

PANA Working Group
Internet-Draft
Expires: April 21, 2006

V. Fajardo
Y. Ohba
TARI
R. Lopez
Univ. of Murcia
October 18, 2005

**State Machines for Protocol for Carrying Authentication for Network
Access (PANA)
draft-ietf-pana-statemachine-02**

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

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 April 21, 2006.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

This document defines the conceptual state machines for the Protocol for Carrying Authentication for Network Access (PANA). The state machines consist of the PANA Client (PaC) state machine and the PANA Authentication Agent (PAA) state machine. The two state machines show how PANA can interface to EAP state machines and can be

implemented with supporting various features including separate NAP and ISP authentications, ISP selection and mobility optimization. The state machines and associated model are informative only. Implementations may achieve the same results using different methods.

Table of Contents

1.	Introduction	4
2.	Interface Between PANA and EAP	5
3.	Document Authority	7
4.	Notations	8
5.	Common Rules	10
5.1.	Common Procedures	10
5.2.	Common Variables	12
5.3.	Constants	13
5.4.	Common Message Initialization Rules	13
5.5.	Common Error Handling Rules	14
5.6.	Common State Transitions	14
6.	PaC State Machine	16
6.1.	Interface between PaC and EAP Peer	16
6.1.1.	Delivering EAP Messages from PaC to EAP Peer	16
6.1.2.	Delivering EAP Responses from EAP Peer to PaC	16
6.1.3.	EAP Restart Notification from PaC to EAP Peer	16
6.1.4.	EAP Authentication Result Notification from EAP Peer to PaC	17
6.1.5.	Alternate Failure Notification from PaC to EAP Peer	17
6.1.6.	EAP Invalid Message Notification from EAP Peer to PaC	17
6.2.	Variables	17
6.3.	Procedures	18
6.4.	PaC State Transition Table	19
7.	PAA State Machine	31
7.1.	Interface between PAA and EAP Authenticator	31
7.1.1.	EAP Restart Notification from PAA to EAP Authenticator	31
7.1.2.	Delivering EAP Responses from PAA to EAP Authenticator	31
7.1.3.	Delivering EAP Messages from EAP Authenticator to PAA	31
7.1.4.	EAP Authentication Result Notification from EAP Authenticator to PAA	31
7.2.	Variables	32
7.3.	Procedures	34
7.4.	PAA State Transition Table	34
8.	Mobility Optimization Support	49
8.1.	Common Variables	49
8.2.	PaC Mobility Optimization State Machine	49

8.2.1.	Variables	49
8.2.2.	Procedures	50
8.2.3.	PaC Mobility Optimization State Transition Table Addendum	50
8.3.	PAA Mobility Optimization	53
8.3.1.	Procedures	53
8.3.2.	PAA Mobility Optimization State Transition Table Addendum	53
9.	Implementation Considerations	55
9.1.	PAA and PaC Interface to Service Management Entity	55
9.2.	Multicast Traffic	55
10.	Security Considerations	56
11.	IANA Considerations	57
12.	Acknowledgments	58
13.	References	59
13.1.	Normative References	59
13.2.	Informative References	59
	Authors' Addresses	60
	Intellectual Property and Copyright Statements	61

1. Introduction

This document defines the state machines for Protocol Carrying Authentication for Network Access (PANA) [[I-D.ietf-pana-pana](#)]. There are state machines for the PANA client (PaC) and for the PANA Authentication Agent (PAA). Each state machine is specified through a set of variables, procedures and a state transition table.

A PANA protocol execution consists of several exchanges to carry authentication information. Specifically, EAP PDUs are transported inside PANA PDUs between PaC and PAA, that is PANA represents a lower layer for EAP protocol. Thus, a PANA state machine bases its execution on an EAP state machine execution and vice versa. Thus this document also shows for each of PaC and PAA an interface between an EAP state machine and a PANA state machine and how this interface allows to exchange information between them. Thanks to this interface, a PANA state machine can be informed about several events generated in an EAP state machine and make its execution conditional to its events.

The details of EAP state machines are out of the scope of this document. Additional information can be found in [[I-D.ietf-eap-statemachine](#)]. Nevertheless PANA state machines presented here have been coordinated with state machines shown by [[I-D.ietf-eap-statemachine](#)].

This document, apart from defining PaC and PAA state machines and their interfaces to EAP state machines (running on top of PANA), provides some implementation considerations, taking into account that it is not a specification but an implementation guideline.

2. Interface Between PANA and EAP

PANA carries EAP messages exchanged between an EAP peer and an EAP authenticator (see Figure 1). Thus a PANA state machine must interact with an EAP state machine.

Two state machines are defined in this document : the PaC state machine (see [Section 6](#)) and the PAA state machine (see [Section 7](#)). The definition of each state machine consists of a set of variables, procedures and a state transition table. A subset of these variables and procedures defines the interface between a PANA state machine and an EAP state machine and the state transition table defines the PANA state machine behavior based on results obtained through them.

On the one hand, the PaC state machine interacts with an EAP peer state machine in order to carry out the PANA protocol on the PaC side. On the other hand, the PAA state machine interacts with an EAP authenticator state machine to run the PANA protocol on the PAA side.

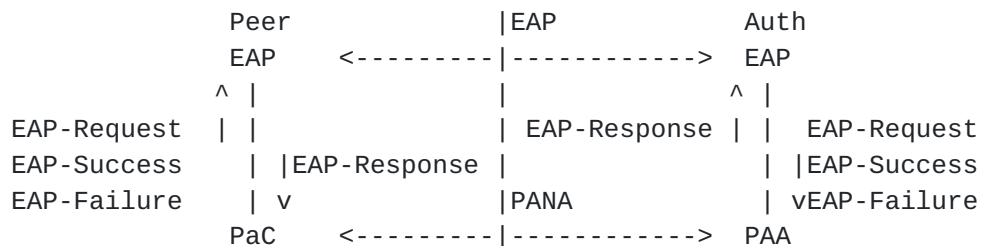


Figure 1: Interface between PANA and EAP

Thus two interfaces are needed between PANA state machines and EAP state machines, namely:

- o Interface between the PaC state machine and the EAP peer state machine
- o Interface between the PAA state machine and the EAP authenticator state machine

In general, the PaC state machine presents EAP messages (EAP-Request, EAP-Success and EAP-Failure messages) to the EAP peer state machine through the interface. The EAP peer state machine processes these messages and sends EAP messages (EAP-Response messages) through the PaC state machine that is responsible for actually transmitting this message.

On the other hand, the PAA state machine presents response messages (EAP-Response messages) to the EAP authenticator state machine through interface defined between them. The EAP authenticator

processes these messages and generate EAP messages (EAP-Request, EAP-Success and EAP-Failure messages) that are send to the PAA state machine to be sent.

For example, [[I-D.ietf-eap-statemachine](#)] specifies four interfaces to lower layers: (i) an interface between the EAP peer state machine and a lower layer, (ii) an interface between the EAP standalone authenticator state machine and a lower layer, (iii) an interface between the EAP full authenticator state machine and a lower layer and (iv) an interface between the EAP backend authenticator state machine and a lower layer. In this document, the PANA protocol is the lower layer of EAP and only the first three interfaces are of interest to PANA. The second and third interfaces are the same. In this regard, the EAP standalone authenticator or the EAP full authenticator and its state machine in [[I-D.ietf-eap-statemachine](#)] are referred to as the EAP authenticator and the EAP authenticator state machine, respectively, in this document. If an EAP peer and an EAP authenticator follow the state machines defined in [[I-D.ietf-eap-statemachine](#)], the interfaces between PANA and EAP could be based on that document. Detailed definition of interfaces between PANA and EAP are described in the subsequent sections.

3. Document Authority

When a discrepancy occurs between any part of this document and any of the related documents ([[I-D.ietf-pana-pana](#)], [I-D.ietf-pana-mobopts], [[I-D.ietf-eap-statemachine](#)] the latter (the other documents) are considered authoritative and takes precedence.

4. Notations

The following state transition tables are completed mostly based on the conventions specified in [[I-D.ietf-eap-statemachine](#)]. The complete text is described below.

State transition tables are used to represent the operation of the protocol by a number of cooperating state machines each comprising a group of connected, mutually exclusive states. Only one state of each machine can be active at any given time.

All permissible transitions from a given state to other states and associated actions performed when the transitions occur are represented by using triplets of (exit condition, exit action, exit state). All conditions are expressions that evaluate to TRUE or FALSE; if a condition evaluates to TRUE, then the condition is met. A state "ANY" is a wildcard state that matches the current state in each state machine. The exit conditions of a wildcard state are evaluated after all other exit conditions of specific to the current state are met.

On exit from a state, the exit actions defined for the state and the exit condition are executed exactly once, in the order that they appear on the page. (Note that the procedures defined in [[I-D.ietf-eap-statemachine](#)] are executed on entry to a state, which is one major difference from this document.) Each exit action is deemed to be atomic; i.e., execution of an exit action completes before the next sequential exit action starts to execute. No exit action execute outside of a state block. The exit actions in only one state block execute at a time even if the conditions for execution of state blocks in different state machines are satisfied. All exit actions in an executing state block complete execution before the transition to and execution of any other state blocks. The execution of any state block appears to be atomic with respect to the execution of any other state block and the transition condition to that state from the previous state is TRUE when execution commences. The order of execution of state blocks in different state machines is undefined except as constrained by their transition conditions. A variable that is set to a particular value in a state block retains this value until a subsequent state block executes an exit action that modifies the value.

On completion of the transition from the previous state to the current state, all exit conditions occurring during the current state (including exit conditions defined for the wildcard state) are evaluated until an exit condition for that state is met.

Any event variable is set to TRUE when the corresponding event occurs

and set to FALSE immediately after completion of the action associated with the current state and the event.

The interpretation of the special symbols and operators used is defined in [[I-D.ietf-eap-statemachine](#)].

5. Common Rules

There are following procedures, variables, message initializing rules and state transitions that are common to both the PaC and PAA state machines.

