IEEE P802.11
Wireless LANs

|  |
| --- |
| Adding Support for Suite B |
| Date: 2012-05-17 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | Aruba Networks | 1322 Crossman ave, Sunnyvale, CA | +1 408 227 4500 | dharkins at arubanetworks dot com  |
|  |  |  |  |  |

Abstract

This document defines the text modifications necessary to add support for Suite B, which includes GCM-256, to the draft and is in response to the resolution of CID 4261 from LB187 which invited text and a discussion on the topic of GCM-256 and Suite B.

***Instruct the editor to modify Tables 8-99 and 8-100 as indicated:***

* Cipher suites

|  |
| --- |
| * Table 8-99 Cipher suite selectors
 |
| OUI | Suite type | Meaning |
| 00-0F-AC | 8 | GCMP-128 – default for a DMG STA |
| 00-0F-AC | <ANA-1> | GCMP-256 |

|  |
| --- |
| * Table 8-100 Cipher suite usage
 |
| Cipher suite selector | GTK | PTK | IGTK |
| GCMP-128 | Yes | Yes | No |
| GCMP-256 | Yes | Yes | No |

***Instruct the editor to modify section 8.4.2.27.3 as indicated:***

* AKM suites

|  |
| --- |
| * Table 8-101 AKM suite selectors
 |
| OUI | Suite type | Meaning |
| Authentication type | Key management type | Key derivation type  |
| 00-0F-AC | <ANA-2> | Authentication negotiated over IEEE 802.1X or using PMKSA caching as defined in 11.5.9.3 (Cached PMKSAs and RSNA key management) using a Suite B compliant EAP method supporting EC of GF(p=256) | RSNA key management as defined in 11.6 (Keys and key distribution) or using PMKSA caching as defined in 11.5.9.3 (Cached PMKSAs and RSNA key management) with HMAC-SHA256 | Defined in 11.6.1.7.2 using HMAC-SHA256 |
| 00-0F-AC | <ANA-3> | Authentication negotiated over IEEE 802.1X or using PMKSA caching as defined in 11.5.9.3 (Cached PMKSAs and RSNA key management) using a Suite B compliant EAP method supporting EC of GP(p=384) | RSNA key management as defined in 11.6 (Keys and key distribution) or using PMKSA caching as defined in 11.5.9.3 (Cached PMKSAs and RSNA key management) with HMAC-SHA384 | Defined in 11.6.1.7.2 using HMAC-SHA384 |

.

The AKM suite selector value 00-0F-AC:<ANA-2> shall only be used with Cipher suite selector value 00-0F-AC:8 (GCMP-128) and AKM suite selector value 00-0F-AC:<ANA-32> shall only be used with Cipher suite selector value 00-0F-AC:<ANA-1>.

***Instruct the editor to modify section 11.4.5.1 as indicated:***

11.4.5.1 GCMP overview

Subclause 11.4.5. specifies the GCMP, which provides data confidentiality, authentication, integrity, and replay protection.

GCMP is based on the GCM of the AES encryption algorithm. GCM combines Galois/Counter Mode for data confidentiality and GMAC for authentication and integrity. GCM protects the integrity of both the MPDU Data field and selected portions of the MPDU header.

The AES algorithm is defined in FIPS PUB 197-2001. AES processing used within GCMP uses AES with a 128-bit key (GCMP-128) or a 256-bit key (GCMP-256).

***Instruct the editor to modify section 11.6.1.2 as indicated:***

* PRF

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

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

In the following, *K* is a key; *A* is a unique label for each different purpose of the PRF; *B* is a variable-length string; *Y* is a single octet containing 0; *X* is a single octet containing the loop parameter *i*; and || denotes concatenation:

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

 PRF(*K*, *A*, *B*, *Len*)

 **for** *i*  0 **to** (*Len*+159)/160 **do**

 *R*  *R* || H-SHA-1(*K*, *A*, *B*, *i)*

 **return** L(*R*, 0, Len)

 PRF-128(*K*, *A*, *B*) = PRF(*K*, *A*, *B*, 128)

 PRF-192(*K*, *A*, *B*) = PRF(*K*, *A*, *B*, 192)

 PRF-256(*K*, *A*, *B*) = PRF(*K*, *A*, *B*, 256)

 PRF-384(*K*, *A*, *B*) = PRF(*K*, *A*, *B*, 384)

 PRF-512(*K*, *A*, *B*) = PRF(*K*, *A*, *B*, 512)

When the negotiated AKM is 00-0F-AC:5, 00-0F-AC:6, or 00-0F-AC:<ANA-2>, the KDF specified in 11.6.1.7.2 (Key derivation function (KDF)) shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context and the PRF functions are defined as follows:

