key element and passes it to AES-SIV-512 with ke. If decryption fails, the exchange fails. If decryption is successful, the STA validates the AP’s certificate. If validation is fails the exchange fails. If validation is successful, the non-AP STA produces a keyed digest of its certificate using km:

msta = HMAC(km, cert-STA)

where HMAC() is instantiated with the hash algorithm from 12.X and the parameter set of the STA’s ephemeral key. It then generates a digital signature using its private static signature key, sks, over a concatenation of the epk, c, and the sid:

sigsta = Sign(sks, epk | c | sid)

**12.X.3.5 Signature Authentication Request**

The non-AP STA then constructs an Authentication frame with the Authentication algorithm set to <ANA3>, the transaction sequence number set to 5, and a PQC Signature element and MIC element in the body of the frame. The non-AP STA encrypts the signature using AES-SIV-512 with ke as the key and copies ciphertext output into the signature into the Signature field of the PQC Signature element. It then encrypts msta using AES-SIV-512 with ke as the key and copies the ciphertext into the MIC field of the MIC element and transmits the frame to the AP.

**STA --> AP: signature({sigsta}ke), MIC({msta}ke)**

Upon receipt of an Authentication frame with the transaction sequence number equal to 5, the AP constructs an Authentication frame with the Authentication algorithm set to <ANA3> and the transaction sequence number set to 6. It then extracts and decrypts the non-AP STA’s signed data, sigsta, from the Signature field of the PQC Signature algorithm and the MIC data from the MIC field of the MIC element. If decryption fails for either the exchange fails and the AP sets the status of the Authentication frame to FILS\_AUTHENTICATION\_FAILURE and transmits the frame to the non-AP STA. Otherwise, it constructs a check value for the MIC and verifies the signature using the non-AP STA’s public key from its certificate to check the signature:

msta’ = HMAC(km, cert-STA)

good-sta = Verify(pks, epk | c | sid, sigsta)

where Verify() is the call to the verification algorithm for the particular digital signature algorithm. If good-sta is false, or if the MIC field does not match msta’, the AP sets the status of the Authentication frame to FILS\_AUTHENTICATION\_FAILURE and transmits the frame to the non-AP STA. Otherwise, the exchange succeeds and the AP produces a keyed digest of its certificate using km and a signature of a concatenation of c, epk, and the sid:

map = HMAC(km, cert-AP)

sigap = Sign(ska, c | epk | sid)

where Sign() is the call to the digital signature algorithm.

**12.X.3.6 Signature Authentication Response**

The AP encrypts its signature using AES-SIV-512 and ke as the key and puts the resulting ciphertext in a PQC Signature element. It encrypts map using AES-SIV-512 and ke as the key and puts the resulting ciphertext in a MIC element in the body of the Authentication frame. The frame is transmitted to the non-AP STA. The exchange succeeds for the AP.

**STA <-- AP: signature({sigap}ke), MIC({map}ke)**

Upon receipt of an Authentication frame with the transaction sequence number equal to 6, the non-AP STA extracts and decrypts the signature, sigap, from the Signature field of the PQC Signature element and the map from the MIC field of the MIC element using AES-SIV-512 and ke as the key. If decryption of either fails the exchange fails. Otherwise, it constructs a check value for the MIC and uses the AP’s public key from its certificate to verify the signature:

map’ = HMAC(km, cert-AP)

good-ap = Verify(pka, c | epk | sid, sigap)

where Verify() is the call to the verification algorithm for the particular digital signature algorithm. If good-ap is false, or if the MIC value does not match map’, the exchange fails. Otherwise, the exchange succeeds for the STA.

If the exchange succeeds, the AP and non-AP STA generate a transcript digest of Authentication frames 1 through 6 per 12.X.1, using the hash algorithm from table 12.X. They then produce a PMK and PMKID as follows:

PMK = HKDF-Expand(bk, “IEEE 802.11 PQC Signature PMK”, 32)

PMKID = Truncate-128(H(sid | epk | c))

**12.X.4 PQC Exchange with Password Authentication**

OQUAKE is used to perform password authentication with PQC (PQPAKE). OQUAKE is a variant of the Encrypted Key Exchange and uses Kemeleon encodings to transform an ML-KEM encapsulation key into a random string prior to encipherment.