Throughout this document, the character string "PANA_MESSAGE_NAME" matches any one of the abbreviated PANA message names, i.e., "PDI", "PSR", "PSA", "PAR", "PAN", "PBR", "PBA", "PFER", "PFEA", "PTR", "PTA", "PPR", "PPA", "PRAR", "PRAA", "PUR", "PUA", "PER" and "PEA".

5.1. Common Procedures

void None()

A null procedure, i.e., nothing is done.

void Disconnect()

A procedure to delete the PANA session as well as the corresponding EAP session and authorization state.

boolean Authorize()

A procedure to create or modify authorization state. It returns TRUE if authorization is successful. Otherwise, it returns FALSE. It is assumed that Authorize() procedure of PaC state machine always returns TRUE.

void Tx:PANA_MESSAGE_NAME()

A procedure to send a PANA message to its peering PANA entity.

void TxEAP()

A procedure to send an EAP message to the EAP state machine it interfaces to.

void RtxTimerStart()

A procedure to start the retransmission timer, reset RTX_COUNTER variable to zero and set an appropriate value to RTX_MAX_NUM variable.

void RtxTimerStop()

A procedure to stop the retransmission timer.

```
void SessionTimerStart()
```

A procedure to start PANA session timer.

```
void SessionTimerStop()
```

A procedure to stop the PANA session timer.

```
void Retransmit()
```

A procedure to retransmit a PANA message and increment RTX_COUNTER by one(1).

```
void EAP_Restart()
```

A procedure to (re)start an EAP conversation resulting in the re-initialization of an existing EAP session.

```
void PANA_MESSAGE_NAME.insert_avp("AVP_NAME")
```

A procedure to insert an AVP of the specified AVP name in the specified PANA message.

```
boolean PANA_MESSAGE_NAME.exist_avp("AVP_NAME")
```

A procedure that checks whether an AVP of the specified AVP name exists in the specified PANA message and returns TRUE if the specified AVP is found, otherwise returns FALSE.

```
boolean key_available()
```

A procedure to check whether the PANA session has a PANA_MAC_KEY. If the state machine already has a PANA_MAC_KEY, it returns TRUE. If the state machine does not have a PANA_MAC_KEY, it tries to retrieve a AAA-Key from the EAP entity. If a AAA-Key is retrieved, it computes a PANA_MAC_KEY from the AAA-Key and returns TRUE. Otherwise, it returns FALSE.

```
boolean fatal(int)
```

A procedure to check whether an integer result code value indicates a fatal error. If the result code indicates a fatal error, the procedure returns TRUE, otherwise, it return FALSE. A fatal error would also result in the termination of the session and release of all resources related to that session.

5.2. Common Variables

PANA_MESSAGE_NAME.S_flag

This variable contains the S-Flag value of the specified PANA message.

PBR.RESULT_CODE

This variable contains the Result-Code AVP value in the PANA-Bind-Request message in process. When this variable carries PANA_SUCCESS when there is only once EAP run in the authentication and authorization phase, it is assumed that the PBR message always contains an EAP-Payload AVP which carries an EAP-Success message.

PFER.RESULT_CODE

This variable contains the Result-Code AVP value in the PANA-FirstAuth-End-Request message in process. When this variable carries PANA_SUCCESS, it is assumed that the PFER message always contains an EAP-Payload AVP which carries an EAP-Success message.

PER.RESULT_CODE

This variable contains the Result-Code AVP value in the PANA-Error-Request message in process.

RTX_COUNTER

This variable contains the current number of retransmissions of the outstanding PANA message.

Rx:PANA_MESSAGE_NAME

This event variable is set to TRUE when the specified PANA message is received from its peering PANA entity.

RTX_TIMEOUT

This event variable is set to TRUE when the retransmission timer is expired.

REAUTH

This event variable is set to TRUE when an initiation of re-authentication phase is triggered.

TERMINATE

This event variable is set to TRUE when initiation of PANA session termination is triggered.

PANA_PING

This event variable is set to TRUE when initiation of liveness test based on PPR-PPA exchange is triggered.

NOTIFY

This event variable is set to TRUE if the PaC or PAA wants to send attribute updates or notifications.

SESS_TIMEOUT

This event variable is set to TRUE when the session timer is expired.

ABORT_ON_1ST_EAP_FAILURE

This variable indicates whether the PANA session is immediately terminated when the 1st EAP authentication fails.

CARRY_DEVICE_ID

This variable indicates whether a Device-Id AVP is carried in a PANA-Bind-Request or PANA_Bind-Answer message. For the PAA, this variable MUST be set when a link-layer or IP address is used as the device identifier of the PaC and a Protection-Capability AVP is included in the PANA-Bind-Request message.

ANY

This event variable is set to TRUE when any event occurs.

5.3. Constants**RTX_MAX_NUM**

Configurable maximum for how many retransmissions should be attempted before aborting.

5.4. Common Message Initialization Rules

When a message is prepared for sending, it is initialized as follows:

- o For a request message, R-flag of the header is set. Otherwise, R-flag is not set.
- o S-flag and N-flag of the header are not set.
- o AVPs that are mandatory included in a message are inserted with appropriate values set.
- o A Notification AVP is inserted if there is some notification string to send to the communicating peer.

5.5. Common Error Handling Rules

For simplicity, the PANA state machines defined in this document do not support an optional feature of sending a PER message when an invalid PANA message is received [[I-D.ietf-pana-pana](#)], while the state machines support sending a PER message generated in other cases as well as receiving and processing a PER message. It is left to implementations as to whether they provide a means to send a PER message when an invalid PANA message is received.

5.6. Common State Transitions

The following transitions can occur at any state.

 State: ANY

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Re-transmissions)- - - - -		
RTX_TIMEOUT && RTX_COUNTER< RTX_MAX_NUM	Retransmit();	(no change)
- - - - - (Reach maximum number of transmissions)- - - - -		
RTX_TIMEOUT && RTX_COUNTER>= RTX_MAX_NUM	Disconnect();	CLOSED
SESS_TIMEOUT	Disconnect();	CLOSED
- - - - - (PANA-Error-Message-Processing)- - - - -		
Rx:PER && fatal (PER.RESULT_CODE) && PER.exist_avp("MAC") && key_available()	PEA.insert_avp("MAC"); Tx:PEA(); Disconnect();	CLOSED
Rx:PER && !fatal (PER.RESULT_CODE) !PER.exist_avp("MAC") !key_available()	Tx:PEA();	(no change)
- - - - -		

The following transitions can occur on any exit condition within the specified state.

 State: CLOSED

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Session termination initiated by PaC) - - - - -		
ANY	None();	CLOSED
- - - - -		

6. PaC State Machine

6.1. Interface between PaC and EAP Peer

This interface defines the interactions between a PaC and an EAP peer. The interface serves as a mechanism to deliver EAP messages for the EAP peer. It allows the EAP peer to receive EAP requests and send EAP responses via the PaC. It also provides a mechanism to notify the EAP peer of PaC events and a mechanism to receive notification of EAP peer events. The EAP message delivery mechanism as well as the event notification mechanism in this interface have direct correlation with the PaC state transition table entries. These message delivery and event notifications mechanisms occur only within the context of their associated states or exit actions.

6.1.1. Delivering EAP Messages from PaC to EAP Peer

TxEAP() procedure in the PaC state machine serves as the mechanism to deliver EAP request, EAP success and EAP failure messages contained in PANA-Auth-Request messages to the EAP peer. This procedure is enabled only after an EAP restart event is notified to the EAP peer and before any event resulting in a termination of the EAP peer session. In the case where the EAP peer follows the EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)], TxEAP() procedure sets eapReq variable of the EAP peer state machine and puts the EAP request in eapReqData variable of the EAP peer state machine.

6.1.2. Delivering EAP Responses from EAP Peer to PaC

An EAP response is delivered from the EAP peer to the PaC via EAP_RESPONSE event variable. The event variable is set when the EAP peer passes the EAP response to its lower-layer. In the case where the EAP peer follows the EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)], EAP_RESPONSE event variable refers to eapResp variable of the EAP peer state machine and the EAP response is contained in eapRespData variable of the EAP peer state machine.

6.1.3. EAP Restart Notification from PaC to EAP Peer

The EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)] has an initialization procedure before receiving an EAP request. To initialize the EAP state machine, the PaC state machine defines an event notification mechanism to send an EAP (re)start event to the EAP peer. The event notification is done via EAP_Restart() procedure in the initialization action of the PaC state machine.

6.1.4. EAP Authentication Result Notification from EAP Peer to PaC

In order for the EAP peer to notify the PaC of an EAP authentication result, EAP_SUCCESS and EAP_FAILURE event variables are defined. In the case where the EAP peer follows the EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)], EAP_SUCCESS and EAP_FAILURE event variables refer to eapSuccess and eapFail variables of the EAP peer state machine, respectively. In this case, if EAP_SUCCESS event variable is set to TRUE and a AAA-Key is generated by the EAP authentication method in use, eapKeyAvailable variable is set to TRUE and eapKeyData variable contains the AAA-Key. Note that EAP_SUCCESS and EAP_FAILURE event variables may be set to TRUE even before the PaC receives a PBR or a PFER from the PAA.

6.1.5. Alternate Failure Notification from PaC to EAP Peer

alt_reject() procedure in the PaC state machine serves as the mechanism to deliver an authentication failure event to the EAP peer without accompanying an EAP message. In the case where the EAP peer follows the EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)], alt_reject() procedure sets altReject variable of the EAP peer state machine. Note that the EAP peer state machine in [[I-D.ietf-eap-statemachine](#)] also defines altAccept variable, however, it is never used in PANA in which EAP-Success messages are reliably delivered by PANA-Bind exchange.

6.1.6. EAP Invalid Message Notification from EAP Peer to PaC

In order for the EAP peer to notify the PaC of a receipt of an invalid EAP message, EAP_INVALID_MSG event variable is defined. In the case where the EAP peer follows the EAP peer state machine defined in [[I-D.ietf-eap-statemachine](#)], EAP_INVALID_MSG event variable refers to eapNoResp variable of the EAP peer state machine.

