Internet Engineering Task Force
INTERNET-DRAFT

Payne et al.
University of Maryland

November 4, 2002

EAP State Machine

Copyright Notice

Abstract

The specification for the Extensible Authentication Protocol (EAP) [2] omits a state machine description. This omission has led to ambiguity in the specification and potential security problems in EAP implementations. This document outlines a state machine to be integrated into the next revision of the EAP RFC.

Introduction

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.

One type of EAP message has been purposely omitted from the state machines: the notification messages. This was done because notification request messages can be sent from the authenticator to the peer at any point in the state machine. Upon receipt of this request, the peer MUST send a notification reply. This transaction does not induce any changes to the peer or authenticator state machines.

Additional details about the meaning of each state, and how to handle messages in each state are included in the text accompanying each state machine diagram.

Requirements language

In this document, the key words "MAY", "MUST, "MUST NOT", "optional", "recommended", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [1].

A protocol submission is not compliant if it fails to satisfy one or more of the MUST or MUST NOT requirements for the capabilities that it implements. A protocol submission that satisfies all the MUST, MUST NOT, SHOULD and SHOULD NOT requirements for its capabilities is said to be "unconditionally compliant"; one that satisfies all the MUST and MUST NOT requirements but not all the SHOULD or SHOULD NOT requirements for its protocols is said to be "conditionally compliant."

Notational conventions used in state diagrams

The following state diagrams have been completed following the conventions specified in [IEEE8021X] 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.

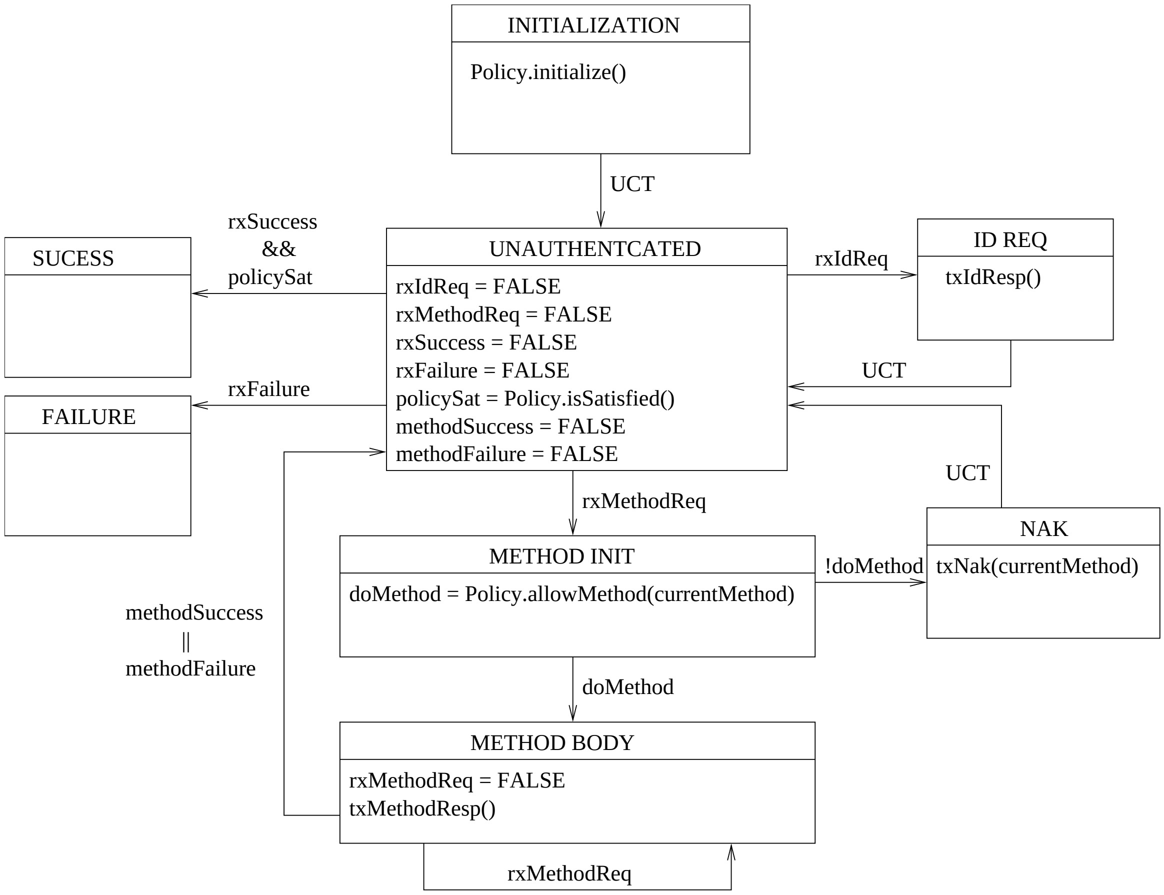
Should a conflict exist between the interpretation of a state diagram and either the corresponding global transition tables or the textual description associated with the state machine, the state diagram takes precedence.

