IEEE P802.11
Wireless LANs

|  |
| --- |
| Comments on SAE State Machine |
| Date: 2017-12-20 |
| Author(s): |
| Name | Affiliation | Address | Phone | email |
| Nehru Bhandaru | Broadcom Ltd. | 250 Innovation Drive, San Jose CA 95134 | +1 408 922 5924 | nehru.bhandaru@broadcom.com |
| Philippe Moutarlier | Broadcom Ltd. | 250 Innovation Drive, San Jose CA 95134 |  | philippe.moutarlier@broadcom.com |
| Dan Harkins | HPE | 3333 Scott boulevardSanta Clara, CaliforniaUnited States of America |  |  |

Abstract

SAE provides robust authentication with a password in an 802.11 RSN that results in the establishment of a PMK (section 12.4 of [1]). It includes the SAE state machine and a description of the processing upon receving various state machine events when a frame from a peer is received or other local management action. This document draws attention to few aspects related to SAE state machine as currently described.

* Indicator *BadAuth* used in transitions from *Confirmed* and Accepted states is not described, whereas indicator *BadConf* (section 12.4.8.5.2) is described but not used.
* A bad *Confirm* frame is discarded in *Accepted* state vs. in the *Confirmed* state the PMK is deleted which results in a teardown of the state machine making it more susceptible denial of service attacks.
* Upon receiving a *Confirm* message in *Committed* state, canceling the timer and incrementing the *Sync* counter makes it more susceptible to denial of service attacks.

Minor changes to the specificiation [1] are also proposed to improve the above aspects.

**Discussion – *BadAuth* vs*. BadConf* indicators in *Confirmed* state**

SAE provides robust authentication with a password in an 80211 RSN and establishes PMK that is confirmed later along using derived keys KCK, KEK, TK via 4-way handshake or otherwise to provide data confidentiality and integrity services.

The SAE protocol is described in 802.11 specification [1] using a state machine (Figure 12-4 SAE finite state machine) and a normative behaviour description that follows (section 12.4.8.6 Behavior of state machine)

The state machine includes transitions in *Confirmed* and Accepted states upon *BadAuth* indicator or its negation !*BadAuth,* but *BadAuth* does not seem to defined otherwise in the specification. Indicators are described in section ‘12.4.8.5.2 Protocol instance variables’. The included indicators contain a *BadConf* indicator that denotes that the ‘contents of a confirm frame were incorrect’ but *Badconf* is not used in the state machine diagram.

It appears that in the described behavior in section ’12.4.8.6.5 Protocol instance behaviour – Confirmed State… ‘Upon receipt of a *Con* event…’ *BadConf* (or is it *BadAuth*) indicator is a result of verifier mismatch described in section ‘12.4.5.6 Processing of a peer’s SAE Confirm message’ which results in the deletion of the PMK and subsequently the state machine transition to *Nothing* state. The SAE protocol instance is then deleted by the parent process of the SAE protocol instance.

This may be an editorial issue and can simply be addressed by renaming *BadConf* to *BadAuth* in instance variable description as described in Option 1 below. However, there is an additional consideration, as discussed in the next section.

**Discussion – *Denial of Service from a mismatched verifier in Confirmed state***

If verification mismatch occurs for an SAE *Confirm* frame in the *Accepted* state, it is discarded. If the verification succeeds, the *Sync* counter is incremented.

A denial of service can be attempted by an attacker blocking the *Confirm* frame(s), possibly selectively, in both directions and then replaying one of the frames with an invalid verifier. For example, if the first *Confirm* from a peer was allowed, but the response *Confirm* from the receiving STA was blocked, the second legitimate *Confirm* from the peer can be blocked with a replay with an invalid verifier forwarded. If such a frame is processed by a STA that is in the *Accepted* state, according to section ‘12.4.5.6 Processing of a peer’s SAE Confirm message’, the PMK would be deleted. One would presume this implies all of the security state derived from the PMK is also deleted, but that behaviour is inconsistent with the text in ’12.4.8.6.6 Protocol instance behavior – Accepted state’ … “If the verification fails, the received frame shall be silently discarded”. 12.4.5.6 should just say that the verification fails and remain in the current state.

Similarly, if verification mismatch occurs from an SAE *Confirm* frame in the *Confirmed* state, the PMK is deleted.

Thus a denial of service threat exists from an MITM attack that may mangle the verifier such that verification fails on the receiver.