6.2. Variables

SEPARATE

This variable indicates whether the PaC desires NAP/ISP separate authentication.

1ST_EAP

This variable indicates whether the 1st EAP authentication is success, failure or yet completed.

AUTH_USER

This event variable is set to TRUE when initiation of EAP-based (re-)authentication is triggered by the application.

EAP_SUCCESS

This event variable is set to TRUE when the EAP peer determines that EAP conversation completes with success.

EAP_FAILURE

This event variable is set to TRUE when the EAP peer determines that EAP conversation completes with failure.

EAP_RESPONSE

This event variable is set to TRUE when the EAP peer delivers an EAP Response to the PaC. This event accompanies an EAP-Response message received from the EAP peer.

EAP_INVALID_MSG

This event variable is set to TRUE when the EAP peer silently discards an EAP message. This event does not accompany any EAP message.

EAP_RESP_TIMEOUT

This event variable is set to TRUE when the PaC that has passed an EAP-Request to the EAP-layer does not receive a corresponding EAP-Response from the the EAP-layer in a given period.

6.3. Procedures**boolean choose_isp()**

This procedure returns TRUE when the PaC chooses one ISP, otherwise returns FALSE.

boolean ppac_available()

This procedure returns TRUE when the Post-PANA-Address-Configuration method specified by the PAA is available in the PaC and that the PaC will be able to comply.

boolean pcap_supported()

This procedure returns TRUE when the cryptographic data protection supplied in the Protection-Capability AVP can be supported by the PaC.

boolean eap_piggyback()

This procedure returns TRUE to indicate whether the next EAP response will be carried in the pending PAN message for optimization.

void alt_reject()

This procedure informs the EAP peer of an authentication failure event without accompanying an EAP message.

void EAP_RespTimerStart()

A procedure to start a timer to receive an EAP-Response from the EAP peer.

void EAP_RespTimerStop()

A procedure to stop a timer to receive an EAP-Response from the EAP peer.

6.4. PaC State Transition Table

State: OFFLINE (Initial State)

Initialization Action:

```
SEPARATE=Set|Unset;
CARRY_DEVICE_ID=Unset;
1ST_EAP=Unset;
RtxTimerStop();
```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PSR processing) - - - - -		
Rx:PSR && PSR.exist_avp ("EAP-Payload")	RtxTimerStop(); EAP_Restart(); TxEAP(); SEPARATE=Unset;	WAIT_EAP_MSG_ IN_DISC


```

Rx:PSR &&                                RtxTimerStop();                WAIT_PAA
!PSR.exist_avp                            if (choose_isp())
("EAP-Payload") &&                        PSA.insert_avp("ISP");
PSR.S_flag==1 &&                          PSA.S_flag=1;
SEPARATE==Set &&                          PSA.insert_avp("Cookie");
PSR.exist_avp                            Tx:PSA();
("Cookie")                               RtxTimerStart();
                                         EAP_Restart();

```

```

Rx:PSR &&                                RtxTimerStop();                WAIT_PAA
!PSR.exist_avp                            if (choose_isp())
("EAP-Payload") &&                        PSA.insert_avp("ISP");
PSR.S_flag==1 &&                          PSA.S_flag=1;
SEPARATE==Set &&                          Tx:PSA();
!PSR.exist_avp                            EAP_Restart();
("Cookie")

```

```

Rx:PSR &&                                RtxTimerStop();                WAIT_PAA
!PSR.exist_avp                            if (choose_isp())
("EAP-Payload") &&                        PSA.insert_avp("ISP");
(PSR.S_flag!=1 ||                         PSA.insert_avp("Cookie");
SEPARATE==Unset) &&                      Tx:PSA();
PSR.exist_avp                            RtxTimerStart();
("Cookie")                               SEPARATE=Unset;
                                         EAP_Restart();

```

```

Rx:PSR &&                                RtxTimerStop();                WAIT_PAA
!PSR.exist_avp                            if (choose_isp())
("EAP-Payload") &&                        PSA.insert_avp("ISP");
(PSR.S_flag!=1 ||                         Tx:PSA();
SEPARATE==Unset) &&                      SEPARATE=Unset;
!PSR.exist_avp                            EAP_Restart();
("Cookie")

```

```

- - - - -
- - - - - (Authentication trigger from application) - - -
AUTH_USER                                Tx:PDI();                OFFLINE
                                         RtxTimerStart();
- - - - -

```

```

-----
State: WAIT_EAP_MSG_IN_DISC
-----

```

```

Exit Condition          Exit Action          Exit State
-----+-----+-----
- - - - - (Return PSA with EAP-Payload) - - - - -

```


EAP_RESPONSE	PSA.insert_avp ("EAP-Payload") if (choose_isp()) PSA.insert_avp("ISP"); Tx:PSA();	WAIT_PAA
--------------	---	----------

EAP_RESP_TIMEOUT EAP_INVALID_MSG	None();	OFFLINE
--	---------	---------

State: WAIT_PAA

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PAR-PAN exchange) - - - - -		
Rx:PAR && !eap_piggyback()	RtxTimerStop(); TxEAP(); EAP_RespTimerStart(); if (key_available()) PAN.insert_avp("MAC"); PAN.S_flag=PAR.S_flag; PAN.N_flag=PAR.N_flag; Tx:PAN();	WAIT_EAP_MSG
Rx:PAR && eap_piggyback()	RtxTimerStop(); TxEAP(); EAP_RespTimerStart();	WAIT_EAP_MSG
Rx:PAN	RtxTimerStop();	WAIT_PAA
- - - - - (1st EAP result) - - - - -		
Rx:PFER && 1ST_EAP==Unset && SEPARATE==Set && PFER.RESULT_CODE== PANA_SUCCESS && PFER.S_flag==1	1ST_EAP=Success; TxEAP();	WAIT_1ST_EAP_ RESULT
Rx:PFER && 1ST_EAP==Unset && SEPARATE==Set && PFER.RESULT_CODE!= PANA_SUCCESS && PFER.S_flag==1 && ABORT_ON_1ST_EAP_FAILURE ==Unset &&	1ST_EAP=Failure; TxEAP();	WAIT_1ST_EAP_ RESULT


```
PFER.exist_avp  
("EAP-Payload")
```

```
Rx:PFER &&                1ST_EAP=Failure;        WAIT_1ST_EAP_  
1ST_EAP==Unset &&          alt_reject();          RESULT  
SEPARATE==Set &&  
PFER.RESULT_CODE!=  
    PANA_SUCCESS &&  
PFER.S_flag==1 &&  
ABORT_ON_1ST_EAP_FAILURE  
==Unset &&  
!PFER.exist_avp  
("EAP-Payload")
```

```
Rx:PFER &&                1ST_EAP=Failure;        WAIT_1ST_EAP_  
1ST_EAP==Unset &&          TxEAP();              RESULT_CLOSED  
SEPARATE==Set &&  
PFER.RESULT_CODE!=  
    PANA_SUCCESS &&  
(PFER.S_flag==0 ||  
ABORT_ON_1ST_EAP_FAILURE  
==Set) &&  
PFER.exist_avp  
("EAP-Payload")
```

```
Rx:PFER &&                1ST_EAP=Failure;        WAIT_1ST_EAP_  
1ST_EAP==Unset &&          alt_reject();          RESULT_CLOSED  
SEPARATE==Set &&  
PFER.RESULT_CODE!=  
    PANA_SUCCESS &&  
(PFER.S_flag==0 ||  
ABORT_ON_1ST_EAP_FAILURE  
==Set) &&  
!PFER.exist_avp  
("EAP-Payload")
```

```
Rx:PBR &&                TxEAP();                WAIT_EAP_RESULT  
1ST_EAP==Unset &&          if (PBR.exist_avp  
SEPARATE==Unset &&              ("Device-Id"))  
PBR.RESULT_CODE==          CARRY_DEVICE_ID=Set;  
    PANA_SUCCESS
```

```
Rx:PBR &&                TxEAP();                WAIT_EAP_RESULT_  
1ST_EAP==Unset &&          CLOSE  
SEPARATE==Unset &&  
PBR.RESULT_CODE!=  
    PANA_SUCCESS &&  
PBR.exist_avp
```


("EAP-Payload")

Rx:PBR &&	alt_reject();	WAIT_EAP_RESULT_
1ST_EAP==Unset &&		CLOSE

SEPARATE==Unset &&
PBR.RESULT_CODE!=
PANA_SUCCESS &&
!PBR.exist_avp
("EAP-Payload")

- - - - -

- - - - - (2nd EAP result) - - - - -

Rx:PBR &&	TxEAP();	WAIT_EAP_RESULT
1ST_EAP==Success &&	if (PBR.exist_avp	
PBR.RESULT_CODE==	("Device-Id"))	
PANA_SUCCESS &&	CARRY_DEVICE_ID=Set;	
PBR.exist_avp		
("EAP-Payload")		

Rx:PBR &&	alt_reject();	WAIT_EAP_RESULT
1ST_EAP==Success &&	if (PBR.exist_avp	
PBR.RESULT_CODE==	("Device-Id"))	
PANA_SUCCESS &&	CARRY_DEVICE_ID=Set;	
!PBR.exist_avp		
("EAP-Payload")		

Rx:PBR &&	TxEAP();	WAIT_EAP_RESULT_
1ST_EAP==Success &&		CLOSE
PBR.RESULT_CODE!=		
PANA_SUCCESS &&		
PBR.exist_avp		
("EAP-Payload")		

Rx:PBR &&	alt_reject();	WAIT_EAP_RESULT_
1ST_EAP==Success &&		CLOSE
PBR.RESULT_CODE!=		
PANA_SUCCESS &&		
!PBR.exist_avp		
("EAP-Payload")		

Rx:PBR &&	TxEAP();	WAIT_EAP_RESULT
1ST_EAP==Failure &&	if (PBR.exist_avp	
PBR.RESULT_CODE==	("Device-Id"))	
PANA_SUCCESS	CARRY_DEVICE_ID=Set;	

