IEEE P802.11  
Wireless LANs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resolution to open security comments not related to SIV | | | | |
| Date: 2014-03-18 | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Ping FANG | Huawei Technologies | Vision Business Park, Road Gaoxin south 9th, Nanshan, Shenzhen, China | +86 755 36835832 | [Ping.fang@huawei.com](mailto:Ping.fang@huawei.com) |
| Rob Sun | Huawei |  |  |  |
| Peter Yee | NSA/IAD |  |  | peter@akayla.com |

Abstract

This submission proposes resolution to CIDs 2196, 2197, 2198, 2200, 2201, 2202, 2203, 2204, 2496, 2497, 2805, 2875, 2876, 2877, 2988, 2989, 2991, 2992, 2994, 2995, 2996, 2997, 3001, , , 3086, 3087, 3088, , 3153, 3154, , 3192, 3193, , and 3245.

*This contribution made changes based on 14/341r2 using tracking mode.*

***Instruct the editor to modify tables 8-32 and 8-33 as indicated:***

**8.3.3.11 Authentication frame format**

**Table 8-32—Authentication frame body**

|  |  |  |
| --- | --- | --- |
| 5 | RSN | The RSNE is present in the FT Authentication frames and FILS Authtentication frames as defined in Table 8-33 (Presence of fields and elements in Authentication frames). |

**Table 8-33—Presence of information elements in Authentication frames**

|  |  |  |  |
| --- | --- | --- | --- |
| **Authentication**  **Algorithm** | **Authentication**  **Transaction**  **Sequence no.** | **Status**  **Code** | **Presence of fields 4-20** |
| FILS | 1 | Status | FILS Session is present  FILS Identity is present  FILS Authentication Type is present.  FILS Nonce is present.  RSNE is present.  FILS Wrapped Data is present if FILS shared key authentication is used.  Finite cyclic group is present if FILSAuthentication type field indicates PFS or if FILS public key authentication is used.  Element is present if FILSAuthentication type field indicates PFS or if FILS public key authentication is used. |
| FILS | 2 | Status | FILS Session is present  RSNE is present  FILS Identity is present if Status is 0.  FILS Authentication Type is present if Status is 0.  FILS Nonce is present if Status is 0.  FILS Wrapped Data is present if Status is 0 and  FILSshared key authentication is used.  Finite cyclic group is present if FILSAuthentication type field indicates PFS or if FILS public key authentication is used.  Element is present if FILSAuthentication type field indicates PFS or if FILS public key authentication is used. |

***Instruct the editor to modify table 8-42 as indicated:***

**8.4.1.9 Status code field**

**Table 8-42—Status codes**

|  |  |  |
| --- | --- | --- |
| **Status** | **Name** | **Meaning** |
| <ANA> |  | Authentication rejected due to FILS authentication failure |

***Instruct the editor to modify table 8-101 as indicated:***

**8.4.2.27.3 AKM suites**

**Table 8-101—AKM suite selectors**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00-0F-AC | <ANA-1> | FILS | FILS key management as defined in 11.11.2 (FILS authentication protocol) using SHA256 and CCM-128 | Defined in 11.11.2.3 (Key derivation with FILS authentication ) |
| 00-0F-AC | <ANA-2> | FILS | FILS key management as defined in 11.11.2.2 (FILS authentication protocol) using SHA384 and CCM-256 | Defined in 11.11.2.3 (Key derivation with FILS authentication) |
|  |  |  |  |  |
|  |  |  |  |  |

***Instruct the editor to modify section 11.6.1.7.2 as indicated:***

**11.6.1.7.2 Key derivation function (KDF)**

The KDF for the FT key hierarchy, and for AKMs 00-0F-AC:11, 00-0F-AC:12, 00-0F-AC:<ANA-1>, and 00-0F-AC:<ANA-2>, is a variant of the pseueo-random function (PRF) defined in 11.6.1.2 (PRF) and is defined as follows:

***Instruct the editor to modify section 11.6.2. as indicated:***

**11.6.2 EAPOL-Key frames**

1. **Key Information**. This field is 2 octets and specifies characteristics of the key. See Figure 11-33 (Key Information bit layout).
   1. Key MIC (bit 8): when using a non-AEAD cipher, this bit is set to 1 if a MIC is in this EAPOL-Key frame and is set to 0 if this message contains no MIC. When using an AEAD cipher this bit is set to 0.
2. **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. When using a non-AEAD cipher, it should be initialized by taking the current value of the global key counter (see 11.6.11 (RSNA Authenticator key management state machine)) and then incrementing the counter. Note that only the lower 16 octets of the counter value are used. When the AKM is 00-0F-AC-<ANA-1> or 00-0F-AC-<ANA-2> the current value of the AEAD counter from the PTKSA is copied to the left-most 13 octets of this field.**Key MIC**. When using a non-AEAD cipher, 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. When using an AEAD cipher, the EAPOL Key MIC is empty. The length of this field depends on the negotiated AKM as defined in 11.6.3 (EAPOL-Key frame construction and processing).
3. **Key Data**.