**12.X.4.1 Kemeleon**

The ML-KEM encapsulation key is a vector of polynomials, each with 256 coefficients, and a 32 octet random string, *rho*. Each coefficient is a12 bit integer less than 3329. Before running the Kemeleon encoding, it is necessary to deserialize the vector of polynomials into an array of numbers. With n = 256, q = 3329, k from the ML-KEM parameter set, and t from table 12.Y, Kemeleon encode is:

KemeleonEncode(pk)

w | rho = pk

a = deserialize(w)

r = 0

b = log\_2(q^(k\*n))

for i from 1 to k\*n:

r += q^(i-1)\*a[i]

m = rand[0,...,floor((2^(b+t)-r)/(q^(k\*n)))]

r = r + m\*q^(k\*n)

z = r | rho

return z

|  |  |  |
| --- | --- | --- |
| **Parameter Set** | **Value of t** | **Size of encoded key** |
| ML-KEM-512 | 128 | 797 |
| ML-KEM-768 | 192 | 1180 |
| ML-KEM-1024 | 256 | 1562 |

**Table 12.Y—Kemeleon Security Parameter**

Encoded keys must be decoded and the decoded output must be serialized back into the form of a vector of polynomials and appended with rho in order to reconstruct the key. Using the same constants, Kemeleon decode is:

KemeleonDecode(z)

r | rho = z

r = r % q^(k\*n)

for i from 1 to k\*n:

a[i] = r % q

r = r / q

w = serialize(a)

pk = w | rho

return pk

**12.X.4.2 PQC Password Authenticated Key Exchange (PQPAKE)**

The non-AP STA provides an identity in the first message of PQPAKE. The AP processes the identity to determine the appropriate password to use with the exchange. As part of the AP’s response, it generates a new identity which differs from the identity the STA provided and encrypts it with an ephemeral secret. This allows the non-AP STA to pass identifiers in subsequent PQPAKE exchanges in a privacy-preserving manner. To the non-AP STA the new identity it receives from a successful PQPAKE exchange is opaque, but to the AP it has significance in that it can be processed to identify the password.

Note: It is RECOMMENDED that the AP maintain a single identifier for each password and use a deterministic key-wrapping algorithm, like AES-SIV, with a sufficient amount of random and changing salt to securely wrap the identity. Each exchange gets new random salt ensuring the opaque string provided to the non-AP STA is unique and non-repeating.

OQUAKE uses the password, *pwd*, and several constants:

* A domain separation tag: DST = SHA256(“IEEE 802.11 PQC PAKE”)
* Length of a random number: RLEN = 96
* Length of an authenticating tag: NKC = 64
* Length of the output key: NKEY = 32

**12.X.4.3 PAKE Commit Request**

The STA initiates the OQUAKE exchange by obtaining the password, pwd, from dot11RSNAConfigPasswordValueTable and then generates a full session identifier, fsid, by appending the opaque identity associated with that password to the non-AP STA’s and AP’s MAC addresses:

fsid = STAMAC | APBSSID | identity

Note: the first time the non-AP STA uses the identifier it will be the provisioned identifier. Every subsequent run of the PQC PAKE protocol will use the opaque identifier from the previous run as the current identifier.

The non-AP STA then generates a key pair, and encodes the public key to obtain a randomized string, z.

(sk, pk) = ML-Kem.KeyGen()

z = KemeleonEncode(pk)

The STA then generates a random number, r of RLEN octets. The STA then computes T and s using HKDF with the hash algorithm described in Table 12.X as follows:

t\_ikm = HKDF-Extract(pwd, DST | “OQUAKE” | fsid | r)

t\_pad = HKDF-Expand(t\_ikm, “t\_pad”, len(z))

T = z xor t\_pad

s\_ikm = HKDF-Extract(pwd, DST | “OQUAKE” | fsid | T)

s\_pad = HKDF-Expand(s\_ikm, “s\_pad”, RLEN)

s = r xor s\_pad

The STA then constructs an Authentication frame with the Authentication algorithm set to <ANA5>, the transaction sequence number set to 1, a Password Identifier element and a PQC Commit element in the body of the frame. It then copies its opaque identity into the Identifier field of the Password Identifier element, copies s into the Random Commit field of the Commit element, sets the KEM Parameter set to the ML-KEM parameter set of the keypair created above, and copies T into the Encoded Public Key field.

**STA --> AP: pi(identifier), commit(s, T)**

This frame is transmitted to the AP.