Rx:PBR &&	TxEAP();	WAIT_EAP_RESULT_
1ST_EAP==Failure &&		CLOSE
PBR.RESULT_CODE!=		
PANA_SUCCESS &&		


```
PBR.exist_avp
("EAP-Payload")
```

```
Rx:PBR &&                                alt_reject();                WAIT_EAP_RESULT_
1ST_EAP==Failure &&                        CLOSE
PBR.RESULT_CODE!=
  PANA_SUCCESS &&
!PBR.exist_avp
("EAP-Payload")
```

```
- - - - -
```

```
-----
State: WAIT_EAP_MSG
-----
```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Return PAN/PAR) - - - - -		
EAP_RESPONSE && eap_piggyback()	EAP_RespTimerStop() PAN.insert_avp ("EAP-Payload"); if (key_available()) PAN.insert_avp("MAC"); PAN.S_flag=PAR.S_flag; PAN.N_flag=PAR.N_flag; Tx: PAN();	WAIT_PAA
EAP_RESPONSE && !eap_piggyback()	EAP_RespTimerStop() PAR.insert_avp ("EAP-Payload"); if (key_available()) PAR.insert_avp("MAC"); PAR.S_flag=PAR.S_flag; PAR.N_flag=PAR.N_flag; Tx: PAR(); RtxTimerStart();	WAIT_PAA
EAP_RESP_TIMEOUT	if (key_available()) PAN.insert_avp("MAC"); PAN.S_flag=PAR.S_flag; PAN.N_flag=PAR.N_flag; Tx: PAN();	WAIT_PAA
EAP_INVALID_MSG EAP_SUCCESS EAP_FAILURE	None();	WAIT_PAA
- - - - -		

 State: WAIT_EAP_RESULT

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - (EAP Result) - - - - -		
EAP_SUCCESS && PBR.exist_avp ("Key-Id") && ppac_available() && (!PBR.exist_avp ("Protection- Capability") (PBR.exist_avp ("Protection- Capability") && pcap_supported()))	PBA.insert_avp("MAC"); PBA.insert_avp("Key-Id"); if (CARRY_DEVICE_ID) PBA.insert_avp ("Device-Id"); PBA.insert_avp("PPAC"); Tx:PBA(); Authorize(); SessionTimerStart();	OPEN
EAP_SUCCESS && !PBR.exist_avp ("Key-Id") && ppac_available() && (!PBR.exist_avp ("Protection- Capability") (PBR.exist_avp ("Protection- Capability") && pcap_supported()))	if (key_available()) PBA.insert_avp("MAC"); if (CARRY_DEVICE_ID) PBA.insert_avp ("Device-Id"); PBA.insert_avp("PPAC"); Tx:PBA(); Authorize(); SessionTimerStart();	OPEN
EAP_SUCCESS && !ppac_available()	if (key_available()) PER.insert_avp("MAC"); PER.RESULT_CODE= PANA_PPAC_CAPABILITY_ UNSUPPORTED Tx:PER(); RtxTimerStart();	WAIT_PEA
EAP_SUCCESS && (PBR.exist_avp ("Protection- Capability") && !pcap_supported())	if (key_available()) PER.insert_avp("MAC"); PER.RESULT_CODE= PANA_PROTECTION_ CAPABILITY_UNSUPPORTED Tx:PER(); RtxTimerStart();	WAIT_PEA
EAP_FAILURE &&	if (key_available())	OPEN

<pre>(SEPARATE==Set) && ppac_available() && (!PBR.exist_avp ("Protection- Capability") (PBR.exist_avp ("Protection- Capability") && pcap_supported()))</pre>	<pre>PBA.insert_avp("MAC"); if (CARRY_DEVICE_ID) PBA.insert_avp ("Device-Id"); PBA.insert_avp("PPAC"); Tx:PBA(); Authorize(); SessionTimerStart();</pre>	
<pre>EAP_FAILURE && (SEPARATE==Set) && !ppac_available()</pre>	<pre>if (key_available()) PER.insert_avp("MAC"); PER.RESULT_CODE= PANA_PPAC_CAPABILITY_ UNSUPPORTED Tx:PER(); RtxTimerStart();</pre>	<pre>WAIT_PEA</pre>
<pre>EAP_FAILURE && (SEPARATE==Set) && (PBR.exist_avp ("Protection- Capability") && !pcap_supported())</pre>	<pre>if (key_available()) PER.insert_avp("MAC"); PER.RESULT_CODE= PANA_PROTECTION_ CAPABILITY_UNSUPPORTED Tx:PER(); RtxTimerStart();</pre>	<pre>WAIT_PEA</pre>
<pre>EAP_INVALID_MSG</pre>	<pre>None();</pre>	<pre>WAIT_PAA</pre>

```
-----
State: WAIT_EAP_RESULT_CLOSE
-----
```

Exit Condition	Exit Action	Exit State
+-----+-----+-----+		
- - - - - (EAP Result) - - - - -		
<pre>EAP_SUCCESS && PBR.exist_avp ("Key-Id")</pre>	<pre>PBA.insert_avp("MAC"); PBA.insert_avp("Key-Id"); Tx:PBA(); Disconnect();</pre>	<pre>CLOSED</pre>
<pre>EAP_SUCCESS && !PBR.exist_avp ("Key-Id")</pre>	<pre>if (key_available()) PBA.insert_avp("MAC"); Tx:PBA(); Disconnect();</pre>	<pre>CLOSED</pre>
<pre>EAP_FAILURE</pre>	<pre>Tx:PBA(); Disconnect();</pre>	<pre>CLOSED</pre>

EAP_INVALID_MSG None(); WAIT_PAA

- - - - -

 State: WAIT_1ST_EAP_RESULT

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (First EAP) - - - - -		
EAP_SUCCESS && PFER.exist_avp ("Key-Id")	PFEA.insert_avp("Key-Id"); PFEA.S_flag=1; PFEA.N_flag=PFER.N_flag; PFEA.insert_avp("MAC"); Tx:PFEA(); EAP_Restart();	WAIT_PAA
(EAP_SUCCESS && !PFER.exist_avp ("Key-Id")) EAP_FAILURE	if (key_available()) PFEA.insert_avp("MAC"); PFEA.S_flag=1; PFEA.N_flag=PFER.N_flag; Tx:PFEA(); EAP_Restart();	WAIT_PAA
EAP_INVALID_MSG	EAP_Restart();	WAIT_PAA
- - - - -		

 State: WAIT_1ST_EAP_RESULT_CLOSE

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (First EAP) - - - - -		
EAP_SUCCESS && PFER.exist_avp ("Key-Id")	PFEA.insert_avp("Key-Id"); PFEA.S_flag=0; PFEA.N_flag=0; PFEA.insert_avp("MAC"); Tx:PFEA(); Disconnect();	CLOSED
(EAP_SUCCESS && !PFER.exist_avp ("Key-Id")) EAP_FAILURE	if (key_available()) PFEA.insert_avp("MAC"); PFEA.S_flag=0; PFEA.N_flag=0; Tx:PFEA(); Disconnect();	CLOSED


```

EAP_INVALID_MSG      None();                               WAIT_PAA
- - - - -

-----
State: OPEN
-----

Exit Condition      Exit Action      Exit State
-----+-----+-----
- - - - - (liveness test initiated by PAA)- - - - -
Rx:PPR              if (key_available())      OPEN
                    PPA.insert_avp("MAC");
                    Tx:PPA();
- - - - -
- - - - - (liveness test initiated by PaC)- - - - -
PANA_PING           if (key_available())      WAIT_PPA
                    PPR.insert_avp("MAC");
                    Tx:PPR();
                    RtxTimerStart();
- - - - -
- - - - - (re-authentication initiated by PaC)- - - - -
REAUTH              SEPARATE=Set|Unset;        WAIT_PRAA
                    1ST_EAP=Unset;
                    if (key_available())
                        PRAR.insert_avp("MAC");
                    Tx:PRAR();
                    RtxTimerStart();
                    SessionTimerStop();
- - - - -
- - - - - (re-authentication initiated by PAA)- - - - -
Rx:PAR &&            SEPARATE=Set|Unset;        WAIT_EAP_MSG
!eap_piggyback()    1ST_EAP=Unset;
                    EAP_RespTimerStart();
                    TxEAP();
                    if (key_available())
                        PAN.insert_avp("MAC");
                    PAN.S_flag=PAR.S_flag;
                    PAN.N_flag=PAR.N_flag;
                    Tx:PAN();
                    SessionTimerStop();

Rx:PAR &&            SEPARATE=Set|Unset;        WAIT_EAP_MSG
eap_piggyback()     1ST_EAP=Unset;
                    EAP_RespTimerStart();
                    TxEAP();
                    SessionTimerStop();
- - - - -
- - - - - (Session termination initiated by PAA) - - - - -

```



```

Rx:PTR                                if (key_available())      CLOSED
                                      PTA.insert_avp("MAC");
                                      Tx:PTA();
                                      Disconnect();
- - - - -
- - - - - (Session termination initiated by PaC) - - - - -
TERMINATE                             if (key_available())      SESS_TERM
                                      PTR.insert_avp("MAC");
                                      Tx:PTR();
                                      RtxTimerStart();
- - - - -
- - - - - (Address update) - - - - -
NOTIFY                                if (key_available())      WAIT_PUA
                                      PUR.insert_avp("MAC");
                                      Tx:PUR();
                                      RtxTimerStart();
- - - - -
- - - - - (Notification update) - - - - -
Rx:PUR                                if (key_available())      OPEN
                                      PUA.insert_avp("MAC");
                                      Tx:PUA();
- - - - -

```

```

-----
State: WAIT_PRAA
-----

```

Exit Condition	Exit Action	Exit State
- - - - - (re-authentication initiated by PaC) - - - - -		
Rx:PRAA	RtxTimerStop();	WAIT_PAA

```

-----
State: WAIT_PPA
-----

```

Exit Condition	Exit Action	Exit State
- - - - - (liveness test initiated by PAA) - - - - -		
Rx:PPA	RtxTimerStop();	OPEN

```

-----
State: WAIT_PUA
-----

```

Exit Condition	Exit Action	Exit State
----------------	-------------	------------


```
-----+-----+-----
- - - - - (PUA processing) - - - - -
Rx:PUA      RtxTimerStop();      OPEN
- - - - -
```

```
-----
State: SESS_TERM
-----
```

```
Exit Condition      Exit Action      Exit State
-----+-----+-----
- - - - - (Session termination initiated by PaC) - - - - -
Rx:PTA      Disconnect();      CLOSED
- - - - -
```

```
-----
State: WAIT_PEA
-----
```

```
Exit Condition      Exit Action      Exit State
-----+-----+-----
- - - - - (PEA processing) - - - - -
Rx:PEA      RtxTimerStop();      CLOSED
             Disconnect();
- - - - -
```


7. PAA State Machine

7.1. Interface between PAA and EAP Authenticator

The interface between a PAA and an EAP authenticator provides a mechanism to deliver EAP messages for the EAP authenticator as well as a mechanism to notify the EAP authenticator of PAA events and to receive notification of EAP authenticator events. These message delivery and event notification mechanisms occur only within context of their associated states or exit actions.

