Internet Engineering Task Force
INTERNET-DRAFT
Vollbrecht & Petroni
Vollbrecht Consulting
March 3, 2003

State Machines for EAP Peer and Authenticators

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http:// www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 1, 2003.

Copyright Notice

Abstract

The current specification for the Extensible Authentication Protocol (EAP) does not include a state machine. The EAP working group would like to include a state machine for peer and authenticator at the same time it releases a revised EAP RFC. This might be done as a separate RFC describing the state machine, or it might be incorporated in the revised EAP RFC.

This memo describes a state machine based on an EAP "Switch" model. This model includes an additional set of events and actions for the interaction between the EAP Switch and EAP methods. The State Machine and associated model are informative only. They are provided to clarify meaning of text in the EAP RFC. Implementations may achieve the same results using different methods.

A brief description of the EAP "Switch" model is given in the next section. The EAP design working group has discussed the EAP Switch model as a way to describe EAP protocol interactions.

This document is based on discussions held on EAP Design Team conference calls. The State machine has also been presented at 802.1x and 802.11f meetings, and some changes have been made to it based on feedback from these meetings. Corresponding changes have been proposed for the state machines for 802.1x and 802.11.

This document is primarily authored by John Vollbrecht and Nick Petroni, but is based on the discussion and feedback of the EAP State Machine Design Team. Furthermore, this document represents the combined work of two previous Internet Drafts by Bryan Payne, Chuk Seng, John Vollbrecht, and Nick Petroni.

This document is still a work in progress. The current EAP document has not been checked to ensure that the state machine and the document agree in all particulars. We expect this to be done over the next few weeks.

1 Specification of Requirements

In this document, several words are used to signify the requirements of the specification. These words are often capitalized. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 The EAP Switch Model

This document offers a proposed state machine for RFC 2284bis (EAP). There is a state machine for the peer and one for the authenticator. Accompanying each state machine diagram is a description of the notation used. Whenever possible, the same notation has been used in both the peer and authenticator state machines.

An EAP authentication consists of one or more EAP methods in sequence followed by an EAP Success or EAP Failure sent from the Authenticator to the peer. The EAP Switches control negotiation of EAP methods and sequences of methods.

```
Peer              Peer  |  Authenticator       Auth
Method                  |                      Method
      \                 |                    /
        \               |                  /
         Peer           |             Auth
         EAP    <-----|----------> EAP
         Switch         |             Switch
```

Figure 1: EAP Switch Model

At both the peer and authenticator one or more EAP method exists. The EAP switches select which methods each is willing to use, and negotiate between themselves to pick a method or sequence of methods.

Note that the methods may also have state machines, but the details of these are out of scope for this paper.

This document describes State Machines for both the peer EAP switch and for the Auth EAP Switch.

3 EAP State Machines

3.1 Notational conventions used in state diagrams

The following state diagrams have been completed following the conventions specified in [IEEE.802-1X.2001], section 8.5.1. That section is reprinted here for completeness.

State diagrams are used to represent the operation of a function as a group of connected, mutually exclusive states. Only one state of a function can be active at any given time. Each state is represented in the state diagram as a rectangular box, divided into two parts by a horizontal line. The upper part contains the state identifier, written in uppercase letters. The lower part contains any procedures that are executed on entry to the state.

All permissible transitions between states are represented by arrows, the arrowhead denoting the direction of the possible transition. Labels attached to arrows denote the condition(s) that must be met in order for the transition to take place. A transition that is global in nature (i.e., a transition that occurs from any of the possible states if the condition attached to the arrow is met) is denoted by an open arrow; i.e., no specific state is identified as the origin of the transition.

On entry to a state, the procedures defined for the state (if any) are executed exactly once, in the order that they appear on the page. Each action is deemed to be atomic; i.e., execution of a procedure completes before the next sequential procedure starts to execute. No procedures execute outside of a state block. On completion of all of the procedures within a state, all exit conditions for the state (including all conditions associated with global transitions) are evaluated continuously until such a time as one of the conditions is met. All exit conditions are regarded as Boolean expressions that evaluate to True or False; if a condition evaluates to True, then the condition is met. When the condition associated with a global transition is met, it supersedes all other exit conditions, including UCT. The label UCT denotes an unconditional transition (i.e., UCT always evaluates to True). The label ELSE denotes a transition that occurs if none of the other conditions for transitions from the state are met (i.e., ELSE evaluates to True if all other possible exit conditions from the state evaluate to False).