Upon receipt of an Authentication frame with the transaction sequence number equal to 1, the AP processes the received identifier to determine the password to use in the PQCPAKE exchange. If processing fails, or if there is no entry in dot11RSNAConfigPasswordValueTable with processed identity, the AP may choose to abandon this exchange or it may, optionally, continue with a random password and random identity in order to prevent an attacker from querying the existance of identities in its configuration. If the AP does not abandon the exchange at this point it generates the full sid as above and inspects the KEM Parameter set in the PQC Commit element. If the value indicates a parameter set that is not acceptable, the exchange fails. Otherwise, s and T are extracted from the PQC Commit element and processed as follows using the same OQUAKE constants as above:

s\_ikm = HKDF-Extract(pwd, DST | “OQUAKE” | fsid | T)

s\_pad = HKDF-Expand(s\_ikm, “s\_pad”, RLEN)

r = s xor s\_pad

t\_ikm = HKDF-Extract(pwd, DST | “OQUAKE” | fsid | r)

t\_pad = HKDF-Expand(t\_ikm, “t\_pad”, len(T))

z = T xor t\_pad

The randomized string z is then passed to KemeleonDecode to reconstruct the key pk. The AP generates a key and ciphertext as follows:

(K, c) = ML-KEM.Encaps(pk)

The AP generates a PMK, a tag, an ephemeral secret, and a PMKID as follows:

prk\_pmk = HKDF-Extract(pwd, DST | "OQUAKE" | fsid | c | K)

PMK = HKDF-Expand(prk\_pmk, DST | "sk", NKEY)

tag = HKDF-Expand(prk\_pmk, DST | “AP confirm”, NKC)

esk = HKDF-Expand(prk\_pmk, DST | “ephemeral secret”, 64)

PMKID = Truncate-128(H(s | T | tag | fsid)

The AP then processes the underlying identity from dot11RSNAConfigPasswordValueTable to produce a new, opaque identity for the STA to use. This new identity is encrypted with AES-SIV-512 using esk as a key.

**12.X.4.4 PAKE Commit Response**

The AP then generates an Authentication frame with the Authentication algorithm set to <ANA5> and the transaction sequence number set to 2 and the encrypted identifier, and a PQC Ciphertext element followed by a MIC element in the body of the frame. It copies the tag to the MIC field of the MIC element, and the Ciphertext in the Ciphertext field of the PQC Ciphertext Element. This frame is transmitted to the STA.

**STA <-- AP: c, {identity}esk, mic**

Upon receipt of an Authentication frame with the transaction sequence number set to 2, the STA makes a copy of the new opaque identifier, extracts the tag and ciphertext, c, from the frame and processes them as follows:

K = ML-KEM.Decaps(c, sk)

prk\_pmk = HKDF-Extract(pwd, DST | "OQUAKE" | fsid | c | K)

verify = HKDF-Expand(prk\_pmk, DST | “AP confirm”, NKC)

If verify differs from tag, the protocol fails.

Otherwise generates the ephemeral secret, decrypts the new opaque identity, stores the new opaque identifier in its network profile configuration for subsequent use, and produces a PMK and PMKID as follows:

esk = HKDF-Expand(prk\_pmk, DST | “ephemeral secret”, 64)

PMK = HKDF-Expand(prk\_pmk, DST | "sk", NKEY)

PMKID = Truncate-128(H(s | T | tag | fsid)

**12.X.4.5 PAKE Confirm**

The non-AP STA then computes its own authenticating tag:

tag2 = HKDF-Expand(prk\_pmk, DST | “STA confirm”, NKC)

It then generates an Authentication frame with the Authentication algorithm set to <ANA5> and the transaction sequence number set to 3 with a MIC element in the body of the frame. It copies the tag2 to the MIC field of the MIC element and transmits the frame to the AP.

Upon receipt of an Authentication frame with the transaction sequence number equal to 3, the AP computes a verifier:

verify2 = HKDF-Expand(prk\_pmk, DST | “STA confirm”, NKC)

and extracts tag2 from the MIC field of the MIC element in the received frame. If verify2 differs from tag2, the authentication exchange fails and the AP deletes the PMK, PMKID, and all state associated with this exchange. Otherwise, the exchange succeeds.

If the exchange succeeds, the AP and non-AP STA generate a transcript digest of Authentication frames 1 through 3 per 12.X.1, using the hash algorithm from table 12.X.

**12.X.5 Opportunistic KEM**

The Opportunistic KEM exchange is performed between a STA and AP in order to create an unauthenticated PMKSA. This exchange establishes a shared secret suitable to generate keys for encryption but that secret is unauthenticated. This exchange is designed for cases where encryption is desired but access control is either not necessary or is handled outside of this standard.

