IEEE P802.11
Wireless LANs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |
| --- |
| LB270 KCK clarification in 12.4 |
| Date: 2023-1-24 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Po-Kai Huang | Intel  |  |  | po-kai.huang@intel.com |
| Ido Ouzieli | Intel |  |  |  |
| Ilan Peer | Intel |  |  |  |
| Mohammad Alam | Microsoft |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

 |

Abstract

This submission proposes resolutions for the following comments from comment collection on P802.11-REVme D2.0:

3742

**Revision History:**

R0: Initial version.

R1: Revision based on offline feedback.

# CID 3742

|  |  |  |
| --- | --- | --- |
| **CID****Clause****Page.Line** | **Comment** | **Proposed Change** |
| 3742 | KCK is used extensively in 12.4. However, for the context in 12.4, KCK should be SAE-KCK. (See page 2822 line 41). Givent that KCK is also used in 4-way. To avoid confusion, we should replace KCK in 12.4 as SAE KCK. | Go through all instances of KCK in 12.4 and change KCK to SAE-KCK after confirming the context is correct. Commenter is willing to submit contribution for the task. |

## Discussion:

KCK has been used in various context like 4-way handshake, group key handshake, SAE, TDLS, etc. This creates confusion on which KCK that the spec text in 12.4 is referred to. SAE KCK has been used in 12.4 and is a much better name for the context in 12.4. Propose to change KCK to SAE-KCK in 12.4. Note that TPK-KCK is used in the context of TDLS.

## Proposed Resolution: CID 3742

**REVISED**

**Instruction to TGme Editor:**

Implement the proposed text updates for CID 3742 in 11-23/0154r1

## Proposed Text Update: CID 3742

*Instruction to TGme Editor: Update REVme D2.0 12.4 as shown below (track change on).*

* **Authentication using a password**
* **Processing of a peer’s SAE Commit message**

If the peer’s SAE Commit message contains a (#2168)password identifier (PWE), the value of that identifier shall be used in construction of the (#2168)PWE for this exchange. If a password identifier is present in the peer’s SAE Commit message and there is no password with the given identifier a STA shall fail authentication.

If the peer’s SAE Commit message contains a Rejected Groups element, the list of rejected groups shall be checked to ensure that all of the groups in the list are groups that would be rejected. If any groups in the list would not be rejected then processing of the SAE Commit message terminates and the STA shall reject the peer’s authentication. While the rejected groups are appended to the Rejected Groups element as they are rejected (see 12.4.7.3 (Encoding and decoding of SAE Commit messages)) there is no inherent order to the groups in the list. The order in which they are sent and received shall be retained when deriving keys.

(M67)If the state of the SAE finite state machine is *Committed* (see 12.4.8.2.2 (Protocol instance states)) and the SAE Commit message that has been sent by the SAE finite state machine to transition into *Committed* state includes an AKM Suite Selector element, the authentication shall fail if either of the following conditions is true:

* the peer’s SAE Commit message does not contain an AKM Suite Selector element
* the peer’s SAE Commit message contains an AKM Suite Selector element and the AKM Suite Selector element does not indicate the same AKM

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

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

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

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

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

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

The entropy of *k* shall then be extracted using H to produce *keyseed*. The key derivation function from 12.7.1.6.2 (Key derivation function (KDF)) shall then be used with the hash algorithm identified for H() (see 12.4.2 (Assumptions on SAE)) to derive a SAE key confirmation key, SAE-KCK, and a pairwise master key, PMK, from *keyseed*.

(M67)The intended AKM for the purpose of PMK and SAE-KCK size determination (see below) is determined as follows:

* If an AKM Suite Selector element is not included in the SAE Commit message from the peer and the state of the SAE finite state machine is *Nothing* (see 12.4.8.2.2 (Protocol instance states)), then 00-0F-AC:8 or 00-0F-AC:9 shall be the intended AKM.
* If the state of the SAE finite state machine is *Committed* (see 12.4.8.2.2 (Protocol instance states)) and the SAE Commit message that has been sent by the SAE finite state machine to transition into *Committed* state does not include an AKM Suite Selector element, then 00-0F-AC:8 or 00-0F-AC:9 shall be the intended AKM.
* If an AKM Suite Selector element that indicates AKM 00-0F-AC:24 or AKM 00-0F-AC:25 is included in the SAE Commit message from the peer and the state of the SAE finite state machine is *Nothing* (see 12.4.8.2.2 (Protocol instance states)), then the indicated AKM shall be the intended AKM.
* If the state of the SAE finite state machine is *Committed* (see 12.4.8.2.2 (Protocol instance states)) and the SAE Commit message that has been sent by the SAE finite state machine to transition into *Committed* state includes an AKM Suite Selector element that indicates AKM 00-0F-AC:24 or AKM 00-0F-AC:25, then the indicated AKM shall be the intended AKM.