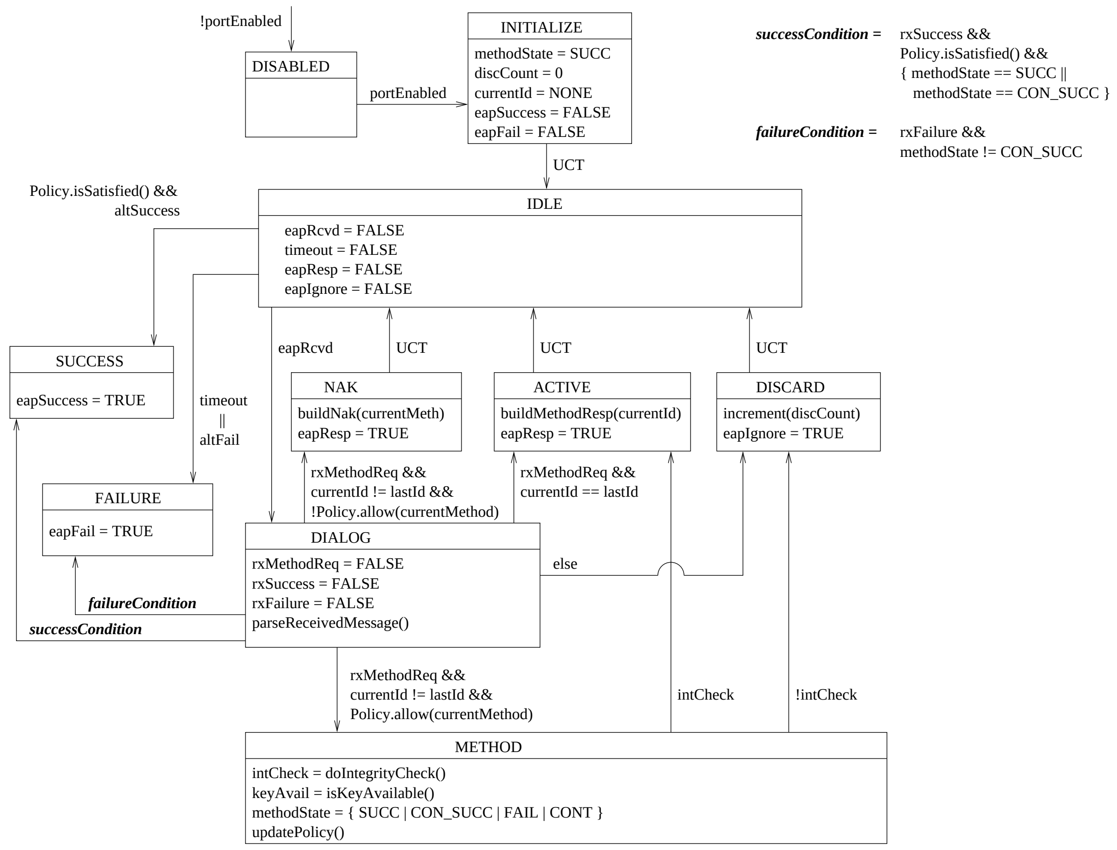
A variable that is set to a particular value in a state block retains this value until a subsequent state block executes a procedure that modifies the value.

Where it is necessary to segment a state machine description across more than one diagram, a transition between two states that appear on different diagrams is represented by an exit arrow drawn with dashed lines, plus a reference to the diagram that contains the destination state. Similarly, dashed arrows and a dashed state box are used on the destination diagram to show the transition to the destination state. In a state machine that has been segmented in this way, any global transitions that can cause entry to states defined in one of the diagrams are deemed to be potential exit conditions for all of the states of the state machine, regardless of which diagram the state boxes appear in.

Should a conflict exist between the interpretation of a state diagram and either the corresponding global transition tables or the textual description associated with the state machine, the state diagram takes precedence.

3.2 Peer State Machine

The following is a diagram of the EAP Peer state machine. Also included is an explanation of the primitives and procedures referenced in the diagram, as well as a clarification of notation.



3.2.1 Variables

- portEnabled.

This variable indicates the lower layer is prepared to allow EAP traffic over it. As long as portEnabled remains TRUE, the EAP conversation can continue. If at any point it becomes FALSE, EAP must transition to the DISABLED state. This variable is controlled entirely by the lower layer and is not set by EAP or its methods.

- methodState.

  Indicates the state of the currently active EAP Method. Possible values are as follows:

  - CONT- The method is "continuing". That is, the method has not come to a finish and therefore has neither failed nor succeeded.
  - SUCC- The method has "succeeded". The method conversation is over and it is believed to have been properly authenticated.
  - CON_SUCC- The method has "confirmed success". Same as SUCC, but the peer has received a protected success to indicate that the conversation should not fail.
  - FAIL- The method has "failed". The peer has not met its expectations for a successful authentication and the method conversation is over.

- discCount.

  The number of EAP messages that have been discarded due to failed integrity checks or messages received in invalid states.

- currentId.

  The EAP identifier value associated with the currently outstanding request. Whenever a valid request is received, the identifier associated with that request is assigned to currentId.

- eapRcvd.

  Indication from the lower layer that a packet has arrived and should be processed by the EAP layer. This variable acts as part of a handshake and is only set TRUE by the lower layer and FALSE by EAP.

- eapResp.

  Indication to the lower layer that an EAP packet (response) is ready to be sent. This variable can never be true at the same time as eapIgnore. This completes the handshake initiated by eapRcvd.

- eapIgnore.

  Indication to the lower layer that the EAP request has been processed and that there is no response to send. This variable can never be true at the same time as eapResp. This completes the handshake initiated by eapRcvd.

- eapSuccess.

  Indication to the lower layer that the EAP conversation has completed successfully. Should only be set to TRUE by reaching the SUCCESS state (see below for SUCCESS conditions).