Policy

As noted in the title, EAP is extensible in nature. Although originally designed to allow for a series of one-way authentications, current uses and new EAP methods allow for mutual authentication via a single run of the protocol. As a direct result of the extensibility of the protocol, both authenticators and peers are given the opportunity to enforce policies via the protocol. For example, a peer may choose to only allow EAP methods that provide mutual authentication or an authenticator may choose to mandate three successful method authentications out of the five it supports before allowing access. While an advantage of this is more flexibility, one disadvantage is the difficulty involved in formalizing the process. In order to provide for such flexibility, we have introduced the concept of a policy for both the authenticator and the peer. Given the procedures described below, the authenticator can insure that its policy is met before sending a success message and the peer can insure that any success message it receives only results in a transition to the success state at an allowable time. The result is a well-defined state machine that maintains the extensibility of the protocol. Of course, the security of the protocol directly depends on the effectiveness of the policy being enforced.

EAP 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.


INITIALIZATION
Policy.initialize()
UCT
rxSuccess
&&
policySat
SUCESS
UNAUTHENTCATED
rxIdReq = FALSE
rxMethodReq = FALSE
rxSuccess = FALSE
rxFailure = FALSE
policySat = Policy.isSatisfied()
methodSuccess = FALSE
methodFailure = FALSE
rxIdReq
ID REQ
txIdResp()
UCT
rxFailure
FAILURE
UCT
rxMethodReq
NAK
txNak(currentMethod)
METHOD INIT
doMethod = Policy.allowMethod(currentMethod)
!doMethod
methodSuccess
||
methodFailure
doMethod
METHOD BODY
rxMethodReq = FALSE
txMethodResp()
rxMethodReq

Variables

- rxIdReq. This variable is set to true if an Identity request is received from the Authenticator.
- rxMethodReq. This variable is set to true if a request with type greater than 3 (not including NAK, Notification, or Identity) is received from the Authenticator.
- rxSuccess. This variable is set to true if a Success packet is received from the Authenticator.
- rxFailure. This variable is set to true if a Failure packet is received from the Authenticator.
- currentMethod. This variable indicates the number of the method currently pending. It is set upon receipt of the first valid request packet from the authenticator and does not change until it is again valid to receive a new request type.
- policySat. This variable is used by the peer to store the answer of a query to Policy to determine if the Policy is currently satisfied. True indicates that the peer is sufficiently satisfied to enter the SUCCESS state, false indicates SUCCESS should not be reached.
- methodSuccess. This variable is used to indicate whether the current method has completed successfully (that is, authentication succeeded).
- methodFailure. This variable is used to indicate whether the current method has completed without successful authentication or without reaching a state satisfactory to the peer.
- doMethod. This variable is used to store the answer of a query to Policy to determine if the Policy will allow continuing with the current method, requested by the Authenticator.