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. If the Key Data field uses an AEAD cipher, then the Key Data field shall not be padded and the AAD for the encipherment operation shall be the data of the EAPOL-Key frame from the EAPOL protocol version field (inclusive) to the Key Data field (exclusive). If the AEAD cipher requires a unique counter it shall use the EAPOL-Key IV. Key Data fields that are encrypted, but do not contain the GroupKey or SMK KDE, shall be accepted.

***Instruct the editor to modify section 11.6.3 (and table 11-8) as indicated:***

**11.6.3 EAPOL-Key frame construction and processing**

**Table 11-8—Integrity and Key Wrap Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **AKM** | **Integrity Algorithm** | **Size of MIC** | **Key Wrap Algorithm** |
| 00-0F-AC-<ANA-1> | AES-CCM-128 | 0 | AES-CCM-128 |
| 00-0F-AC-<ANA-2> | AES-CCM-256 | 0 | AES-CCM-256 |
|  |  |  |  |
|  |  |  |  |

***Instruct the editor to modify section 11.11.2.1 as indicated:***

**11.11.2.1 Key establishment with FILS shared key authentication**

***Instruct the editor to modify section 11.6.1.7.2 as indicated:***

**11.11.2.3 Key derivation with FILS authentication**

Key derivation with FILS Authentication uses the KDF from 11.6.1.7.2 to create keys for a PMKSA—a a Pairwise Master Key (PMK)-- and a PTKSA—a key confirmation key (KCK), a key encryption key (KEK), and a traffic key (TK). In both cases, when the AKM used is 00-0F-AC:<ANA-1> the hash algorithm used for the KDF shall be SHA256 and when the AKM used is 00-0F-AC:<ANA-2> the hash algorithm used for the KDF shall be SHA384.

For PMKSA generation, the inputs to the KDF are a string of zeros whose length is equal to the block size of the hash algorithm used for the kdf , a constant label, the EAP-RP secret result if shared key authentication is being used, and, the Diffie-Hellman shared secret, ss, if PFS is being used or public key authentication is being used. The KDF produces a PMK and a PMKID which is used to uniquely identify the PMKSA. The length of the PMK shall be 256 bits, and the length of the PMKID shall be 128 bits:

PMKID | PMK = KDF-384(<zero>, “FILS PMKSA Derivation”, [rMSK][ss])

Where:

* <zero> is a string of zeros of length 256 or a length of 384, depending on the AKM used
* rMSK is the output of the EAP-RP exchange if shared key authentication was used
* ss is the result of the Diffie-Hellman exchange if public key authentication was used or if PFS was used with shared key authentication

Upon completion of PMK generation the shared secret, ss, and rMSK, if applicable, shall be irretrievably destroyed.

When using PMKSA caching, a new PMKSA is not created. Instead, the PMKSA used for PMKSA caching remains and continues to be identified by the appropriate PMKID. Regardless of whether PMKSA caching is used or not, a PTKSA shall be generated with each FILS authentication exchange.

For PTKSA key generation, the inputs to the KDF are the two 16 octet nonces NSTA and NAP produced by the STA and AP, respectively, a constant label, and the PMK of the PMKSA. When the AKM used is 00-0F-AC:<ANA-1>, the length of KEK shall be 128 bits, and the length of the KCK, shall be 256 bits. When the AKM used is 00-0F-AC:<ANA-2>, the length of the KEK shall be 192 bits, and the length of KCK shall be 384 bits, The total amount of bits extracted from the KDF shall therefore be 384+TK or 576+TK bits depending on the AKM used, where TK\_bits is determined from table 11-4.

KCK | KEK | TK = KDF-X(NSTA | NAP, “FILS PTKSA Derivation”, PMK))

Where:

* X is 384+TK\_bits or 576+TK bits from table 11-4, depending on the AKM used.
* PMK is the PMK from the PMKSA, either created from an initial FILS connection or from a cached PMKSA, when PMKSA caching is used.

If the negotiated AKM is 00-0F-AC-<ANA-1> or 00-0F-AC-<ANA-2>, FILS requires an additional element: a 13 octet AEAD counter to be part of the newly created PTKSA. The STA shall set the AEAD counter to 13 octets of zero and the AP shall set the first octet to the value 128 and the remaining octets to zero (i.e. the first bit of the AEAD counter is 1 and the rest of the bits in the counter are 0).