- eapFail.

  Indication to the lower layer that the EAP conversation has completed and failed. Should only be set to TRUE by reaching the FAILURE state (see below for FAILURE conditions).

- altSuccess.

  Indication that an "alternative success" has been received. Such successes are not EAP messages, but are defined on an implementation-specific basis.

- altFail.

  Indication that an "alternative failure" has been received. Such failures are not EAP messages, but are defined on an implementation-specific basis.

- timeout.

  Indication that an internal timer has completed while waiting for another packet from the authenticator. The result is a transition to the FAILURE state.

- lastId.

  The EAP identifier value associated with the last response sent (and therefore the last request received). When a request is received with an identifier equal to the previous identifier, the same response is immediately sent (without going back to the method for processing). This is because if the request were not identical a new identifier would be used by the authenticator.

- rxMethodReq.

  Internal variable to EAP indicating that the incoming packet has been parsed and it is, in fact, a valid method request. If policy allow that method to be performed, the indication causes a transition to the METHOD state. This variable is set by the parseReceivedMessage() function.

- rxSuccess.

  Internal variable to EAP indicating that the incoming packet has been parsed and that it is an EAP success packet. If the peer's policy is satisfied, this signal will cause a transition to SUCCESS. This variable is set by the parseReceivedMessage() function.

- rxFail.

  Internal variable to EAP indicating that the incoming packet has been parsed and that it is an EAP failure packet. Unless the method state is "CON_SUCC", this will always cause a transition to the FAILURE state. This method is set by the parseReceivedMessage() function.

- intCheck.

  Internal variable to EAP indicating the success or failure of the method's integrity check on the incoming packet. If no such integrity check exists, or if the method cannot determine the integrity for a specific packet (as with multiple-message integrity checks), this variable MUST be set to TRUE.

- keyAvail.

  Internal variable to EAP indicating that key material is now available for use by EAP methods or the lower layer.

3.2.2 Procedures

- buildNak()

  Function to produce a Nak for the current method being proposed by the authenticator. Results in the setting of eapRsp since a packet will be sent.

- buildMethodResp()

  Function to produce the method's response to a method request. This should only be called in the METHOD state.

- increment()

  Mathematical function to increase a numerical variable by 1.

- parseReceivedMessage()

  Function to determine the type and identifier of an EAP message. When appropriate, this function will update the currentId variable as well as set at MOST ONE of rxMethodReq, rxSuccess, or rxFail to TRUE.

- doIntegrityCheck()

  Function that performs the method-specific integrity check for the current EAP method request and returns TRUE if the message is valid, FALSE otherwise.

- updatePolicy()

  Function whereby the EAP method updates variables associated with Policy satisfaction. Such variables may include, but are not limited to mutual authentication, the completion of a particular method, and key availability

- Policy.allow()

  Function to test whether the method proposed by an EAP request should be performed based on the peer's policy.

- Policy.isSatisfied()

  Function to determine whether or not the peer's SUCCESS policy has been met. A transition to the SUCCESS state SHOULD NOT occur without a satisfied policy.

- isKeyAvailable()

  Function which indicates whether or not cryptographic key material is available for use. Returns TRUE if so, FALSE otherwise.

3.2.3 States

- DISABLED

  This state is reached anytime service from the lower layer is interrupted or unavailable. Immediate transition to INITIALIZE occurs when the port become enabled.

- INITIALIZE

  This state is used to set initial conditions for the EAP conversation. Discard counters are reset, current identifier values erased, and success and failure indications are set to FALSE. This state is only reached once and before any EAP messages are sent in either direction.

- IDLE

  The state in which the peer waits for a request from the authenticator. eapRcvd is set to FALSE to await indication from the lower layer and the timer is reset. This state is reached after every request/response pair until sufficient SUCCESS or FAILURE conditions are met.

- DIALOG

  An EAP message has been received by the lower layer and is to be processed in this state. Internal message type indications are reset and the incoming message is parsed. If a method request is received, there are 3 possible transitions. If the request was a retransmission, the current response is resent. Otherwise, if the proposed method is allowed by the peer's policy then the request is passed to the corresponding method and a transition is made to the METHOD state. If the proposed method is disallowed, a Nak is sent.

- NAK

  The state reached by receipt of a method request for a disallowed method. A Nak is built based on the peer's policy and preferred method and the Nak is passed to the lower layer.

- ACTIVE

  The state reached when a valid EAP request has been received and an EAP response formulated. In this state the message is built and passed to the lower layer.

- DISCARD

  The state reached by the receipt of an invalid EAP packet. A failed method integrity check or any EAP packet that is received at an improper time will result in the peer discarding the packet by setting eapIgnore. The discard count is incremented in this state.

- METHOD

  The state representing action by an EAP method. This state is reached upon receipt of an allowable EAP request with a new identifier value. Each method must verify the integrity of the request, determine if keying material is available as a result of the received packet or ensuing response, update the methodState variable, and update variables related to Peer policy.