It seems it is an overkill to delete the PMK when the verification fails tearing down the SAE protocol instance. A better alternative upon verification failure would be to not cancel the t0 timer in *Confirmed* state and do nothing in the *Accepted* state. When the threshold condition *big(sync)* is satisfied, the corresponding state transition would change *Accepted* or *Confirmed* states to *Nothing* tearing down the state machine after giving the peer sufficient time to complete the authentication.

An MITM attack would still be possible with above suggested change, but that would force an MITM attacker to be actively present for much longer on the channel and would also reduce the overhead of starting a new protocol instance for recovery in the presence of an attack.

Note to editor: Instructions are relative to TGmd Draft 0.5 [1]. *pnnnn.mm* refers to page nnnn, line mm – nnnn refers to the page number printed on that page in the pdf

***Instruct the editor to replace the ‘***(Con, BadAuth)/Del’ ***transition in the state machine (Figure 12-4,*** *p2370.1****) out of the Confirmed state with ‘***(Con, BadAuth, !big(sync)/-***’.***

***Instruct the editor to modify instance variable description in section* ‘12.4.8.5.2 Protocol instance variables’ *as follows***

*p2372.40*

— *~~BadConf~~ BadAuth* —The contents of a confirm frame were incorrect. Authentication rejected due to verifier mismatch while processing SAE confirm message.

***Instruct the editor to modify section* ‘12.4.5.6 Processing of a peer’s SAE Confirm message’ *as follows***

***…***

*p2366.33*

If the verifier differs from the peer-confirm, verification of the peer’s SAE Confirm message shall fail..

***Instruct the editor to modify section* ‘12.4.8.6.5 Protocol instance behavior – Confirmed state’ *as follows***

*p2376.5*

Upon receipt of a *Con* event, the SAE Confirm message

shall be processed according to 12.4.5.6 (Processing of a peer’s SAE Confirm message). If processing is unsuccessful and the SAE Confirm message is not verified, protocol instance shall remain in *Confirmed* state. If processing is

successful and the SAE Confirm message has been verified, the Rc variable shall be set to the send-confirm portion of the frame, Sc shall be set to the value 2^16 – 1, the t1 (key expiration) timer shall be set, the t0 (retransmission) timer shall be cancelled, and the protocol instance shall transition to Accepted  state.

**Discussion – *Denial of Service from a Sync counter increment on Con event in Committed state***

Upon receiving a *Confirm* message in *Committed* state, canceling the timer and incrementing the Sync counter makes it more susceptible to denial of service attacks. An attacker can simply time and transmit Confirm messages and create a denial of service.

A *Con* event should not increment the *Sync* counter, but should simply transmit the last SAE Commit message to the peer. It probably should not reset the t0 (retransmission) timer – if the STA does not receive a Commit message from the peer before big(sync) event, the instance will terminate per timer processing described for the *Committed* state.

Another minor editorial issue in the description of *Committed* state in in section ‘12.4.8.2.2 Protocol instance states’ seems to be the conjunction ‘and’ ‘In the *Committed* state, the finite state machine has sent an SAE Commit message and is awaiting an SAE Commit message and an SAE Confirm message from the peer’.

The state machine can receive either of the messages – so, an ‘or’ might be more appropriate. What it receives can be different than what it is expecting. Perhaps it is expecting both of the messages in order to finish SAE. However, receiving a SAE Confirm message in the Committed state is rather an exception than a norm. We suggest clarifying the ‘and’ condition referring to the Confirm message in Committed state. Later text in ‘12’4.8.6.4 Protocol instance behavior – Committed state’ already specifies how the state machine needs to handle the message.

***Instruct the editor to modify the* action *from the ‘***Con, !big(Sync)…’ ***transition in the state machine (Figure 12-4, p2370.1) out of the Committed state:***

 (Con,!big(Sync)/( 1(0), set(t0))

***Instruct the editor to modify section* ‘12.4.8.6.4 Protocol instance behavior – Committed state’ *as follows***

*p2375.42*

Upon receipt of a *Con* event the protocol instance

checks the value of *Sync*. If it is greater than dot11RSNASAESync, the protocol instance shall send a Del

 event to the parent process and transition back to *Nothing* state. If *Sync* is not greater than

dot11RSNASAESync, the protocol instance shall and retransmit the last SAE Commit message

sent to the peer

***Instruct the editor to modify section* ‘12.4.8.2.2 Protocol instance states’ *as follows***

…

*p2370.53*

— Committed —In the *Committed* state, the finite state machine has sent an SAE Commit message and

needs both SAE Commit and SAE Confirm messages from the peer.

**References:**

[1] IEEE P802.11-REVmdTM/D0.5, December 2017

[2] IEEE Std 802.11-2016