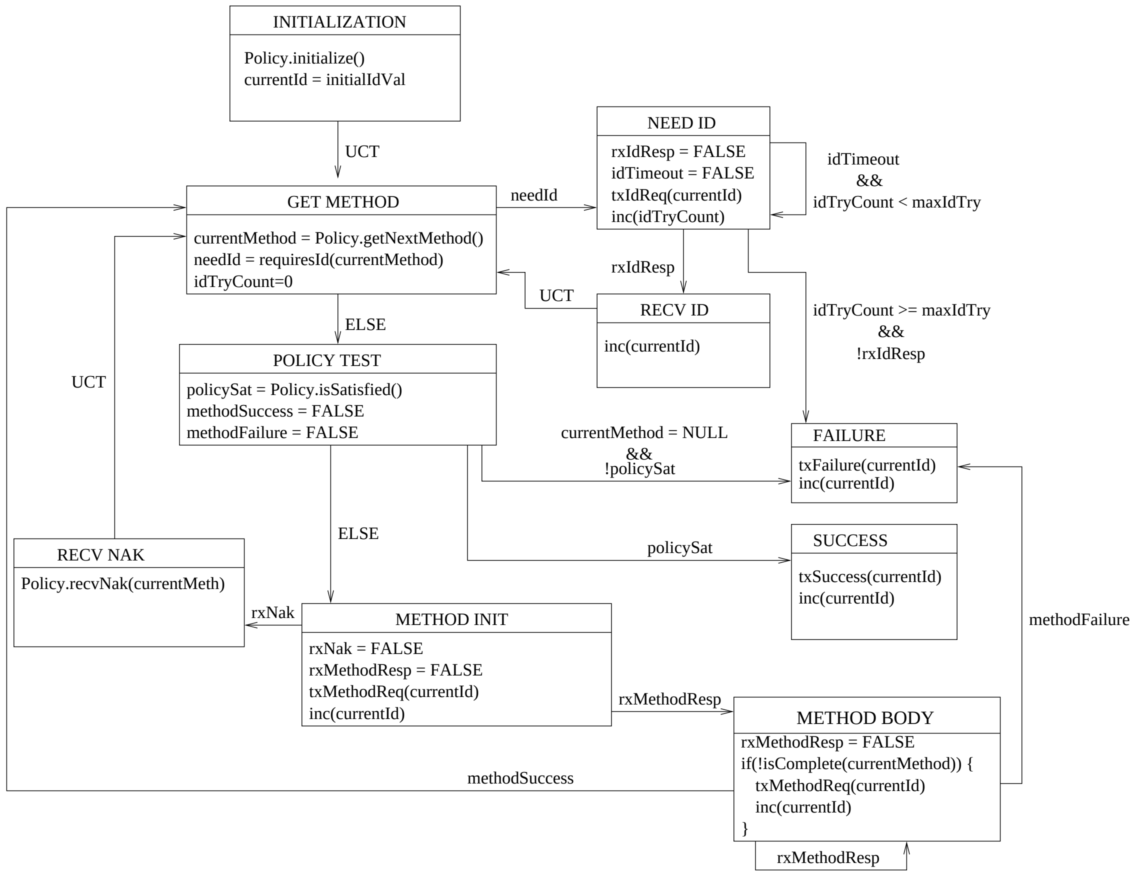
Procedures

- Policy.initialize. A policy function to reset all state variables that affect policy. This function clears all prior authentications and method completions.
- Policy.isSatisfied. A policy function that indicates whether the current state of the authentication sequence meets the policy of peer as configured.
- Policy.allowMethod. A policy method indicating whether the passed method should can be run based on the peer's policy. The passed method is based on a request from the Authenticator.
- txIdResp. This function sends an Identity response associated with the most recent valid Identity request. The identifier corresponding to this request is used in the reply, as specified by RFC 2284.
- txNak. This function sends a Nak packet for the passed method. The contents of the Nak are based on the policy of the peer.
- txMethodResp. This function sends a response packet for the associated method request. Each method maintains its own state to determine the correct contents of that response.

States

- INITIALIZATION. In this state, the peer resets all state variables relating to the satisfaction of peer policy.
- UNAUTHENTICATED. In this state the peer is active and is fielding requests from the Authenticator. These may be Identity requests or authentication method requests. Once either of these is received, the peer leaves this state until the completion of that method. This state also performs the peer policy check to determine if Success or Failure packets should allow state transitions.
- ID_REQ. In this state the peer has received an identity request and is required to respond to such a request.
- METHOD_INIT. In this state, one of the two states that represents the Method State Machine, only a single request has been received from the authenticator. The peer checks if proceeding with this method is consistent with its policy. If so, the peer continues with the method body. If not, the peer responds with a Nak.
- METHOD_BODY. The main portion of a Method's state machine. All methods must receive requests and respond until signaling Success or Failure. Until that time, the peer stays in this state.
- NAK. The peer has decided not to perform the requested method and must respond with the appropriate Nak packet. In this state the peer performs all steps to facilitate this response.
- SUCCESS. In this state the peer has received a Success packet and has determined its policy sufficiently met.
- FAILURE. In this state the peer has received a Failure packet from the Authenticator. Authentication does not continue.

EAP Authenticator State Machine



Variables

- currentId. This is the identifier, used in any EAP packet. Its value is set initially by the Authenticator and subsequently updated with each new packet.
- currentMethod. The method currently being attempted/executed by the Authenticator.
- needId. This variable stores the response of a query to a method of whether or not it would like the protocol to perform an Identity request.
- idTryCount. The current count of how many times an Identity request has been sent before receiving a response.
- rxIdResp. This variable is set to true if an Identity response is received from the Peer.
- idTimeout. This variable is set to true if the Authenticator has timed-out in waiting for an Identity response before resending the request.