7.1.1. EAP Restart Notification from PAA to EAP Authenticator

An EAP authenticator state machine defined in [I-D.ietf-eap-statemachine] has an initialization procedure before sending the first EAP request. To initialize the EAP state machine, the PAA state machine defines an event notification mechanism to send an EAP (re)start event to the EAP peer. The event notification is done via EAP_Restart() procedure in the initialization action of the PAA state machine.

7.1.2. Delivering EAP Responses from PAA to EAP Authenticator

TxEAP() procedure in the PAA state machine serves as the mechanism to deliver EAP-Responses contained in PANA-Auth-Answer messages to the EAP authenticator. This procedure is enabled only after an EAP restart event is notified to the EAP authenticator and before any event resulting in a termination of the EAP authenticator session. In the case where the EAP authenticator follows the EAP authenticator state machines defined in [I-D.ietf-eap-statemachine], TxEAP() procedure sets eapResp variable of the EAP authenticator state machine and puts the EAP response in eapRespData variable of the EAP authenticator state machine.

7.1.3. Delivering EAP Messages from EAP Authenticator to PAA

An EAP request is delivered from the EAP authenticator to the PAA via EAP_REQUEST event variable. The event variable is set when the EAP authenticator passes the EAP request to its lower-layer. In the case where the EAP authenticator follows the EAP authenticator state machines defined in [I-D.ietf-eap-statemachine], EAP_REQUEST event variable refers to eapReq variable of the EAP authenticator state machine and the EAP request is contained in eapReqData variable of the EAP authenticator state machine.

7.1.4. EAP Authentication Result Notification from EAP Authenticator to PAA

In order for the EAP authenticator to notify the PAA of the EAP authentication result, EAP_SUCCESS, EAP_FAILURE and EAP_TIMEOUT event variables are defined. In the case where the EAP authenticator follows the EAP authenticator state machines defined in [I-D.ietf-eap-statemachine], EAP_SUCCESS, EAP_FAILURE and EAP_TIMEOUT event variables refer to eapSuccess, eapFail and eapTimeout variables of the EAP authenticator state machine, respectively. In this case, if EAP_SUCCESS event variable is set to TRUE, an EAP-Success message is contained in eapReqData variable of the EAP authenticator state machine, and additionally, eapKeyAvailable variable is set to TRUE and eapKeyData variable contains a AAA-Key if the AAA-Key is generated as a result of successful authentication by the EAP authentication method in use. Similarly, if EAP_FAILURE event variable is set to TRUE, an EAP-Failure message is contained in eapReqData variable of the EAP authenticator state machine. The PAA uses EAP_SUCCESS, EAP_FAILURE and EAP_TIMEOUT event variables as a trigger to send a PBR or a PFER message to the PaC.

7.2. Variables

USE_COOKIE

This variable indicates whether the PAA uses Cookie.

EAP_PIGGYBACK

This variable indicates whether the PAA is able to piggyback an EAP-Request in PANA-Start-Request.

SEPARATE

This variable indicates whether the PAA provides NAP/ISP separate authentication.

1ST_EAP

This variable indicates whether the 1st EAP authentication is a success, failure or yet completed.

PSA.SESSION_ID

This variable contains the Session-Id AVP value in the PANA-Start-Answer message in process.

CARRY_LIFETIME

This variable indicates whether a Session-Lifetime AVP is carried in PANA-Bind-Request message.

PROTECTION_CAP_IN_PSR

This variable indicates whether a Protection-Capability AVP is carried in a PANA-Start-Request message.

PROTECTION_CAP_IN_PBR

This variable indicates whether a Protection-Capability AVP is carried in a PANA-Bind-Request message.

CARRY_NAP_INFO

This variable indicates whether a NAP-Information AVP is carried in PANA-Start-Request message.

CARRY_ISP_INFO

This variable indicates whether an ISP-Information AVP is carried in PANA-Start-Request message.

NAP_AUTH

This variable indicates whether a NAP authentication is being performed or not.

CARRY_PPAC

This variable indicates whether a Post-PANA-Address-Configuration AVP is carried in PANA-Start-Request message.

PAC_FOUND

This variable is set to TRUE during the EP-to-PAA notification as a result of a traffic-driven PAA discovery or link-up event notification by the EP as a result of the presence of a new PaC.

EAP_SUCCESS

This event variable is set to TRUE when EAP conversation completes with success. This event accompanies an EAP- Success message passed from the EAP authenticator.

EAP_FAILURE

This event variable is set to TRUE when EAP conversation completes with failure. This event accompanies an EAP- Failure message passed from the EAP authenticator.

EAP_REQUEST

This event variable is set to TRUE when the EAP authenticator delivers an EAP Request to the PAA. This event accompanies an EAP-Request message received from the EAP authenticator.

EAP_TIMEOUT

This event variable is set to TRUE when EAP conversation times out without generating an EAP-Success or an EAP-Failure message. This event does not accompany any EAP message.

7.3. Procedures

boolean new_key_available()

A procedure to check whether the PANA session has a new PANA_MAC_KEY. If the state machine already have a PANA_MAC_KEY, it returns FALSE. If the state machine does not have a PANA_MAC_KEY, it tries to retrieve a AAA-Key from the EAP entity. If a AAA-Key has been retrieved, it computes a PANA_MAC_KEY from the AAA-Key and returns TRUE. Otherwise, it returns FALSE.

boolean new_source_address()

A procedure to check the PaC's source IP address from the current PUR message. If the source IP address of the message is different from the last known IP address stored in the PANA session, this procedure returns TRUE. Otherwise, it returns FALSE.

void update_popa()

A procedure to extract the PaC's source IP address from the current PUR message and update the PANA session with this new IP address.

7.4. PAA State Transition Table

State: OFFLINE (Initial State)

Initialization Action:

```

USE_COOKIE=Set|Unset;
EAP_PIGGYBACK=Set|Unset;
SEPARATE=Set|Unset;
if (USE_COOKIE==Unset && EAP_PIGGYBACK==Set)
    SEPARATE=Unset;
1ST_EAP=Unset;
ABORT_ON_1ST_EAP_FAILURE=Set|Unset;
CARRY_LIFETIME=Set|Unset;
CARRY_DEVICE_ID=Set|Unset;
CARRY_NAP_INFO=Set|Unset;
CARRY_ISP_INFO=Set|Unset;
CARRY_PPAC=Set|Unset;
PROTECTION_CAP_IN_PSR=Set|Unset;
PROTECTION_CAP_IN_PBR=Set|Unset;
if (PROTECTION_CAP_IN_PBR=Unset)
    PROTECTION_CAP_IN_PSR=Unset;
else
    CARRY_DEVICE_ID=Set;
NAP_AUTH=Unset;
RTX_COUNTER=0;
RtxTimerStop();

```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Stateful discovery) - - - - -		
(Rx:PDI PAC_FOUND) && USE_COOKIE==Unset && EAP_PIGGYBACK==Set	EAP_Restart();	WAIT_EAP_MSG_ IN_DISC
(Rx:PDI PAC_FOUND) && USE_COOKIE==Unset && EAP_PIGGYBACK==Unset	if (SEPARATE==Set) PSR.S_flag=1; if (CARRY_NAP_INFO==Set) PSR.insert_avp ("NAP-Information"); if (CARRY_ISP_INFO==Set) PSR.insert_avp ("ISP-Information"); if (CARRY_PPAC==Set) PSR.insert_avp ("Post-PANA-Address- Configuration"); if (PROTECTION_CAP_IN_PSR ==Set) PSR.insert_avp ("Protection-Cap.");	STATEFUL_DISC


```

Tx:PSR();
RtxTimerStart();
- - - - -
- - - - - (Stateless discovery) - - - - -
(Rx:PDI ||
PAC_FOUND) &&
USE_COOKIE==Set
if (SEPARATE==Set)
    PSR.S_flag=1;
    PSR.insert_avp
    ("Cookie");
    if (CARRY_NAP_INFO==Set)
        PSR.insert_avp
        ("NAP-Information");
    if (CARRY_ISP_INFO==Set)
        PSR.insert_avp
        ("ISP-Information");
    if (CARRY_PPAC==Set)
        PSR.insert_avp
        ("Post-PANA-Address-
        Configuration");
    if (PROTECTION_CAP_IN_PSR
        ==Set)
        PSR.insert_avp
        ("Protection-Cap.");
Tx:PSR();
- - - - -
- - - - - (PSA processing) - - - - -
Rx:PSA &&
USE_COOKIE==Set
if (SEPARATE==Set &&
    PSA.S_flag==0)
    SEPARATE=Unset;
    if (SEPARATE==Set)
        NAP_AUTH=Set|Unset;
    EAP_Restart();
- - - - -

