IEEE P802.11  
Wireless LANs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resolution of some security CIDs | | | | |
| Date: 2015-06-16 | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | Aruba Networks | 1322 Crossman avenue, Sunnyvale, California, 94089, United States of America | +1 408 227 4500 | dharkins at aruba networks dot com |
|  |  |  |  |  |

Abstract

This submission proposes resolutions to CIDs 6106, 6183, 6184, 6275, 6276, 6277, 6278, 6367, 6421, 6509, 6510, 6511, and 6521.

***Instrut the editor to modify table 8-130 in section 8.4.2.24.3 as indicated:***

**8.4.2.24.3 AKM suites**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00-0F-AC | 3 | FT authentication negotiated over IEEE Std 802.1X | FT key management as defined in 11.6.1.7 (FT key hierarchy) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 4 | FT authentication using PSK | FT key management as defined in 11.6.1.7 (FT key hierarchy) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 5 | Authentication negotiated over IEEE Std 802.1X or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | RSNA Key Management as defined in 8.5 or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management), | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 6 | PSK | RSNA Key Management as defined in 11.6 (Keys and key distribution) using PSK | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 7 | TDLS | TPK Handshake | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 8 | SAE authentication with SHA-256 or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | RSNA key management as defined in 11.6 (eys and key distribution), PMKSA caching as defiend in 11.5.10.3 (Cached PMKSAs and RSNA key management) or authenticated mesh peering exchange as defined in 13.5 (Authenticated mesh peering exchange (AMPE)) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 9 | FT authentication over SAE | FT key management define din 11.6.1.7 (FT key hierarchy) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 10 | APPeerKey Authentication with SHA-256 or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | RSNA key management as defined in 11.6 (Keys and key distribution) or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 11 | Authentication negotiated over IEEE 802.1X or using PMKSA caching as defined in 11.5.10.3 (Cachehed PMKSAs and RSNA key management) using Suite B compliant EAP method supporting EC of GF(p=256) | RSNA key management as defiend in 11.6 (Keys and key distribution) or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-256 |
| 00-0F-AC | 12 | Authentication negotiated over IEEE 802.1X or using PMKSA caching as defined in 11.5.10.3 (Cachehed PMKSAs and RSNA key management) using Suite B compliant EAP method supporting EC of GF(p=384) | RSNA key management as defiend in 11.6 (Keys and key distribution) or using PMKSA caching as defined in 11.5.10.3 (Cached PMKSAs and RSNA key management) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-384 |
| 00-0F-AC | 13 | FT authentication negotiated over IEEE 802.1X | FT key management as defined in 11.6.1.7 (FT key hierarchy) | Defined in 11.6.1.7.2 (Key derivation function (KDF)) using SHA-384 |

***Instruct the editor to modify section 11.3.4.2.2 as indicated:***

**11.3.4.2.2 Generation of the password element with ECC groups**

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* x *pwd-value* + b ) is a quadratic residue modulo p

then

*x* = *pwd-value*

*save* = *pwd-seed*

*found* = 1

*base* = random()

fi

fi

*counter* = *counter* + 1

} while ((*counter* <= *k* ) or (*found* =0))

*y* = sqrt(*x*3 + *a*x + b ) modulo *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 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the AKM in Table 8-130 (AKM suite selectors) generating a key of length Length.

***Instruct the editor to modify section 11.3.4.3.2 as indicated:***

**11.3.4.3.2 Generation of the password element with FFC groups**

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  modulo *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 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the AKM in Table 8-130 (AKM suite selectors) generating a key of length Length.

***Instruct the editor to modify section 11.3.5.4 as indicated:***

**11.3.5.4 Proessing of a peer’s SAE Commit message**

The entropy of k shall then be extracted using H to produce keyseed . The key derivation function from 11.6.1.7.2 (Key derivation function (KDF)) shall then be used with the Hash function defined by the AKM in Table 8-130 (AKM suite selectors) 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 zero (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 || PMK = KDF-Hash-512(keyseed , “SAE KCK and PMK”,

(commit-scalar + peer-commit-scalar ) modulo r )

where

KDF-Hash-512 is the key derivation function defined in 11.6.1.7.2 (Key derivation function (KDF)) using the hash function defined by the negotiated AKM in Table 8-130 (AKM suite selectors) to generate keying material of length 512 .

***Instruct the editor to modify section 11.6.9.2 as indicated:***

**11.6.9.2 TDLS Peer Key handshake**

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 defined by the negotiated AKM specified in Table 8-130.

Length = TK\_bits + 128. TK\_bits is cipher-suite specific and specified in Table 11-4 (Cipher suite key lengths)

KDF-Hash-Length is the key derivation function defined in 11.6.1.7.2 (Key derivation function (KDF))

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.

