IEEE P802.11
Wireless LANs

|  |
| --- |
| Clarifications of SAE |
| Date: 2013-01-11 |
| 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

An implementer of the SAE protocol found parts of the SAE protocol description ambiguous and confusing. This submission clears up the problematic text.

***Instruct the editor to modify section 11.3.2 as indicated:***

**11.3.2 Assumptions on SAE**

When used with AKMs 00-0F-AC:8 or 00-0F-AC:9 from Table 8-102 (AKM suite selectors), CN is instantiated as a function that takes a key, a counter, and a sequence of data. Each piece of data is converted to an octet string and concatenated together before being concatenated to the counter and passed, along with the key, to HMAC-SHA256:

CN(key, counter, X, Y, Z, …) = HMAC-SHA256(key, counter || D2OS(X) || D2OS(Y) || D2OS(Z) || …)

where D2OS() represents the data to octet string conversion functions in 11.3.7.2 (Data type conversion).

***Instruct the editor to modify 11.3.5.5 and 11.3.5.6 as indicated:***

**11.3.5.5 Construction of a Confirm Message**

A peer generates a Confirm Message by passing the KCK, the current value of the *send-confirm* counter (see 8.4.1.37 (Send-Confirm field)), the scalar and element from the sent Commit Message, and the scalar and element from the received Commit Message to the confirmation function CN.

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

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

The *send-confirm* counter shall be in the format specified in section 8.2.2 in the order in which it is transmitted over the air. The message shall be transmitted to the peer as described in 11.3.7 (Framing of SAE).

**11.3.5.5 Processing of a peer’s Confirm Message**

Upon receipt of a peer’s Confirm Message a *verifier* is computed, which is the expected value of the peer’s confirmation, *peer-confirm*, extracted from the received Confirm Message. The *verifier* is computed by passing the KCK, the peer’s send-confirm counter from the received Confirm Message (see 8.4.1.37 (Send-Confirm field)), the scalar and element from the received Commit Message, and scalar and element from the sent Commit Message to the confirmation function CN.

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

*commit-scalar*, ***COMMIT-ELEMENT***)

The *peer-send-confirm* shall be in the format in section 8.2.2, as extracted out of the received frame. If the *verifier* equals *peer-confirm*, the STA shall accept the peer’s authentication and set the lifetime of the PMK to the value dot11RSNAConfigPMKLifetime. If the *verifier* differs from the *peer-confirm*, the STA shall reject the peer’s authentication and destroy the PMK.

***Instruct the editor to modify section 11.3.8.6.1 as indicated:***

**11.3.8.6.1 Parent process behavior**

Upon receipt of a Commit Message, the parent process checks whether a protocol instance for the peer MAC address exists in the database. If one does, and it is in either *Committed* state or *Confirmed* state the frame shall be passed to the protocol instance. If one does and it is in *Accepted* state, the scalar in the received frame is checked against the *peer-scalar* used in authentication of the existing protocol instance (in *Accepted*state). If it is identical, the frame shall be dropped. If not, the parent process checks the value of *Open* . If *Open* is greater or equal to than dot11RSNASAEAntiCloggingThreshold, the parent process shall check for the presence of an Anti-Clogging Token. If an Anti-Clogging Token exists and is correct, the parent process shall create a protocol instance. If the Anti-Clogging Token is incorrect, the frame shall be silently discarded. If *Open* is greater than or equal to dot11RSNASAEAntiCloggingThreshold and there is no Anti-Clogging Token in the received frame, the parent process shall construct a response as an (#100) Authentication frame with Authentication sequence number one (1), Status code 76, and the body of the frame consisting of an Anti-Clogging Token (see 11.3.6 (Anti-clogging tokens)). If *Open* is less than dot11RSNASAEAntiCloggingThreshold, the parent process shall create a protocol instance and the frame shall be sent to the protocol instance as a *Com* event.

**References:**