```

```

-----
State: WAIT_EAP_MSG_IN_DISC
-----

```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Send PSR with EAP-Request) - - - - -		
EAP_REQUEST	PSR.insert_avp ("EAP-Payload"); if (CARRY_NAP_INFO==Set) PSR.insert_avp ("NAP-Information"); if (CARRY_ISP_INFO==Set) PSR.insert_avp ("ISP-Information");	STATEFUL_DISC


```

    if (CARRY_PPAC==Set)
        PSR.insert_avp
            ("Post-PANA-Address-
            Configuration");
    Tx:PSR();
    RtxTimerStart();

```

```

EAP_TIMEOUT          None();          OFFLINE
-----

```

```

-----
State: STATEFUL_DISC
-----

```

Exit Condition	Action	Exit State
	(Stateful discovery)-	
Rx:PSA	<pre> if (SEPARATE==Set && PSA.S_flag==0) SEPARATE=Unset; if (PSA.exist_avp ("EAP-Payload")) TxEAP(); else { if (SEPARATE==Set) NAP_AUTH=Set Unset; EAP_Restart(); } RtxTimerStop(); </pre>	WAIT_EAP_MSG
EAP_TIMEOUT	<pre> if (key_available()) PER.insert_avp("MAC"); Tx:PER(); RtxTimerStart(); </pre>	WAIT_PEA

```

-----
State: WAIT_EAP_MSG
-----

```

Exit Condition	Exit Action	Exit State
	(Receiving EAP-Request)-	
EAP_REQUEST	<pre> if (key_available()) PAR.insert_avp("MAC"); if (SEPARATE==Set) { </pre>	WAIT_PAN_OR_PAR


```

        PAR.S_flag=1;
        if (NAP_AUTH==Set)
            PAR.N_flag=1;
    }
    Tx:PAR();
    RtxTimerStart();
- - - - -
- - - - - (Receiving EAP-Success/Failure single EAP) - - - - -
EAP_FAILURE &&                PBR.insert_avp                WAIT_FAIL_PBA
1ST_EAP==Unset &&            ("EAP-Payload");
SEPARATE==Unset            if (key_available())
                            PBR.insert_avp("MAC");
                            Tx:PBR();
                            RtxTimerStart();

EAP_SUCCESS &&                PBR.insert_avp                WAIT_SUCC_PBA
1ST_EAP==Unset &&            ("EAP-Payload");
SEPARATE==Unset &&            if (CARRY_DEVICE_ID==Set)
Authorize()                PBR.insert_avp
                            ("Device-Id");
                            if (CARRY_LIFETIME==Set)
                                PBR.insert_avp
                                ("Session-Lifetime");
                            if (PROTECTION_CAP_IN_PBR
                                ==Set)
                                PBR.insert_avp
                                ("Protection-Cap.");
                            if (new_key_available())
                                PBR.insert_avp
                                ("Key-Id");
                            if (key_available())
                                PBR.insert_avp("MAC");
                            Tx:PBR();
                            RtxTimerStart();

EAP_SUCCESS &&                PBR.insert_avp                WAIT_FAIL_PBA
1ST_EAP==Unset &&            ("EAP-Payload");
SEPARATE==Unset &&            if (new_key_available())
!Authorize()                PBR.insert_avp
                            ("Key-Id");
                            if (key_available())
                                PBR.insert_avp("MAC");
                            Tx:PBR();
                            RtxTimerStart();

EAP_TIMEOUT &&                if (key_available())                WAIT_PEA
1ST_EAP==Unset &&            PER.insert_avp("MAC");
SEPARATE==Unset            Tx:PER();

```



```

                                RtxTimerStart();

- - - - - (Receiving EAP-Success/Failure for 1st EAP)- - - - -
EAP_FAILURE &&                  1ST_EAP=Failure                WAIT_PFEA
1ST_EAP==Unset &&              PFER.insert_avp
SEPARATE==Set &&                ("EAP-Payload");
ABORT_ON_1ST_EAP_FAILURE if (key_available())
==Unset                        PFER.insert_avp("MAC");
                                PFER.S_flag=1;
                                if (NAP_AUTH)
                                    PFER.N_flag=1;
                                Tx:PFER();
                                RtxTimerStart();

EAP_FAILURE &&                  1ST_EAP=Failure                WAIT_FAIL_PFEA
1ST_EAP==Unset &&              PFER.insert_avp
SEPARATE==Set &&                ("EAP-Payload");
ABORT_ON_1ST_EAP_FAILURE if (key_available())
==Set                          PFER.insert_avp("MAC");
                                PFER.S_flag=0;
                                Tx:PFER();
                                RtxTimerStart();

EAP_SUCCESS &&                 1ST_EAP=Success                WAIT_PFEA
1ST_EAP==Unset &&              PFER.insert_avp
SEPARATE==Set                  ("EAP-Payload");
                                if (new_key_available())
                                    PFER.insert_avp
                                    ("Key-Id");
                                if (key_available())
                                    PFER.insert_avp("MAC");
                                PFER.S_flag=1;
                                if (NAP_AUTH)
                                    PFER.N_flag=1;
                                Tx:PFER();
                                RtxTimerStart();

EAP_TIMEOUT &&                 1ST_EAP=Failure                WAIT_PFEA
1ST_EAP==Unset &&              if (key_available())
SEPARATE==Set &&                PFER.insert_avp("MAC");
ABORT_ON_1ST_EAP_FAILURE PFER.S_flag=1;
==Unset                        if (NAP_AUTH)
                                PFER.N_flag=1;
                                Tx:PFER();
                                RtxTimerStart();

EAP_TIMEOUT &&                 1ST_EAP=Failure                WAIT_FAIL_PFEA

```



```

1ST_EAP==Unset &&          if (key_available())
SEPARATE==Set &&           PFER.insert_avp("MAC");
ABORT_ON_1ST_EAP_FAILURE SEPARATE=Unset;
==Set                     PFER.S_flag=0;
                           Tx:PFER();
                           RtxTimerStart();

- - - - -
- - - - - (Receiving EAP-Success/Failure for 2nd EAP)- - - - -
EAP_FAILURE &&             PBR.insert_avp          WAIT_FAIL_PBA
                           ("EAP-Payload");
1ST_EAP==Failure &&       if (key_available())
SEPARATE==Set             PBR.insert_avp("MAC");
                           PBR.S_flag=1;
                           if (NAP_AUTH)
                               PBR.N_flag=1;
                           Tx:PBR();
                           RtxTimerStart();

EAP_FAILURE &&             PBR.insert_avp          WAIT_SUCC_PBA
1ST_EAP==Success &&       ("EAP-Payload");
SEPARATE==Set &&         if (CARRY_DEVICE_ID==Set)
Authorize()              PBR.insert_avp
                           ("Device-Id");
                           if (CARRY_LIFETIME==Set)
                               PBR.insert_avp
                                   ("Session-Lifetime");
                           if (PROTECTION_CAP_IN_PBR
                               ==Set)
                               PBR.insert_avp
                                   ("Protection-Cap.");
                           if (key_available())
                               PBR.insert_avp("MAC");
                           PBR.S_flag=1;
                           if (NAP_AUTH)
                               PBR.N_flag=1;
                           Tx:PBR();
                           RtxTimerStart();

EAP_FAILURE &&             PBR.insert_avp          WAIT_FAIL_PBA
1ST_EAP==Success &&       ("EAP-Payload");
SEPARATE==Set &&         if (key_available())
!Authorize()             PBR.insert_avp("MAC");
                           PBR.S_flag=1;
                           if (NAP_AUTH)
                               PBR.N_flag=1;
                           Tx:PBR();
                           RtxTimerStart();

```


EAP_SUCCESS && 1ST_EAP==Success && SEPARATE==Set && Authorize()	PBR.insert_avp ("EAP-Payload"); if (CARRY_DEVICE_ID==Set) PBR.insert_avp ("Device-Id"); if (CARRY_LIFETIME==Set) PBR.insert_avp ("Session-Lifetime"); if (PROTECTION_CAP_IN_PBR ==Set) PBR.insert_avp ("Protection-Cap."); if (new_key_available()) PBR.insert_avp ("Key-Id"); if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; if (NAP_AUTH) PBR.N_flag=1; Tx:PBR(); RtxTimerStart();	WAIT_SUCC_PBA
EAP_SUCCESS && 1ST_EAP==Success && SEPARATE==Set && !Authorize()	PBR.insert_avp ("EAP-Payload"); if (new_key_available()) PBR.insert_avp ("Key-Id"); if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; if (NAP_AUTH) PBR.N_flag=1; Tx:PBR(); RtxTimerStart();	WAIT_FAIL_PBA
EAP_SUCCESS && 1ST_EAP==Failure && SEPARATE==Set && Authorize()	PBR.insert_avp ("EAP-Payload"); if (CARRY_DEVICE_ID==Set) PBR.insert_avp ("Device-Id"); if (CARRY_LIFETIME==Set) PBR.insert_avp ("Session-Lifetime"); if (PROTECTION_CAP_IN_PBR ==Set) PBR.insert_avp ("Protection-Cap.");	WAIT_SUCC_PBA

	<pre> if (new_key_available()) PBR.insert_avp ("Key-Id"); if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; if (NAP_AUTH) PBR.N_flag=1; Tx:PBR(); RtxTimerStart(); </pre>	
<pre> EAP_SUCCESS && 1ST_EAP==Failure && SEPARATE==Set && !Authorize() </pre>	<pre> PBR.insert_avp ("EAP-Payload"); if (new_key_available()) PBR.insert_avp ("Key-Id"); if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; if (NAP_AUTH) PBR.N_flag=1; Tx:PBR(); RtxTimerStart(); </pre>	WAIT_FAIL_PBA
<pre> EAP_TIMEOUT && 1ST_EAP==Failure && SEPARATE==Set </pre>	<pre> if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; if (NAP_AUTH) PBR.N_flag=1; Tx:PBR(); RtxTimerStart(); </pre>	WAIT_FAIL_PBA
<pre> EAP_TIMEOUT && 1ST_EAP==Success && SEPARATE==Set && Authorize() </pre>	<pre> if (CARRY_DEVICE_ID==Set) PBR.insert_avp ("Device-Id"); if (CARRY_LIFETIME==Set) PBR.insert_avp ("Session-Lifetime"); if (PROTECTION_CAP_IN_PBR ==Set) PBR.insert_avp ("Protection-Cap."); if (new_key_available()) PBR.insert_avp ("Key-Id"); if (key_available()) PBR.insert_avp("MAC"); PBR.S_flag=1; </pre>	WAIT_SUCC_PBA


```

    if (NAP_AUTH)
        PBR.N_flag=1;
    Tx:PBR();
    RtxTimerStart();