- SUCCESS

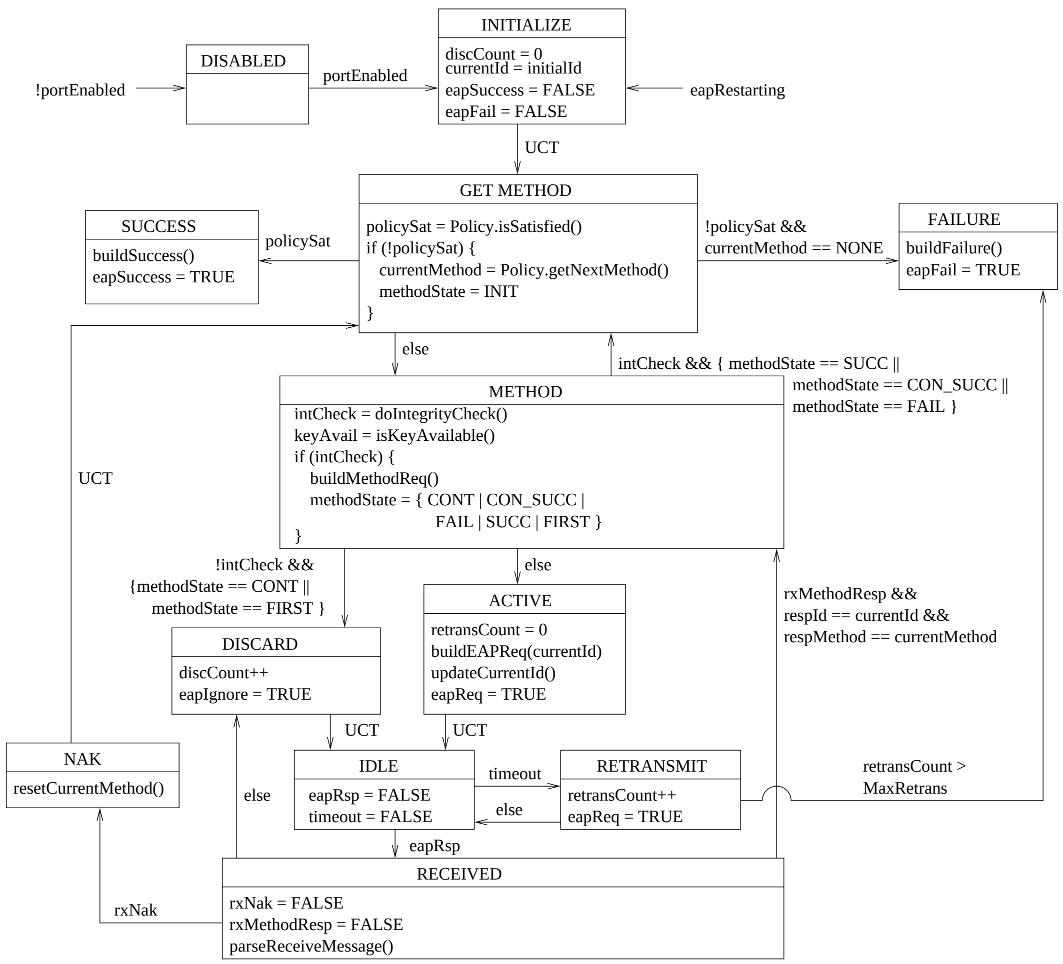
  The state reached upon successful authentication of the peer. SUCCESS can only be reached by receipt of a success packet or an alternative success indication, as well as satisfaction of the Peer's policy.

- FAILURE

  The state reached whenever the EAP conversation cannot continue any longer and has failed to authenticate the Peer or failed to satisfy the Peer's policy.

3.3 Authenticator State Machine

The following is a diagram of the EAP Authenticator state machine. Also included is an explanation of the primitives and procedures referenced in the diagram, as well as a clarification of notation.



3.3.1 Variables

- portEnabled

  This variable indicates the lower layer is prepared to allow EAP traffic over it. As long as portEnabled remains TRUE, the EAP conversation can continue. If at any point it becomes FALSE, EAP must transition to the DISABLED state. This variable is controlled entirely by the lower layer and is not set by EAP or its methods.

- discCount

  The number of EAP messages that have been discarded due to failed integrity checks or messages received in invalid states.

- currentId

  The EAP identifier value associated with the currently outstanding request. Whenever a valid request is created, the identifier associated with that request is assigned to currentId. The authenticator has no need for for a lastId value because the peer does not retransmit.

- eapRestarting

Variable used for communication by the lower layer to indicate to EAP that something has changed and that authentication should start over. This is a shortcut to disabling and then re-enabling the port, which may have additional overhead. EAP should not set this signal to TRUE, as it is set by the lower layer.

- policySat

  An internal variable that represents the latest outcome of a query to policy. The authenticator cannot enter the success state until successfully completing EAP methods consistent with a configurable policy. Since the authenticator controls the EAP conversation, this check can be done after each method has completed in the "GET METHOD" state.

- currentMethod

  The current method being performed by the authenticator. Only one method should be active at a time and, upon changing methods, the state of the previous method shall be reset or is otherwise undefined. That is, the authenticator cannot carry on multiple method conversations at once. A decision to change methods indicates completion of the former method and initiation of the new method.