***Instruct the editor to modify section 11.11.2.4 as indicated, creating sub-sections 11.11.2.4.1 and 11.11.2.4.2:***

**11.11.2.4 Key confirmation with FILS authentication**

Key confirmation for FILS Authentication is an Association Request followed by an Association Response. The Association Request and Association Response shall be protected using KEK according to 11.11.2.6 (Encrypt and authenticate operation for FILS association frames) and 11.11.2.7 (Decrypt and verify operation for FILS association frames).

**11.11.2.4.1 Association Request for FILS key confirmation**

The STA constructs an 802.11 Association response frame for FILS Authentication per section 8.3.3.5 (Association Request frame format). Hashing functions are used to generate the Key Confirmation element and the specific hash function depends on the AKM negotiated (see 8.4.2.27.3 (AKM suites)).

For FILS shared key authentication, the Key Auth field of the Key Confirmation element is constructed by using the HMAC mode of the negotiated hash function with a key of KCK on a concatenation of the STA’s nonce, the AP’s nonce, the STA’s MAC address, and the AP’s BSSID, in that order:

Key-Auth = HMAC-Hash(KCK, NSTA | NAP | STA-MAC | AP-BSSID).

For FILS public key authentication, the Key Auth field of the Key Confirmation element is a digital signature, using the STA's private key, of the negotiated hash function on a concatentation of the STA’s Diffie-Hellman value, the AP’s Diffie-Hellman value, the STA’s nonce, the AP’s nonce, the STA’s MAC address, and the AP’s BSSID, in that order:

Key-Auth = SigSTA[Hash(gSTA | gAP | NSTA | NAP | STA-MAC | AP-BSSID)].

Where SigSTA[] indicates a digital signature using the STA's private key.

The 802.11 Association Request frame shall be securedwith KEK using the AEAD algorithm as defined in 11.11.2.5 (AEAD scheme). The AEAD algorithm takes AAD that is authenticated but not encrypted. The AAD for the 802.11 Association request is constructed by concatenating the following data together in order:

* The STA MAC
* The AP BSSID
* The STA's nonce
* The AP's nonce
* The contents of the Association Request frame from the capability (inclusive) to the FILS Session element (inclusive)

The plaintext passed to the AEAD encryption algorithm is the data that would follow the FILS session element in an unencrypted frame. If the AEAD cipher requires a unique counter, the current value of the AEAD counter from the PMKSA shall be passed to the AEAD encryption algorithm. The ciphertext output by the AEAD encryption operation becomes the data that follows the FILS session element in the encrypted and authenticated 802.11 Association Request frame. The resulting 802.11 Association Request frame shall be transmitted to the AP.

The AP decrypts and verifies the received 802.11 Association Request frame with KEK. The AAD is reconstructed as defined in this section above and is passed, along with the ciphertext of the received frame to the AEAD decrypt operation. If the AEAD cipher mode requires an AEAD counter, the AP implicitly uses the STA’s initial AEAD counter of all zeros to decrypt and verify the received frame.

If the output from the AEAD decryption operation returns a failure, the authentication exchange shall be deemed a failure. If the output does not return failure, the returned plaintext replaces the ciphertext as portion of the frame that follows the FILS session element and processing of the received frame continues by checking the value of the Key Confirmation element.

For FILS shared key authentication, the AP constructs a verifier, Key-Auth’, in an identical manner as the STA constructed its Key-Auth above. The AP compares Key-Auth' with the Key-Auth field in the Key Confirmation element of the received frame.If they differ, authentication shall be deemed a failure.

For FILS public key authentication, the AP uses the STA's (certified) public key from the FILS Public Key element to verify that the contents of the Key-Auth field of the Key Confirmation element consist of a hash of a concatentation of the STA’s Diffie-Hellman value, the AP’s Diffie-Hellman value, the STA’s nonce, the AP’s nonce, the STA’s MAC address, and the AP’s BSSID, in that order, using the negotiated hash function. The specific technique for verification depends on the crypto-system used by the public key. If verification fails, authentication shall be deemed a failure.

If authentication is deemed a failure, KCK, KEK, and TK shall be irretrievably destroyed and the AP shall return an 802.11 Authentication frame with a status code set to <ANA> (Authentication rejected due to FILS authentication failure)..

**11.11.2.4.2 Association response for FILS key confirmation**

The AP constructs an 802.11 Association response frame for FILS Authentication per section 8.3.3.6 (Association Response frame format). As with the Association request frame, hashing functions are used to generate the Key Confirmation element and the specific hash function depends on the AKM negotiated (see 8.4.2.27.3 (AKM suites)).

The AP constructs a GTK Transfer TLV, indicating the current GTK for the BSS, and a Key RSC TLV (8.4.2.181.5 (Key RSC TLV)) indicating the current RSC for the GTK. The AP puts these TLVs into FILS Secure Container (8.4.2.181 (FILS Secure Container element)) of the Association Response frame.