**12.X.5.1 Opportunistic Request**

The Opportunistic KEM exchange is a simple exchange of an KEM encapsulation key and a ciphertext. The STA initiates the exchange by generating a keypair:

(sk, pk) = KEM.KeyGen()

The STA then constructs an Authentication frame with the Authentication algorithm set to <ANA6> and with the transaction sequence number set to 1, with a PQC Key Element with the parameter set of the key pair indicated in the KEM Parameter Set and pk in the PQC Key field. The STA transmits this frame to the AP.

Upon receipt of an Authentication frame with transaction sequence 1, the AP inspects the PQC Key Element. If the KEM Parameter Set is not acceptable the exchange fails. Otherwise, the AP extracts the encapsulation key from the PQC Key field and generates a key and a ciphertext:

(K, c) = Kem.Encaps(pk)

**12.X.5.2 Opportunistic Response**

The AP constructs an Authentication frame with the Authentication algorithm set to <ANA6> and the transaction sequence number set to 2, with a PQC Ciphertext element and c in the Ciphertext field. The AP transmits this frame to the STA. It then computes a transcript digest of Authentication frames 1 and 2 per 12.X.1 using the hash algorithm from table 12.X.

Upon receipt of an Authentication frame with transaction sequence 2, the STA extracts the ciphertext from the PQC Ciphertext element and generates a key:

K = KEM.Decaps(sk, c)

It then computes a transcript digest in the same fashion as the AP.

Both the STA and AP export their transcript digest D, and generate a PMK and a PMKID as:

PMK = HKDF-Expand(HKDF-Extract(c, K), “IEEE 802.11 Opportunistic KEM”, 32)

PMKID = Truncate-128(H(pk | c))

Where H() is the hash algorithm from table 12.X.

**12.X.6 PQC PMK Caching**

If a STA believes it shares a PMKSA with an AP it can attempt to use the corresponding PMK in order to avoid an additional PQC authentication. To do PQC PMK caching, the non-AP STA constructs an Authentication frame with the Authentication algorithm set to <ANA7> and the transaction sequence number set to 1, and the Authentication algorithm field set to PQC PMK Caching and:

* Include an RSNE to indicate the pairwise cipher suite. The Version field shall be set to 1. The Pairwise Cipher Suite Count field shall be set to 1. The PMKID count field and the PMKID List field is set corresponding to PMKSA identifiers. The AKM Suite Count shall be set to the PMKID count and the AKM Suite selectors shall be those from the PMKSA as identified by the PMKID. All other fields shall be as specified in 9.4.2.5 (TIM element) and 12.6.3 (RSNA policy selection in an infrastructure BSS).
* Include an RSNXE.
* Include a PQC Key element.
  + Select a KEM parameter set from 9.4.2.W.
  + Generate an ML-KEM keypair, sk/ek, using the indicated parameter set, indicate the size of the resulting encapsulation key in the Length of Public Key field, and place the encapsulation key, ek, into the KEM key field.

The non-AP STA then transmits the Authentication frame to the AP.

Upon receipt of an Authentication frame with the transaction sequence number equal to 1 the AP generates an Authentication frame with the Authentication algorithm set to <ANA 7> and the transaction sequence number set to 2. Next, the AP checks the PMKID(s) in the RSNE to see if it has any PMKSAs in its database of PMKSAs whose PMKIDs match those listed in the RSNE. If not, the AP sets the status code of the Authentication frame to REASON\_INVALID\_PMKID, and terminates the exchange. If the AP has a PMKID match it:

* Verify that the corresponding AKM indicated in the RSNE is supported and corresponds to the AKM in the named PMKSA. Otherwise, the responder shall reject set the status code of the Authentication frame to STATUS\_INVALID\_AKMP and fail the exchange.
* Verify that the pairwise cipher indicated in the RSNE is supported. Otherwise, the responder shall set the status code of the Authentication frame to STATUS\_INVALID\_PAIRWISE\_CIPHER and fail the exchange.
* Validate that the parameter set in the PQC Key element is supported. If not the status code in the Authentication frame is set to INVALID\_PUBLIC\_KEY and the exchange fails.