- methodState

  Indicates the state of the currently active EAP Method. Possible values are as follows:

  – CONT- The method is "continuing". That is, the method has not come to a finish and therefore has neither failed nor succeeded.

  – SUCC- The method has "succeeded". The method conversation is over and it has successfully authenticated the peer

  – CON_SUCC- The method has "confirmed success". Same as SUCC, but the peer has been sent a protected success to indicate that the conversation should not fail. In terms of transitions within the authenticator, SUCC and CON_SUCC are no different, but they are maintained on the authenticator for accounting.

  – FAIL- The method has "failed". The authenticator has not met its expectations for a successful authentication and the method conversation is over.

- eapSuccess

  Indication to the lower layer that the EAP conversation has completed successfully. Should only be set to TRUE by reaching the SUCCESS state (see below for SUCCESS conditions).

- eapFail

  Indication to the lower layer that the EAP conversation has completed and failed. Should only be set to TRUE by reaching the FAILURE state (see below for FAILURE conditions).

- eapRsp

  Indication from the lower layer that a response packet has arrived and should be processed by the EAP layer. This variable acts as part of a handshake and is only set TRUE by the lower layer and FALSE by EAP.

- eapIgnore

  Indication to the lower layer that the EAP response has been processed and that there is no new request to send. This variable can never be true at the same time as eapReq. This completes the handshake initiated by eapRsp.

- eapReq

  Indication to the lower layer that an EAP packet (request) is ready to be sent. This variable can never be true at the same time as eapIgnore. This completes the handshake initiated by eapRcvd.

- eapSuccess

  Indication to the lower layer that the EAP conversation has completed successfully. Should only be set to TRUE by reaching the SUCCESS state (see below for SUCCESS conditions).

- intCheck

  internal variable to EAP indicating the success or failure of the method's integrity check on the incoming packet. If no such integrity check exists, or if the method cannot determine the integrity for a specific packet (as with multiple-message integrity checks), this variable MUST be set to TRUE.

- retransCount

  The number of times the current request has been retransmitted. This counter gets reset after each newly built response packet and incremented for each pass through the RETRANSMIT state.

- respId

  The EAP identifier associated with the most recently received response packet. EAP has a window size of 1 and therefore can only have a single outstanding request. Therefore, all packets received with respId != currentId are discarded.

- respMethod

  The EAP method type associated with the most recently received response packet. EAP has a window size of 1 and therefore can only have a single outstanding request. Therefore, all packets received with respMethod != currentMethod are discarded.

- rxMethodResp

  Internal variable to EAP indicating that the incoming packet has been parsed and it is, in fact, a valid method response. The packet is then checked for the appropriate method and identifier and passed to the method. This variable is set by the parseReceivedMessage() function.

- rxNak

  Internal variable to EAP indicating that the incoming packet has been parse and is a Nak packet. A transfer occurs to the NAK state where the current method is reset and a transition is made to GET METHOD to determine the next part of the conversation. This variable is set by the parseReceivedMessage() function.

- timeout

  Indication that an internal timer has completed while waiting for a response packet from the peer. Unlike the peer, the authenticator is responsible for retransmissions and therefore a timeout results in either a retransmission or a failure (depending on MaxRetrans).

- keyAvail

  Internal variable to EAP indicating that key material is now available for use by EAP methods or the lower layer.

3.3.2 Constants

- initialId

  The initial value to be used for the EAP Identifier field. This can be any value between 0 and 255.

- MaxRetrans

  The (configurable) maximum number of request retransmissions allowable before failure.

3.3.3 Procedures

- Policy.isSatisfied()

  Function to determine whether or not the authenticator's SUCCESS policy has been met. A transition to the SUCCESS state SHOULD NOT occur without a satisfied policy.

- Policy.getNextMethod()

  Function to determine which method the authenticator should perform next (if any) in a given conversation. While an implementation may perform the same methods for all peers or in the same order, this is not required.

- buildSuccess()

  Function to produce an EAP success message to be sent by the lower layer. This function should only be called from the SUCCESS state.

- buildFailure()

  Function to produce an EAP failure message to be sent by the lower layer. This function should only be called from the FAILURE state.

- buildMethodReq()

  Function to produce a method's request. This occurs during the METHOD state. Once produced, the message is passed to the EAP layer where buildEAPReq() is called to add the current identifier. An implementation can successfully combine these two functions into a single buildMethodReq() without changing the behavior of the authenticator.

- buildEAPReq()

  EAP-layer function to process the method request built by buildMethodReq(). Typically this involves simply choosing the correct identifier value for the EAP packet. As specified above, this function may be combined with buildMethodReq().

- doIntegrityCheck()

  Function that performs the method-specific integrity check for the current EAP method response and returns TRUE if the message is valid, FALSE otherwise.

- isKeyAvailable()

  Function which indicates whether or not cryptographic key material is available for use. Returns TRUE if so, FALSE otherwise.

- updateCurrentId()

  Function to update the currentId variable. This must occur when a new request is built in order to insure proper error checking on subsequent response packet.

- resetCurrentMethod()

  Function to reset the state of a method because of the receipt of a Nak from the peer.