```

```

EAP_TIMEOUT &&
1ST_EAP==Success &&
SEPARATE==Set &&
!Authorize()
    if (key_available())
        PBR.insert_avp("MAC");
    PBR.S_flag=1;
    if (NAP_AUTH)
        PBR.N_flag=1;
    Tx:PBR();
    RtxTimerStart();

```

```

WAIT_FAIL_PBA

```

```

-----
State: WAIT_PFEA
-----

```

Event/Condition	Action	Exit State
-----+-----+-----		
- - - - - (PFEA Processing) - - - - -		
Rx:PFEA && (1ST_EAP==Success (PFEA.S_flag==1 && 1ST_EAP==Failure))	RtxTimerStop(); EAP_Restart(); if (NAP_AUTH==Set) NAP_AUTH=Unset; else NAP_AUTH=Set;	WAIT_EAP_MSG
Rx:PFEA && PFEA.S_flag==0 && 1ST_EAP==Failure	RtxTimerStop(); Disconnect();	CLOSED

```

-----
State: WAIT_FAIL_PFEA
-----

```

Event/Condition	Action	Exit State
-----+-----+-----		
- - - - - (PFEA Processing) - - - - -		
Rx:PFEA	RtxTimerStop(); Disconnect();	CLOSED

```

-----
State: WAIT_SUCC_PBA
-----

```


Event/Condition	Action	Exit State
-----+-----+-----		
- - - - - (PBA Processing) - - - - -		
Rx:PBA && (CARRY_DEVICE_ID==Unset (CARRY_DEVICE_ID==Set && PBA.exit_avp("Device-Id")))	SessionTimerStart();	OPEN
Rx:PBA && CARRY_DEVICE_ID==Set && !PBA.exit_avp ("Device-Id")	PER.RESULT_CODE= PANA_MISSING_AVP Tx:PER(); RtxTimerStart();	WAIT_PEA

State: WAIT_FAIL_PBA

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PBA Processing) - - - - -		
Rx:PBA	RtxTimerStop(); Disconnect();	CLOSED

State: OPEN

Event/Condition	Action	Exit State
-----+-----+-----		
- - - - - (re-authentication initiated by PaC) - - - - -		
Rx:PRAR	if (key_available()) PRAA.insert_avp("MAC"); EAP_Restart(); 1ST_EAP=Unset; NAP_AUTH=Set Unset; Tx:PRAA(); SessionTimerStop();	WAIT_EAP_MSG

- - - - - (re-authentication initiated by PAA) - - - - -		
REAUTH	EAP_Restart(); 1ST_EAP=Unset; NAP_AUTH=Set Unset; SessionTimerStop();	WAIT_EAP_MSG

- - (liveness test based on PPR-PPA exchange initiated by PAA) -		
PANA_PING	Tx:PPR();	WAIT_PPA


```

                                RtxTimerStart();
- - - - -
- - (liveness test based on PPR-PPA exchange initiated by PaC)-
Rx:PPR                                if (key_available())      OPEN
                                      PPA.insert_avp("MAC");
                                      Tx:PPA();
- - - - -
- - - - - (Session termination initiated from PAA) - - - - -
TERMINATE                            if (key_available())      SESS_TERM
                                      PTR.insert_avp("MAC");
                                      Tx:PTR();
                                      RtxTimerStart();
- - - - -
- - - - - (Session termination initiated from PaC) - - - - -
Rx:PTR                                if (key_available())      CLOSED
                                      PTA.insert_avp("MAC");
                                      Tx:PTA();
                                      Disconnect();
- - - - -
- - - - - (Notification message) - - - - -
NOTIFY                                if (key_available())      WAIT_PUA
                                      PUR.insert_avp("MAC");
                                      Tx:PUR();
                                      RtxTimerStart();
- - - - -
- - - - - (Notification/Address update) - - - - -
Rx:PUR                                If (key_avaialble())      OPEN
                                      PUA.insert_avp("MAC");
                                      Tx:PUA();
                                      if (new_source_address())
                                          update_popa();
- - - - -

```

```

-----
State: WAIT_PPA
-----

```

Exit Condition	Exit Action	Exit State
- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -
Rx:PPA	RtxTimerStop();	OPEN
- - - - -	- - - - -	- - - - -

```

-----
State: WAIT_PAN_OR_PAR
-----

```


Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Pass EAP Response to the EAP authenticator)- - - -		
Rx: PAN && PAN.exist_avp ("EAP-Payload")	TxEAP();	WAIT_EAP_MSG
Rx: PAR	TxEAP(); if (key_available()) PAN.insert_avp("MAC"); if (SEPARATE==Set) { PAN.S_flag=1; if (NAP_AUTH==Set) PAN.N_flag=1; } RtxTimerStop(); Tx: PAN();	WAIT_EAP_MSG
- - - - - (PAN without an EAP response) - - - - -		
Rx: PAN && !PAN.exist_avp ("EAP-Payload")	RtxTimerStop();	WAIT_PAN_OR_PAR
- - - - - (EAP retransmission) - - - - -		
EAP_REQUEST	if (key_available()) PAR.insert_avp("MAC"); if (SEPARATE==Set) { PAR.S_flag=1; if (NAP_AUTH==Set) PAR.N_flag=1; } Tx: PAR(); RtxTimerStart();	WAIT_PAN_OR_PAR
- - - - - (EAP authentication timeout)- - - - -		
EAP_TIMEOUT && 1ST_EAP==Unset && SEPARATE==Unset	if (key_available()) PER.insert_avp("MAC"); Tx: PER(); RtxTimerStart();	WAIT_PEA
- - - - - (EAP authentication timeout for 1st EAP)- - - - -		
EAP_TIMEOUT && 1ST_EAP==Unset && SEPARATE==Set && ABORT_ON_1ST_EAP_FAILURE ==Unset	1ST_EAP=Failure if (key_available()) PFER.insert_avp("MAC"); PFER.S_flag=1; if (NAP_AUTH) PFER.N_flag=1; Tx: PFER();	WAIT_PFEA


```

                                RtxTimerStart();

EAP_TIMEOUT &&                1ST_EAP=Failure                WAIT_FAIL_PFEA
1ST_EAP==Unset &&            if (key_available())
SEPARATE==Set &&                PFER.insert_avp("MAC");
ABORT_ON_1ST_EAP_FAILURE SEPARATE=Unset;
==Set                        PFER.S_flag=0;
                                Tx:PFER();
                                RtxTimerStart();

- - - - - (EAP authentication timeout for 2nd EAP) - - - - -
EAP_TIMEOUT &&                if (key_available())                WAIT_FAIL_PBA
1ST_EAP==Failure &&            PBR.insert_avp("MAC");
SEPARATE==Set                PBR.S_flag=1;
                                if (NAP_AUTH)
                                    PBR.N_flag=1;
                                Tx:PBR();
                                RtxTimerStart();

EAP_TIMEOUT &&                if (CARRY_DEVICE_ID==Set) WAIT_SUCC_PBA
1ST_EAP==Success &&            PBR.insert_avp
SEPARATE==Set &&                ("Device-Id");
Authorize()                  if (CARRY_LIFETIME==Set)
                                PBR.insert_avp
                                    ("Session-Lifetime");
                                if (PROTECTION_CAP_IN_PBR
                                    ==Set)
                                    PBR.insert_avp
                                        ("Protection-Cap.");
                                if (new_key_available())
                                    PBR.insert_avp
                                        ("Key-Id");
                                if (key_available())
                                    PBR.insert_avp("MAC");
                                PBR.S_flag=1;
                                if (NAP_AUTH)
                                    PBR.N_flag=1;
                                Tx:PBR();
                                RtxTimerStart();

EAP_TIMEOUT &&                if (key_available())                WAIT_FAIL_PBA
1ST_EAP==Success &&            PBR.insert_avp("MAC");
SEPARATE==Set &&            PBR.S_flag=1;
!Authorize()                  if (NAP_AUTH)
                                PBR.N_flag=1;
                                Tx:PBR();
                                RtxTimerStart();

```


- - - - -

State: WAIT_PUA

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - -	- (PUA processing)-	- - - - -
Rx:PUA	RtxTimerStop();	OPEN
- - - - -		- - - - -

State: SESS_TERM

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - -	- (PTA processing) -	- - - - -
Rx:PTA	RtxTimerStop(); Disconnect();	CLOSED
- - - - -		- - - - -

State: WAIT_PEA

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - -	- (PEA processing) -	- - - - -
Rx:PEA	RtxTimerStop(); Disconnect();	CLOSED
- - - - -		- - - - -

8. Mobility Optimization Support

The state machines outlined in preceeding sections provide only PANA base protocol functionality. In order to support PANA mobility optimization outlined in [[I-D.ietf-pana-mobopts](#)], additions and changes to the PaC and PAA state machines is required. The additions and changes provides only basic mobility optimization and is not explicit on integration of other mobility functionality such as context-transfer mechanisms. However, it does provide enough flexibility to accomodate future inclusion of such mechanisms.