***Instruct the editor to modify section 11.6.1.7.3 as indicated:***

**11.6.1.7.3 PMK-R0**

The first-level key in the FT key hierarchy, PMK-R0, is derived using the KDF defined in 11.6.1.7.2 (Key derivation function (KDF)). 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 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the AKM in Table 8-130 (AKM suite selectors).
* If the AKM negotiated is 00-0F-AC:3, thenQ shall be 256, and XXKey shall be the second 256 bits of the MSK (which is derived from the IEEE Std 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).

***Instruct the editor to modify section 11.6.1.7.4 as indicated:***

**11.6.1.7.4 PMK-R1**

The second-level key in the FT key hierarchy, PMK-R1, is a key used to derive the PTK. The PMKR1 is derived using the KDF defined in 11.6.1.7.2 (Key derivation function (KDF)):

PMK-R1 = KDF-Hash-Length(PMK-R0, "FT-R1", R1KH-ID || S1KH-ID)

Where

* KDF-Hash-Length is the KDF as defined in 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the AKM in Table 8-130 (AKM suite selectors) to generate a key whose length is equal to the bitlength 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.

***Instruct the editor to modify section 11.6.1.7.5 as indicated:***

**11.6.1.7.5 PTK**

Using the KDF defined in 11.6.1.7.2 (Key derivation function (KDF)), 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 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the AKM in Table 8-130 (AKM suite selectors) to generate a PTK of length Length.
* 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 11-4 (Cipher suite key lengths) in 11.6.2 (EAPOL-Key frames) and Table 11-8 (Integrity and key-wrap algorithms) in 11.6.3 (EAPOL-Key frame construction and processing).

***Instruct the editor to add a new section 11.7.9 as indicated:***

**11.7.9 Mapping GTK to GCMP keys**

See 11.6.1.4 (Group key hierarchy) for the definition of the EAPOL temporal key derived from GTK.

A STA shall use the temporal key as the CCMP key.

***Instruct the editor to modify section 11.10.2 as indicated:***

**11.10.2 AP PeerKey Protocol**

The PMK shall be a 256-bit key derived from keyseed using the key derivation function (KDF) from 11.6.1.7.2 (Key derivation function (KDF)) using the hash algorithm defined for the negotiated AMK according to Table 8-130 (AKM suite selectors) according to Equation (11-4) and the PMKID shall be derived according to Equation (11-5).

PMK = KDF-Hash-256(keyseed , “AP Peerkey Protocol”,

0x00 || Max(LOCAL-MAC, PEER-MAC) || Min(LOCAL-MAC, PEER-MAC) ) (11-4)

PMKID = Truncate-128(SHA-256(Q1 || Q2 ||

Max(LOCAL-MAC, PEER-MAC) ||

Min(LOCAL-MAC, PEER-MAC)) (11-5)

where

KDF-Hash-256 is the key derivation function defined in 11.6.1.7.2 (Key derivation function (KDF)) for the negotiated AKM generating a 256-bit key.

***Instrut the editor to modify section 13.5.7 as indicated:***

**13.5.7 Keys and key derivation algorithm for the authenticated mesh peering exchange (AMPE)**

To execute the AMPE and mesh group key handshake with a candidate peer STA, the local STA shall derive an authenticated encryption key (AEK) and a mesh temporal key (MTK) using the PMK it shares with the candidate peer STA.

The AEK is derived statically from the shared PMK. The MTK is derived from the shared PMK and dynamic information provided by the STA and candidate peer STA.

The AEK is mutually derived by the local STA and the peer STA once a new PMK has been selected. The

AEK shall be derived from the PMK by

AEK 🡨 KDF-Hash-256(PMK, “AEK Derivation”, Selected AKM Suite ||

min(localMAC, peerMAC) || max(localMAC, peerMAC)).

where KDF-Hash-256 is the key derivation function defined in 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the negotiated AKM to generate an AEK of length of 256 bits.

The temporal key (MTK) shall be derived from the PMK by

MTK 🡨 KDF-Hash-Length(PMK, “Temporal Key Derivation”, min(localNonce,

peerNonce) || max(localNonce, peerNonce) || min(localLinkID,

peerLinkID) || max(localLinkID, peerLinkID) || Selected AKM Suite ||

min(localMAC, peerMAC) || max(localMAC, peerMAC)).

where KDF-Hash-Length is the key derivation function defined in 11.6.1.7.2 (Key derivation function (KDF)) using the Hash algorithm defined by the negotiated AKM to generate an MTK of a specified length. Both CCMP and GCMP use Length = 128. 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

as specified in 11.6.1.3 (Pairwise key hierarchy). The “min” and “max” operations for nonces are encoded as

specified in 8.2.2 (Conventions). The “min” and “max” operations for LinkIDs select the minimum and

maximum, respectively, of the two unsigned integers.

**References:**