- parseReceivedMessage()

  Function to determine the type, identifier, and method of an EAP message. When appropriate, this function will update the respId and respMethod variables as well as set at MOST ONE of rxMethodRsp or rxNak to TRUE.

3.3.4 States

- DISABLED

  This state is reached anytime service from the lower layer is interrupted or unavailable. Immediate transition to INITIALIZE occurs when the port become enabled.

- INITIALIZE

  This state is used to set initial conditions for the EAP conversation. Discard counters are reset, current identifier value initialized, and success and failure indications are set to FALSE. Unlike the peer, this state may be reached multiple times, but always indicates an entirely new conversation whereby all previous state is lost. This restart occurs when the lower layer sets eapRestarting to TRUE.

- GET METHOD

  This state occurs "between methods" and is the state in which the EAP layer makes a decision about which method to run. Upon successful policy satisfaction, a transition to SUCCESS occurs. Similarly, if no more methods are available and the policy is not satisfied, a transition to FAILURE occurs.

- METHOD

  This state represents the part of the conversation controlled by the EAP method itself. Until a method signals its completion, or a Nak is received, the method maintains control by producing new requests and checking the validity of responses. This method must update the methodState variable and alert the lower layer to key material when it becomes available.

- ACTIVE

  The state reached when a valid EAP request has been formulated. In this state the message is built and passed to the lower layer. The current identifier value is updated in this state.

- DISCARD

  The state reached by the receipt of an invalid EAP packet. A failed method integrity check or any EAP packet that is received at an improper time will result in the peer discarding the packet by setting eapIgnore. The discard count is incremented in this state.

- IDLE

  The state in which the authenticator waits for a response from the peer. eapRsp is set to FALSE to await indication from the lower layer and the timer is reset. This state is reached after every request that is received, except when the METHOD has reached a final state of SUCC, CON_SUCC, or FAIL.

- RETRANSMIT

  The state reached by a timeout when no response packet has been received. An exact copy of the last method request is made available to the lower layer.

- RECEIVED

  The state reached when a packed has arrived from the peer, as indicated by the lower layer eapRsp variable. In this state the received message is parsed to determine if it should be passed to the method or is a Nak to be handled by the EAP layer. Any other EAP packet is discarded

- NAK

  State reached by receipt of a valid Nak packet from the peer. The current method is reset and the authenticator returns to the GET METHOD state to determine its next METHOD, if any.

- SUCCESS

  The state reached upon successful authentication of the peer. SUCCESS can only be reached by complete satisfaction of the authenticator's policy. In this state the success packet is built and the lower layer notified of success.

- FAILURE

  The state reached whenever the conversation cannot continue and the authenticator's policy has not yet been satisfied. In this state the failure packet is built and the lower layer notified of failure.

4 EAP Protocol Operation

4.1 EAP Messages

EAP is a protocol between an Authenticator and a Peer. The Authenticator initiates EAP exchanges, and the Peer responds to Authenticator messages.

EAP Protocol transports data between EAP methods at the Authenticator and the Peer. A Method supports an "EAP Algorithm". Authenticators send Method Data in EAP Request messages and Peers send Method Data in EAP Reply messages. Request and Reply messages include a Type field indicating the Method using the Data.

EAP supports the negotiation of Methods to use by Authenticator and Peer. The Authenticator proposes a Method and the Peer accepts or rejects it. The Authenticator proposes a Method by sending a Request for the Method Type; the Peer accepts the proposal by sending a Reply with the same type, or rejects the proposal by sending a reply with Type of NAK.

An EAP dialog may consist of a sequence of Methods negotiated between the Authenticator and Peer. The authenticator terminates the dialog by sending an EAP Success or EAP Failure message to the Peer. The Peer does not respond to Success or Failure messages.

4.2 EAP Methods

EAP Methods are a sequence of EAP Request/Reply exchanges between the Authenticator and Peer. Each Request/Reply exchange has a unique ID field assigned by the Authenticator.

Methods may consist of one or more Request/Reply exchanges. Method implementations drive the exchange until the Method determines whether it has succeeded or failed. Note that both Authenticator and Peer Method must make this determination independently.

Once the Authenticator and Peer have agreed on a particular Method, the method MUST continue without interruption by messages from another method until both Authenticator and Peer have determined the method is complete (i.e. it succeeded or failed).

In the EAP Switch Model assumed by the state machine, the Method is able to signal the Switch when the Method is complete and whether it has succeeded or failed.

4.3 EAP Method Negotiation

EAP assumes some logic at the Authenticator and Peer that allows them to determine a mutually acceptable sequence of EAP methods to authenticate and authorize a session. The logic must be constrained by requirements for good authentication (e.g. don't negotiate down), and these requirements are outside the scope of this document. For purposes here the assumption is that the Authenticator can propose any Method and the peer can accept or reject that Method. Once started a Method must run to completion. If any accepted Method fails the EAP dialog fails.

When a Method completes successfully the Authenticator may propose another Method by sending a Request for the Method, or it may terminate the EAP dialog by sending a Success message. If an EAP method results in failure, the Authenticator terminates the dialog with an EAP Failure message.

When a Peer Method completes successfully the Peer waits for a Request for a new Method or a Success Message from the Authenticator. If it receives a Failure or nothing (timeout) it terminates the EAP dialog locally in a Failure state.