 PRF-128(K, A, B) = KDF-SHA256-128(K, A, B)

 PRF-192(K, A, B) = KDF-SHA256-192(K, A, B)

 PRF-256(K, A, B) = KDF-SHA256-256(K, A, B)

 PRF-384(K, A, B) = KDF-SHA256-384(K, A, B)

 PRF-512(K, A, B) = KDF-SHA256-512(K, A, B)

When the negotiated AKM is 00-0F-AC:<ANA-3> the KDF specified in 11.6.1.7.2 (Key derivation function (KDF)) shall be used instead of the PRF construction defined here. In this case, A is used as the KDF label and B as the KDF Context and the PRF function is defined as follows:

 PRF-704(K, A, B) = KDF-SHA384-704(K, A, B)

***Instruct the editor to modify section 11.6.1.3 as indicated:***

* Pairwise key hierarchy

Except when preauthentication is used, the pairwise key hierarchy utilizes PRF-384, PRF-512 or PRF-704 to derive session-specific keys from a PMK, as depicted in Figure 11-24 (Pairwise key hierarchy). When using AKM suite selector 00-0F-AC:<ANA-3>, the length of the PMK, PMK\_bits, shall be 384 bits. With all other AKM suite selectors, the length of the PMK, PMK\_bits, shall be 256 bits. The pairwise key hierarchy takes a PMK and generates a PTK. The PTK is partitioned into KCK, KEK, and temporal keys, which are used by the MAC to protect individually addressed communication between the Authenticator’s and Supplicant’s respective STAs. PTKs are used between a single Supplicant and a single Authenticator.

***Instruct the editor to modify figure 11-4 to replace “L(PTK, 0, 128) (KCK)” with “L(PTK, 0, KCK\_bits) (KCK)”, “L(PTK, 128, 128) (KEK)” with “L(PTK, KCK\_bits, KEK\_bits) (KEK)” and “L(PTK,256,TK\_bits) (TK)” with “L(PTK, KCK\_bits+KEK\_bits, TK\_bits) (TK)”***

When not using a PSK, the PMK is derived from the MSK. The PMK shall be computed as the first PMK\_bitsbits (bits 0–PMK\_bits) of the MSK: PMK  L(MSK, 0, PMK\_bits). The PTK shall not be used longer than the PMK lifetime as determined by the minimum of the PMK lifetime indicated by the AS, e.g., Session-Timeout + dot1xAuthTxPeriod or from dot11RSNAConfigPMK--Lifetime. When RADIUS is used and the Session-Timeout attribute is not in the RADIUS Accept message, and if the key lifetime is not otherwise specified, then the PMK lifetime is -infinite.

NOTE 1—If the protocol between the Authenticator (or AP) and AS is RADIUS, then the MS-MPPE-Recv-Key attribute (-vendor-id = 17; see Section 2.4.3 in IETF RFC 2548-1999 [B30]) is available to be used to transport the first 32 octets of the MSK to the AP, and the MS-MPPE-Send-Key attribute (vendor-id = 16; see Section 2.4.2 in IETF RFC 2548-1999 [B30]) is available to be used to transport the remaining 32 octets of the MSK.

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

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

Here, the following assumptions apply:

* SNonce is a random or pseudorandom value contributed by the Supplicant; its value is taken when a PTK is instantiated and is sent to the PTK Authenticator.
* ANonce is a random or pseudorandom value contributed by the Authenticator.
* The PTK shall be derived from the PMK by

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

where X = KCK\_bits + KEK\_bits+ TK\_bits. The values of KCK\_bits and KEK bits are AKM suite dependent and are listed in Table 11-9 (Integrity and key wrap algorithms). The value of TK\_bits is cipher-suite dependent and is defined in Table 11-4 (Cipher suite key lengths). The Min and Max operations for IEEE 802 addresses are with the address converted to a positive integer treating the first transmitted octet as the most significant octet of the integer. The Min and Max operations for nonces are with the nonces treated as positive integers converted as specified in 8.2.2 (Conventions).

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

* The KCK shall be computed as the first KCK\_bits bits (bits 0–KCK\_bits) of the PTK:

KCK  L(PTK, 0, KCK\_bits)

The KCK is used by IEEE Std 802.1X-2004 to provided data origin authenticity in the 4-Way Handshake and Group Key Handshake messages.

* The KEK shall be computed as the next KEK\_bits bits of the PTK:

KEK  L(PTK, KCK\_bits, KEK\_bits)

The KEK is used by the EAPOL-Key frames to provide data confidentiality in the 4-Way Handshake and Group Key Handshake messages.