- policySat. This variable is used by the Authenticator to store the answer of a query to Policy to determine if the Policy is currently satisfied. True indicates that the Authenticator is sufficiently satisfied to enter the SUCCESS state, false indicates SUCCESS should not be reached.
- methodSuccess. This variable is used to indicate whether the current method has completed successfully (that is, authentication succeeded).
- methodFailure. This variable is used to indicate whether the current method has completed without successful authentication or without reaching a state satisfactory to the Authenticator.
- rxNak. This variable is set to true if a Nak is received from the Peer.
- rxMethodResp. This variable is set to true if a response with type equal to the currentMethod and with identifier equal to the currentId is received from the Peer.

Constants

- initialIdVal. The value at which to start the unique identifier for sent packets.
- maxIdTry. The maximum value allowable for transmit retries before failing when performing an Identity request.

Procedures

- Policy.initialize. A policy function to reset all state variables that affect policy. This function clears all prior authentications and method completions.
- Policy.getNextMethod. A policy function that indicates which should be the next method attempted by the Authenticator. If no method is next, it should return NULL.
- Policy.isSatisfied. A policy function that indicates whether the current state of the authentication sequence meets the policy of Authenticator as configured.
- Policy.recvNak. A policy method that updates the current authentication sequence by indicating a refusal to perform the current method. This indication includes any suggestions given by the peer for supported methods.
- requiresId. A predicate to determine if the passed method would like an Identity request to be sent.
- txIdReq. This method sends an Identity request with the passed identifier.
- inc. Update the passed counter by adding 1.
- txFailure. Send a Failure packet with the passed identifier.
- txSuccess. Send a Success packet with the passed identifier.
- txMethodReq. Send a Method request with type greater than 3. The contents of the message depend on the internal state of the method being executed.
- isComplete. A predicate to determine if the method in question is complete (that is, if it has no more messages to send). This is equivalent to testing if the method has sent its last message.

States

- INITIALIZATION. In this state, the Authenticator resets all state variables relating to the satisfaction of Authenticator policy, as well as initializing the identification counter.
- GET_METHOD. This is the base state for the Authenticator from which decisions are made as to what the next method will be and whether or not that method requires an Identity request. This state is returned to following an Identity response so that a new method could be chosen if necessary.
- POLICY_TEST. In this state the Authenticator tests whether or not it is time to send a Success message based on the state of Policy satisfaction. Policy can only be satisfied if there are no methods remaining to be completed.
- NEED_ID. In this state the Authenticator has determined an Identity response is needed and so an Identity request is sent until a response is received or until a try counter has exceeded its limit.
- RECV_ID. The state indicating receipt of an Identity response. In this state the Authenticator does post processing on the result if necessary.
- METHOD_INIT. The first of two states representing the Method State Machine. In this state, the first message of the current method is formulated and sent. The Authenticator then waits for either a response to that method, indicating the method conversation will continue, or a Nak indicating a new method should be tried if possible.
- RECV_NAK. The state to handle the case where the chosen method is not desired by the Peer. In this state, the necessary post-processing is done to determine how the Authenticator will proceed.
- METHOD_BODY. This represents a Method in its most general sense. A method formulates requests and waits for corresponding responses until it is complete. Once complete, either the methodSuccess or methodFailure variables must be set.
- SUCCESS. The state reached when authentication has succeeded.
- FAILURE. The state reached when authentication has failed.

References

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

[2] Blunk, L., Vollbrecht, J., Aboba, B., "Extensible Authentication Protocol (EAP)", RFC 2284, March 1998.

[IEEE8021X] IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std 802.1X-2001, June 2001.

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.

IANA Considerations

This draft does not create any new number spaces for IANA administration.

Acknowledgments

Thanks to the members of the EAP working group who have discussed and commented on this document. We would also like to thank William Arbaugh, Univ of Maryland, for providing resources needed to complete this document. Finally, we would like to thank Bernard Aboba, Microsoft, for being both flexible and helpful as we begin working with the IETF.

Authors' Addresses

Bryan D. Payne
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
Email: bdpayne@cs.umd.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

Chuk Yang Seng
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
Email: sengcy@cs.umd.edu

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; 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-payne-eap-sm-01.ps>, and expires May 4, 2003.