Note that the Notify Method is an exception to rules for other Methods in that it can be sent in the middle of other Methods, and the Peer must always respond with a Notify Response.

4.4 EAP Message Retransmission

The Authenticator is responsible for retransmitting Requests if it does not receive a Response. The Peer must be able to handle a retransmitted Request. The Peer can recognize a retransmitted Request if it has the same ID field as the previous Request. On receiving a retransmitted Request the Peer MUST send a Response. If a Response to the earlier Request has not been sent it MAY send a single Response. If it has sent the Response for the earlier Request prior to receiving the retransmitted Request, it MUST resend the same Response it sent to the initial Request.

The Authenticator MUST expect to receive a Response to each Request it sends; the initial as well as each retransmitted one. This means it may receive a Response after its Method has started processing an earlier response with the same ID.

Note that the requirement for link layer ordering means that no a Response for a Different ID should never come between an initial Response and Responses to retransmitted Requests.

Message retransmission and handling of Retransmitted Requests and Responses to Retransmitted Requests is assumed to be done below the EAP Switch layer in the Model assumed with this State Machine. This means that the EAP Switch Layer will need to inform the lower layers when a message it receives has been processed. In the state machine in this paper the signal intCheck is used to inform lower that EAP has processed the message successfully.

4.5 Valid EAP Requests

A Peer MUST send a Response to every Valid EAP Request. A valid request is one that meets certain EAP message sequencing rules and also meets the expectations of the Method for which the message is intended.

The EAP message sequencing rules are as follows:

1. For an Authenticator
   (a) A Response MUST have the same ID as the Request sent by the Authenticator
   (b) A Response MUST have the same Type as the Request sent by the Authenticator, or have a Type of NAK
   (c) A Response received after the initial valid response to a Method Request MUST NOT be a NAK (this is because after the initial Response the Peer has agreed to do the method and may not "negotiate" out of the Method once started).

   If an Authenticator receives a Response that fails any of the rules above, it MUST Silently Discard the Response.

2. For a Peer
   (a) An EAP message (Request, Success, Failure) received after the Peer has sent an initial Response for a Method and before the Peer has completed the Method MUST be a Request with the Type Field identical to that in the initial Method.

   If a Peer receives an EAP Message that fails the above rule, it MUST Silently Discard the Message.

3. In addition to the sequencing rules, each Method may have Integrity Check Rules specific to the Method. These rules may have to do with message format, or with Method specific security (e.g. MIC) algorithms. The EAP Switch MUST be able to handle these checks and Silently Discard messages that fail the Method Specific Integrity Checks.

4.6 Timeouts and termination of EAP dialog

At the end of an EAP Dialog the Authenticator and Peer signal to their respective lower layer that the dialog is over and is successful or is over and failed.

Authenticator termination

The Authenticator drives the dialog. It signals the end of an EAP dialog by sending an EAP Success or EAP Failure to the Peer. It also signals the lower layer that the authentication succeeded or failed.

Peer termination

The Peer may or may not receive the EAP Success or Failure message sent by the Authenticator. It does not acknowledge the Success and Failure messages, so the Authenticator does not know if the Peer received the message.

The Peer determines whether the dialog is successful independently of the Authenticator. This leaves a few possible case:

1. Peer success, Authentication Success - Peer signals lower layer success
2. Peer success, Authentication Failure - Peer signals lower layer failure.

   Note that there has been some discussion that in some cases a Peer Method could know that an EAP Failure would not be be sent by the Authenticator because the Authenticator signaled via the method that authentication succeeded. This has been suggested as a possible way to mitigate DOS attacks that use the unprotected Failure message. This is still under discussion
3. Peer success, Timeout waiting for EAP Success/Failure - Peer signals lower layer Failure.

   Note that some discussion has suggested that since the Peer believes the dialog is successful and the Success message is not retransmitted, a timeout could mean success or failure. If the Authenticator succeeded it will signal success to its lower layer and the session will be active. If the Authenticator failed it will signal failure to its lower layer and the session will not be active.
4. Peer Failure, Receive EAP Success - Peer signals lower layer failure
5. Peer Failure, Receive EAP Failure - Peer signals lower layer failure
6. Peer Failure, Timeout waiting for EAP Success/Failure - Peer signals lower layer failure
7. Peer Timeout waiting for EAP Request from Authenticator (to finish a Method) - Peer signals lower layer failure

4.7 Interfaces between EAP, EAP Methods, and lower layer

4.7.1 EAP Message passing between EAP and lower layer

EAP messages are presented to EAP Switch by the lower layer using the eapRcvd signal. EAP Switch indicates that a message is not valid using the eapIgnore signal. If the message is not valid the lower layer does whatever is required to get another message. The EAP Switch presents EAP messages to the lower layer. The lower layer is responsible for transmission of these messages.

4.7.2 Starting and Terminating EAP

The EAP Switch is initiated by a lower layer using the portEnabled signal. When portEnabled goes False, the EAP Switch is forced to the DISABLED state. When it goes from False to True the EAP Switch goes to the INITIALIZE state. When the EAP dialog is complete it sets either eapSuccess or eapFailure which signals the lower layer that the dialog is completed and has succeeded or failed.