* The temporal key (TK) shall be computed as the next TK\_bits of the PTK:

TK  L(PTK, KCK\_bits + KEK\_bits, TK\_bits)

The EAPOL-Key state machines (see 11.6.10 (RSNA Supplicant key management state machine) and 11.6.11 (RSNA Authenticator key management state machine)) use the MLME-SETKEYS.request primitive to configure the temporal key into the STA. The STA uses the temporal key with the pairwise cipher suite; interpretation of this value is cipher-suite-specific.

A PMK identifier is defined as

PMKID = HMAC-SHA1-128(PMK, "PMK Name" || AA || SPA)

Here, HMAC-SHA1-128 is the first 128 bits of the HMAC-SHA1 of its argument list.

When the negotiated AKM is 00-0F-AC:5 or 00-0F-AC:6, HMAC-SHA-256 is used to calculate the PMKID, and the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-256(PMK, "PMK Name" || AA || SPA))

When the negotiated AKM is 00-0F-AC:<ANA-2>, HMAC-SHA-256 is used to calculate the PMKID, and the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-256(KCK, "PMK Name" || AA || SPA))

When the negotiated AKM is 00-0F-AC:<ANA-3>, HMAC-SHA-384 is used to calculate the PMKID, and the PMK identifier is defined as

PMKID = Truncate-128(HMAC-SHA-384(KCK, "PMK Name" || AA || SPA))

NOTE—When the PMKID is calculated for the PMKSA as part of RSN preauthentication, the AKM has not yet been negotiated. In this case, the HMAC-SHA1-128 based derivation is used for the PMKID calculation..

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

* Key derivation function (KDF)

The KDF for the FT key hierarchy, and for AKMs 00-0F-AC:<ANA-2> and 00-0F-AC:<ANA-3>, is a variant of the pseudorandom function (PRF) defined in 11.6.1.2 (PRF) and is defined as follows:

**Output**  **KDF-Hash-Length (K, label, Context) where**

Input: *K*, a key derivation key whose length equals the block size of the hash function

 *Hash*, a cryptographically strong hash function

 *label*, a string identifying the purpose of the keys derived using this KDF

 *Context*, a bit string that provides context to identify the derived key

 *Length*, the length of the derived key in bits

Output: a *Length*-bit derived key

*result*  ""

*iterations*  (*Length*+255)/256

**do** *i* = 1 **to** *iterations*

 *result*  *result* || HMAC-Hash(*K*, *i* || *label* || *Context* || *Length*)

**od**

**return** first *Length* bits of *result,* and securely delete all unused bits

In this algorithm, *i* and *Length* are encoded as 16-bit unsigned integers, represented using the bit ordering conventions of 8.2.2 (Conventions). *K*, *label*, and *Context* are bit strings and are represented using the ordering conventions of 8.2.2 (Conventions).

***Instruct the editor to modify section 11.6.3, and table 11-9, as indicated:***

* EAPOL-Key frame construction and processing

EAPOL-Key frames are constructed and processed according to the AKM negotiated at association time. The negotiated AKM determines what algorithm is used to construct and verify a MIC, the size of the MIC, and the algorithm used to wrap and unwrap the Key Data field.

Table 11-9 (Integrity and key-wrap algorithms) indicates the particular algorithms to use when constructing and processing EAPOL-Key frames. The AKM of “Deprecated” indicates an AKM of 00-0F-AC:1 or 00-0F-AC:2 when either TKIP or “Use Group Cipher” is the negotiated pairwise cipher. For all other AKMs the negotiated pairwise cipher suite does not influence the algorithms used to process EAPOL-Key frames.

|  |  |  |
| --- | --- | --- |
|  | * Table 11-9 Integrity and key-wrap algorithms
 |  |
| AKM | Integrity algorithm | KCK\_bits | Size of MIC | Key-wrap algorithm | KEK\_bits |
| Deprecated | HMAC-MD5 | 128 | 16 | ARC4 | 128 |
| 00-0F-AC:1 | HMAC-SHA1-128 | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:2  | HMAC-SHA1-128 | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:3 | AES-128-CMAC | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:4 | AES-128-CMAC | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:5 | AES-128-CMAC | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:6 | AES-128-CMAC | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:<ANA-2> | HMAC-SHA256 | 128 | 16 | NIST AES Key Wrap | 128 |
| 00-0F-AC:<ANA-3> | HMAC-SHA384 | 192 | 24 | NIST AES Key Wrap | 256 |

**References:**

* Special Publication 800-56A, National Institute of Standards and Technology
* Suite B Implementer’s Guide to NIST SP 800-56A, National Security Agency (USA)