(M67)If the intended AKM is (M21)00-0F-AC:8 or 00-0F-AC:9 and the looping method of PWE generation (see 12.4.4.2.2 (Generation of the password element with ECC groups by looping) and 12.4.4.3.2 (Generation of the password element with FFC groups by looping)), both the SAE-KCK and PMK shall be 256 bits in length. (M67)If the intended AKM is 00-0F-AC:8 or 00-0F-AC:9 and the hash-to-element method(#344) of PWE generation (see 12.4.4.2.3 (Hash-to-element(#331) generation of the password element with ECC groups) and 12.4.4.3.3 (Direct generation of the password element with FFC groups)), the SAE-KCK shall have(M67) the length of the digest generated by H() and the PMK shall be 256 bits in length (M21)(see 12.7.1.3 (Pairwise key hierarchy)). (M67)If the intended AKM is 00-0F-AC:24 or 00-0F-AC:25, the hash-to-element method(#344) of PWE generation (see 12.4.4.2.3 (Hash-to-element(#331) generation of the password element with ECC groups) and 12.4.4.3.3 (Direct generation of the password element with FFC groups)) shall be used, the SAE-KCK and the PMK shall have the length of the digest generated by H().(M67) Use of other AKMs with the hash-to-element method(#344) will require definition of the length of the PMK. If both SAE Commit messages indicated a status code of SAE\_HASH\_TO\_ELEMENT, a salt consisting of the concatenation of the rejected groups from each peer’s Rejected Groups element shall be passed to the KDF; those of the peer with the highest MAC address go first (if only one sent a Rejected Groups element then the salt will consist of that list). If neither peer sent a Rejected Groups element or the status code was not SAE\_HASH\_TO\_ELEMENT, the salt shall consist of a series of octets of the value zero whose length equals the length of the digest of the hash function used to instantiate H().

*keyseed* = H(*salt*, *k*)

*context* = (*commit-scalar* + *peer-commit-scalar*) mod *r*

*Length* = *Q* + *PMK\_bits*(M21)

(#478)*sae*\_*kck\_and\_pmk* = KDF-*Hash*-*Length*(*keyseed*, “SAE KCK and PMK”, *context*)

*SAE-KCK* = L(*sae*\_*kck\_and\_pmk*, 0, *Q*)

*PMK* = L(*sae*\_*kck\_and\_pmk*, *Q*, *PMK\_bits*)(M21)

where

*salt* is either a series of 0 octets or a list of rejected groups (see 12.4.7.3 (Encoding and decoding of SAE Commit messages))

(#478)KDF-*Hash*-*Length* is the key derivation function defined in 12.7.1.6.2 (Key derivation function (KDF)) using the hash algorithm defined for H()

*Q* is the length of the digest of the H(), the hash function used

*context* is treated as an integer and converted into an octet string of length *m* such that 28m > *r* according to 12.4.7.2.2 (Integer to octet string conversion)

*PMK\_bits*  is the length of the PMK in bits, as defined in 12.7.1.3 (Pairwise key hierarchy)(M21)

The PMK identifier is defined as follows:

PMKID = L(*context*, 0, 128)

* **Construction of an SAE Confirm message**

A peer generates a confirmation, *confirm*, and inserts it into an SAE Confirm message by passing the SAE-KCK, the current value of the *send-confirm* counter (see 9.4.1.37 (Send-Confirm field)), the scalar and element from the sent SAE Commit message, and the scalar and element from the received SAE Commit message to the confirmation function CN.

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

***PEER-COMMIT-ELEMENT***)

The *send-confirm* counter shall be encoded according to 9.2.2 (Conventions). The elements and scalars shall be in the format they were encoded in when transmitted in an SAE Commit message as described in 12.4.7.3 (Encoding and decoding of SAE Commit messages). The message shall be transmitted to the peer as described in 12.4.7 (Framing of SAE).

* **Processing of a peer’s SAE Confirm message**

Upon receipt of a peer’s SAE Confirm message a *verifier* is computed, which is the expected value of the peer’s confirmation, *peer-confirm*, extracted from the received an SAE Confirm message. The *verifier* is computed by passing the SAE-KCK, the peer’s send-confirm counter from the received an SAE Confirm message (see 9.4.1.37 (Send-Confirm field)), the scalar and element from the received SAE Commit message, and scalar and element from the sent SAE Commit message to the confirmation function CN.

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

The *peer-send-confirm* shall be encoded according to 9.2.2 (Conventions). The elements and scalars shall be in the format they were encoded in when transmitted in an SAE Commit message as described in 12.4.7.3 (Encoding and decoding of SAE Commit messages). If the *verifier* differs from the *peer*-*confirm,* verification of the peer’s SAE Confirm message shall fail.