4.7.3 Other Signals between EAP and lower layers

Some EAP methods methods can set other variables for lower layers. The most important of these in existing applications are: 1. keyAvail is set by EAP when key material is available for use by lower layer (proposed for 802.11i) 2. Port Valid is set by lower layer to indicate that the lower layer link is protected. In 802.11 this would mean the link layer keys are installed.

4.7.4 EAP Method interface to EAP Switch

EAP Methods receive EAP messages from the EAP Switch. They return an indication of whether the message was valid and the internal state of the Method. In the state machine described in this paper the method state includes [Continuing, Done-success, Done-fail, first-req-sent, Done-confirmed-success]

5 State Machine Walkthroughs

5.1 EAP Peer State Machine Walkthrough

The EAP Peer state machine goes from DISABLED to INITIALIZE state when the portEnabled signal goes from False to True. portEnabled is the signal from the lower layer that indicates the port is ready to be used. When portEnabled is False the Peer is unconditionally transitioned to DISABLED state.

INITIALIZE sets the initial conditions and then goes to the IDLE state, waiting for an EAP Message from the Authenticator. When a message arrives, the lower layer sets eapRcvd True, and the Peer transitions to the DIALOG state.

DIALOG is where the incoming Request is evaluated. The parseReceivedMessage() function sets the rx flags that determine the next State. DIALOG may exit to NAK to refuse a proposed Method, to METHOD to respond to a Request for a Method, to ACTIVE to retransmit a previously sent Response, or to DISCARD if the Message fails an EAP Message Sequence rule.

DIALOG may also exit to FAILURE or SUCCESS states if the message is a Success or Failure and the Method State is appropriate.

Assuming the Request is allowed and DIALOG state is exited to METHOD, the METHOD state is where the EAP Method logic is called. METHOD exits either to ACTIVE where the EAP Response is built and a flag set to tell the lower layer a message is ready to be sent, or to DISCARD if the incoming Request failed a Method specific Integrity Check.

ACTIVE, DISCARD and NAK all exit to IDLE where the peer again waits for the next EAP Message.

When an EAP Dialog is completed and an EAP Success or Failure is received in the IDLE State, the transition is to DIALOG and then to Either SUCCESS or FAILURE. These are terminal states and are exited only when portEnabled turns False. [Some further modification may be needed to support re-authentication]

In the IDLE state a Timeout will cause a transition to FAILURE state. A provision is also made for alternative Success or Failure indications to cause a transition to SUCCESS or FAILURE. The authors do not yet have a good example of an alternative indication, so this could be changed in the future.

5.2 EAP Authenticator State Machine Walkthrough

The EAP Authenticator state machine goes from DISABLED to INITIALIZE state when the portEnabled signal goes from False to True. portEnabled is the signal from the lower layer that indicates the port is ready to be used. When portEnabled is False the Authenticator is unconditionally transitioned to DISABLED state.

From INITIALIZE the Authenticator transitions to GET METHOD state. Here routine is called to select the METHOD to be proposed. In the terminal case no Method is selected. In this case a function is called to determine whether the Authenticator's policy has been met. If the policy has been satisfied then SUCCESS state is entered and an EAP Success Message is sent. If the policy has not been satisfied, then FAILURE state is entered and an EAP Failure Message is Sent.

The case where a Method is selected, GET METHOD transitions to METHOD. METHOD calls the METHOD function which returns methodState. In the case where a Request is to be sent METHOD transition to ACTIVE where the Request is built and a signal (eapReq) is set to tell the lower layer a Request is Ready to be sent. In a multi-Request Method a successful dialog will transition from ACTIVE to IDLE (waiting for Response) to RECEIVED and back to METHOD to create the next Request.

In the METHOD state a function is called to do Method specific integrity checks, and if the method fails the check, the signal eapIgnore is set to indicate to the lower layer that the message has been rejected and it should present the next message when it is available.

In the RECEIVED state, if the Response from the Peer is a NAK, the transition is top NAK and then to GET METHOD. The current method is reset in NAK and in GET METHOD the function to select a next method must deal with whether this is a permissible action. [Note that alternatives to this are still being discussed in the EAP group. In particular it may be desirable to only go to NAK if methodState=FIRST, meaning that a NAK will not be accepted after Peer has sent an initial good Response.]

RETRANSMIT is reached from IDLE when no Response has been received in the allocated timeout period.

6 Security Considerations

This document's intent is to describe the EAP state machine fully. To this end, any security concerns with this document are likely a reflection of security concerns with EAP itself.

References

[IEEE.802-1X.2001]
Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X, September 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2284] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, March 1998.

Authors' Addresses

John R. Vollbrecht
Vollbrecht Consulting LLC
9682 Alice Hill Drive
Dexter, MI 48130
Phone: 734 426-1026
EMail: jrv@umich.edu

Nick L. Petroni, Jr.
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
Email: npetroni@cs.umd.edu

Intellectual Property Statement

neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Expiration Date

This memo is filed as <draft-vollbrecth-eap-state-01.ps>, and expires September 1, 2003.