If the status code is non-zero, the Authentication frame is transmitted to the non-AP STA and the exchange fails. Otherwise, the AP extracts the encapsulation key, ek, from the KEM Key field of PQC Key element and generates a secret, K, and a ciphertext, c, using the ML-KEM.encapsulate API for the indicated parameter set:

(K, c) = ML-KEM.encaps(ek)

The AP then:

* Include an RSNE to indicate the AKM of the selected PMKSA and pairwise cipher indicated in the first Authentication frame with the PMKID corresponding to the PMKSA in the RSNE.
* Include a Ciphertext element, setting the Length of Ciphertext field to the length of the ciphertext, and copying the ciphertext, c, into the Ciphertext field.

The Authentication frame is transmitted to the STA. The AP then computes a transcript digest, D, of Authentication frames 1 and 2 using the hash algorithm from table 12.X based on the parameter set of the encapsulation key in the PQC Key element, assigns K to be the salt and the PMK from the selected PMKSA as the PMK and proceeds to Key Derivation and Proof-of-Possession per 12.X.7. The exchange terminates successfully for the AP.

Upon receipt of an Authentiation frame with the transaction sequence number equal to 2 and a zero status, the STA checks if the PMKID in the RSNE is one it sent in the first frame. If not, it abandons the exchange. If so, it extracts the ciphertext, c, from the Ciphertext field of the Ciphertext element and generates a secret using c and its secret key, sk:

K = MK-KEM.decaps(c, sk)

The STA then computes a transcript digest, D, of Authentication frames 1 and 2 using the hash algorithm from table 12.X based on the parameter set of the STA’s encapsulation key, assigns K to be the salt and the PMK from the selected PMKSA as the PMK and proceeds to Key Derivation and Proof-of-Possession per 12.X.7. The exchange terminates successfully for the non-AP STA.

*Instruct the editor to create section 12.X.7*

**12.X.7 Key Derivation and Proof-of-Possession for PQC Exchanges**

The 4-way handshake (12.7.6.4 (4-way handshake)) performs derivation of a PTK and proof-of-possession of the PMK between two peers for classical cryptographic, non PQC, protocols. PQC exchanges do PTK derivation and proof-of-possession after completion of the authentication exchange [TBD: align with 11bi protected association text but do this form of key derivation] by computing a digest of the transcript of the PQC exchange and binding that to the keys derived from the key derivation function.

Upon successfully terminating, each PQC key exchange protocol shall generate a PMK, a PMKID, and the digest representing a hash of the transcript, D. The PQC key exchanges use a salt of 32 octets of the value zero which the PQC PMK Caching exchange uses the ephemeral secret generated as a salt.

This data is used to derive a PTK as:

PTK = HKDF-expand(HKDF-extract(salt, PMK | D), “IEEE 802.11 PQC PTK Derivation” | SPA | AUA, PTKLen)

Where:

* salt is either 32 octets of zero or an ML-KEM-derived secret.
* PMK is the shared secret created as a result of running the PQC algorithm
* D is a digest of the transcript from the PQC algorithm that generated the PMK
* For MLO, SPA is the non-AP MLD MAC address, otherwise it is the MAC address of the non-AP STA
* For MLO, AUA is the AP MLD MAC address, otherwise it is the BSSID of the AP/Authenticator
* PTKLen is the Bits required for KCK, TK, and KDK depending on the pairwise cipher negotiated during authentication
* The hash algorithm used with HKDF is the one defined for the AKM of the PMK’s PMKSA; see Table 9-190

PTK is composed of the Key Confirmation Key (KCK), Temporal Key (TK), and the Key Derivation Key (KDK), which are derived as follows:

KCK = ExtractBits(PTK, 0, 256)

TK = ExtractBits(PTK, 256, TK\_Length\_Bits)

TK is the transient key whose length is the same as a key for the pairwise cipher in the RSNE provided by the AP in the second Authentication frame which preceded PTK generation. TK\_Length\_Bits is the TK\_bits in Table 12-8.

KDK = ExtractBits(PTK, 256 + TK\_Length\_Bits, KDK\_bits)

The KDK is of bit length KDK\_bits, which has the value 256 if a KDK is derived (see 12.7.1.3) or 0 otherwise.

KDK shall be derived if dot11SecureLTFImplemented is true and the peer STA has indicated Secure HE-LTF

support capability in its advertised Extended Capabilities.

**References:**

FIPS 203 <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf>

FIPS 204 <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>

Kemeleon <https://datatracker.ietf.org/doc/html/draft-veitch-kemeleon-00>

OQUAKE <https://datatracker.ietf.org/doc/html/draft-vos-cfrg-pqpake-00>

SIGMA: <https://www.iacr.org/cryptodb/archive/2003/CRYPTO/1495/1495.pdf>