For FILS shared key authentication, the Key Auth field of the Key Confirmation element is constructed by using the HMAC mode of the negotiated hash function with a key of KCK on a concatenation of the AP’s nonce, the STA’s nonce, the AP’s BSSID, and the STA’s MAC address, in that order :

Key-Auth = HMAC-Hash(KCK, NAP | NSTA | AP-BSSID | STA-MAC).

For FILS public key authentication, the Key Auth field of the Key Confirmation element is a digital signature using the AP's private key of the output from the negotiated hash function on a concatentation of the AP’s Diffie-Hellman value, the STA’s Diffie-Hellman value, the AP’s nonce, the STA’s nonce, AP’s BSSID, and the STA’s MAC address, in that order. The specific construction of the digital signature depends on the crypto-system of the public/private keypair:

Key-Auth = SigAP[Hash(gAP | gSTA | NAP | NSTA | AP-BSSID | STA-MAC )].

Where SigAP[] indicates a digital signature using the AP's private key.

The 802.11 Association Response frame shall be secured with KEK. The AAD used with the AEAD algorithm for the 802.11 Association Response is constructed by concatenating the following data together in order :

* The AP BSSID
* The STA MAC
* The AP's nonce
* The STA's nonce
* The contents of the Association Response frame from the capability (inclusive) to the FILS Session element (inclusive)

The plaintext passed to the AEAD encryption algorithm is the data that would follow the FILS session element in an unencrypted frame. If the AEAD cipher requires a unique counter, the current value of the AEAD counter from the PMKSA shall passed to the AEAD encryption algorithm.The ciphertext output by the AEAD encryption operation becomes the data that follows the FILS session element in the encrypted and authenticated 802.11 Association Response frame. The resulting 802.11 Association Response frame shall be transmitted to the STA.

The STA decrypts and verifies the received 802.11 Association Response frame with KEK. The AAD is reconstructed as defined in this section above and is passed with the ciphertext of the received frame to the AEAD decrypt operation. If the AEAD cipher mode requires an AEAD counter, the STA implicitly uses the AP’s initial AEAD counter of the value 128 followed by 12 octets of zero to decrypt and verify the received frame.

If the output from the AEAD decrypt operation returns failure, the authentication exchange shall be deemed a failure. If the output does not return failure, the output plaintext replaces the ciphertext as portion of the frame that follows the FILS session element and processing of the received frame continues by checking the value of the Key Confirmation element.

For FILS shared key authentication, the STA constructs a verifier, Key-Auth’, in an identical manner as the AP constructed its Key-Authabove. :

The STA compares Key-Auth' with the Key-Auth field in the Key Confirmation element of the received frame. If they differ, authentication shall be deemed a failure.

For FILS public key authentication, the STA uses the AP's (certified) public key from the FILS Public Key element to verify that the contents of the Key-Auth field of the Key Confirmation element consists of a hash of a concatentation of the AP’s Diffie-Hellman value, the STA’s Diffie-Hellman value, the AP’s nonce, the STA’s nonce, the AP’s BSSID, and the STA’s MAC address, in that order, using the negotiated hash function. The specific technique for verification depends on the crypto-system used by the public key. If verification fails, authentication shall be deemed a failure.

If authentication is deemed a failure, the KCK2, KEK2, KCK, KEK, PMK, and TK shall be irretrievably destroyed and the STA shall abandon the exchange. Otherwise authentication succeeds and the STA and AP shall irretrievably destroy the temporary keys KCK2 and KEK2 and both shall use the TK with the cipher indicated by the negotiated cipher suite. The KCK, KEK, and PMK shall be used for subsequent key management as specified in clause 11.5. The STA and AP shall set the lifetime of the PMKSA to the value dot11RSNAConfigPMKLifetime.

Upon successful completion of the FILS Authentication procedure, the STA shall process the Secure Container elements as described in 8.4.2.181 (FILS Secure Container element). The STA install GTK and set key RSC. GTK rekeying shall be performed as described in 11.6.7 (Group Key Handshake).

***Instruct the editor to modify section 11.11.2.5 as indicated:***

**11.11.2.5 AEAD cipher mode for FILS**

FILS authentication uses an AEAD cipher mode to preotect Association and EAPOL frames after FILS key establishment. The AEAD cipher mode is determined by the specific FILS AKM negotiated.

AES-CCM-128 is used if the AKM is 00:0F-AC-<ANA-1> and AES-CCM-256 is used if the AKM is 00-0F-AC-<ANA-2>

When the AEAD cipher mode used is CCM, the nonce, N, shall be set to the AEAD counter in the PMKSA as passed in the frame being protected and the following values for parameters M and L shall be used:

* M=16
* L=2

Each successive invocation of the encryption operation of CCM shall increment the AEAD counter by one (1).

***Instruct the editor to remove sections 11.11.2.6 and 11.11.2.7, adjusting the numbering of subsequent subsections as necessary***

**References:**