The variables, procedures and state transition described in this section is designed to be seamlessly integrated into the appropriate base protocol state machines. They should be treated as a mobility optimization addendum to the base protocol state machine. In this addendum, no additional states has been defined but some modifications to the base protocol state machine is required. The modifications are to accomodate the mobility variables and procedures as they relate to existing state transition actions and events. These modifications to existing state transition are noted in state transition tables in this section. These modified state transitions are intended to replace thier base protocol counterpart. Addition of new state transitions specific to mobility optimization is also present. Variable initialization also need to be added to the appropriate base protocol state to complete the mobility optimization support.

8.1. Common Variables

MOBILITY

This variable indicates whether the mobility handling feature described in [[I-D.ietf-pana-mobopts](#)] is supported. This should be present in both PaC and PAA state machine. Existing state transitions in the base protocol state machine that can be affected by mobility optimization must treat this variable as being Unset unless the state transitions is explicitly redefined in this section.

8.2. PaC Mobility Optimization State Machine

8.2.1. Variables

PANA_SA_RESUMED

This variable indicates whether the PANA SA of a previous PANA session was resumed during the discovery and initial handshake.

8.2.2. Procedures

boolean resume_pana_sa()

This procedure returns TRUE when a PANA SA for a previously established PANA Session is resumed, otherwise returns FALSE. Once a PANA SA is resumed, key_available() procedure must return TRUE. Existing state transitions in the base protocol state machine that can be affected by mobility optimization must assume that this procedure always returns FALSE unless the state transition is explicitly redefined in this section.

8.2.3. PaC Mobility Optimization State Transition Table Addendum

State: OFFLINE (Initial State)

Initialization Action:

MOBILITY=Set|Unset;
PANA_SA_RESUMED=Unset;

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PSR processing with mobility support)- - - - -		
- The following state transitions are intended to be added -		
- to the OFFLINE state of the PaC base protocol state -		
- machine. -		

Rx:PSR && !PSR.exist_avp ("EAP-Payload") && MOBILITY==Set && resume_pana_sa() && PSR.exist_avp ("Cookie")	RtxTimerStop(); PSA.insert_avp ("Session-Id"); SEPARATE=Unset; PANA_SA_RESUMED=Set; PSA.insert_avp("Cookie"); PSA.insert_avp("MAC"); Tx:PSA(); RtxTimerStart();	WAIT_PAA
Rx:PSR && !PSR.exist_avp ("EAP-Payload") && MOBILITY==Set && resume_pana_sa() &&	RtxTimerStop(); PSA.insert_avp ("Session-Id"); PSA.insert_avp("MAC"); Tx:PSA();	WAIT_PAA


```
!PSR.exist_avp          PANA_SA_RESUMED=Set;
("Cookie")
```

```
-----
State: WAIT_PAA
-----
```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PAR-PAN exchange) - - - - -		
- The following state transitions are intended to replace		-
- existing base protocol state transitions. Original base		-
- protocol state transitions can be referenced by the same		-
- exit conditions that exist in the WAIT_PAA state of the PaC		-
- base protocol state machine.		-
- - - - -		

Rx:PAR && !eap_piggyback()	RtxTimerStop(); TxEAP(); PANA_SA_RESUMED=Unset; EAP_RespTimerStart(); if (key_available()) PAN.insert_avp("MAC"); PAN.S_flag=PAR.S_flag; PAN.N_flag=PAR.N_flag; Tx:PAN();	WAIT_EAP_MSG
-------------------------------	---	--------------

Rx:PAR && eap_piggyback()	RtxTimerStop(); TxEAP(); PANA_SA_RESUMED=Unset; EAP_RespTimerStart();	WAIT_EAP_MSG
------------------------------	--	--------------

- - - - - (1st EAP result) - - - - -		
- The following state transitions are intended to replace		-
- existing base protocol state transitions. Original base		-
- protocol state transitions can be referenced by exit		-
- conditions that excludes PANA_SA_RESUMED variable checks.		-
- - - - -		

Rx:PBR && 1ST_EAP==Unset && SEPARATE==Unset && PBR.RESULT_CODE== PANA_SUCCESS && PANA_SA_RESUMED!=Set	TxEAP(); if (PBR.exist_avp ("Device-Id")) CARRY_DEVICE_ID=Set;	WAIT_EAP_RESULT
--	---	-----------------

- - - - - (PBR processing with mobility support)- - - - -		
- The following state transitions are intended to be added		-

- to the WAIT_PAA state of the PaC base protocol state machine.
-

```

- - - - -
Rx:PBR &&                                PBA.insert_avp("Key-Id"); OPEN
1ST_EAP==Unset &&                        PBA.insert_avp("MAC");
SEPARATE==Unset &&                        if (PBR.exist_avp
PBR.RESULT_CODE==                        ("Device-Id"))
    PANA_SUCCESS &&                    PBA.insert("Device-Id");
PANA_SA_RESUMED==Set && Tx:PBA();
PBR.exist_avp                          Authorize();
("Key-Id") &&                          SessionTimerStart();
PBR.exist_avp
("MAC")

```

```

-----
State: OPEN
-----

```

Exit Condition	Exit Action	Exit State
----------------	-------------	------------

```

-----+-----+-----
- - - - - (re-authentication initiated by PaC)- - - - -
- The following state transitions are intended to replace
- existing base protocol state transitions. Original base
- protocol state transitions can be referenced by the same
- exit conditions that exist in the OPEN state of the PaC
- base protocol state machine.
- - - - -

```

REAUTH	SEPARATE=Set Unset; 1ST_EAP=Unset; PANA_SA_RESUMED=Unset; if (key_available()) PRAR.insert_avp("MAC"); Tx:PRAR(); RtxTimerStart(); SessionTimerStop();	WAIT_PRAA
--------	---	-----------

```

- - - - - (re-authentication initiated by PAA)- - - - -

```

Rx:PAR && !eap_piggyback()	SEPARATE=Set Unset; 1ST_EAP=Unset; PANA_SA_RESUMED=Unset; EAP_RespTimerStart(); TxEAP(); if (key_available()) PAN.insert_avp("MAC"); PAN.S_flag=PAR.S_flag; PAN.N_flag=PAR.N_flag; Tx:PAN(); SessionTimerStop();	WAIT_EAP_MSG
-------------------------------	--	--------------


```

Rx:PAR &&                               SEPARATE=Set|Unset;           WAIT_EAP_MSG
eap_piggyback()                          1ST_EAP=Unset;
                                          PANA_SA_RESUMED=Unset;
                                          EAP_RespTimerStart();
                                          TxEAP();
                                          SessionTimerStop();

```

```

-----+-----+-----
- - - - - (PSR processing with mobility support)- - - - -
- The following state transitions are intended to be added -
- to the OPEN state of the PaC base protocol state machine -
-----

```

```

Rx:PSR &&                                RtxTimerStop();           WAIT_PAA
!PSR.exist_avp                           PSA.insert_avp
("EAP-Payload") &&                       ("Session-Id");
MOBILITY==Set &&                         SEPARATE=Unset;
resume_pana_sa() &&                     PANA_SA_RESUMED=Set;
PSR.exist_avp                           PSA.insert_avp("Cookie");
("Cookie")                             PSA.insert_avp("MAC");
                                          Tx:PSA();
                                          RtxTimerStart();

```

```

Rx:PSR &&                                RtxTimerStop();           WAIT_PAA
!PSR.exist_avp                           PSA.insert_avp
("EAP-Payload") &&                       ("Session-Id");
MOBILITY==Set &&                         PSA.insert_avp("MAC");
resume_pana_sa() &&                     Tx:PSA();
!PSR.exist_avp                           PANA_SA_RESUMED=Set;
("Cookie")

```

8.3. PAA Mobility Optimization

8.3.1. Procedures

```
boolean retrieve_pana_sa(Session-Id)
```

This procedure returns TRUE when a PANA SA for the PANA Session corresponds to the specified Session-Id has been retrieved, otherwise returns FALSE.

8.3.2. PAA Mobility Optimization State Transition Table Addendum

 State: OFFLINE (Initial State)

Initialization Action:

MOBILITY=Set|Unset;

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PSA processing with mobility support) - - - - -		
- The following state transitions are intended to replace		-
- existing base protocol state transitions. Original base		-
- protocol state transitions can be referenced by exit		-
- conditions that excludes MOBILITY variable checks and		-
- retrieve_pana_sa() procedure calls.		-
- - - - -		
Rx:PSA && (!PSA.exist_avp ("Session-Id") MOBILITY==Unset (MOBILITY==Set && !retrieve_pana_sa (PSA.SESSION_ID)))	if (SEPARATE==Set && PSA.S_flag==0) SEPARATE=Unset; if (SEPARATE==Set) NAP_AUTH=Set Unset; EAP_Restart();	WAIT_EAP_MSG
- - - - -		
- - - - - (PSA processing with mobility support) - - - - -		
Rx:PSA && PSA.exist_avp ("Session-Id") && MOBILITY==Set && retrieve_pana_sa (PSA.SESSION_ID)	PBR.insert_avp("MAC"); PBR.insert_avp("Key-Id"); if (CARRY_DEVICE_ID==Set) PBR.insert_avp ("Device-Id"); if (PROTECTION_CAP_IN_PBR ==Set) PBR.insert_avp ("Protection-Cap."); Tx:PBR(); RtxTimerStart();	WAIT_SUCC_PBA
- - - - -		

9. Implementation Considerations

9.1. PAA and PaC Interface to Service Management Entity

In general, it is assumed in each device that has a PANA protocol stack that there is a Service Management Entity (SME) that manages the PANA protocol stack. It is recommended that a generic interface (i.e., the SME-PANA interface) between the SME and the PANA protocol stack be provided by the implementation. Especially, common procedures such as startup, shutdown, re-authenticate signals and provisions for extracting keying material should be provided by such an interface. The SME-PANA interface in a PAA device should also provide a method for communicating filtering parameters to the EP(s). When cryptographic filtering is used, the filtering parameters include keying material used for bootstrapping per-packet ciphering. When a PAA device interacts with the backend authentication server using a AAA protocol, its SME may also have an interface to the AAA protocol to obtain authorization parameters such as the authorization lifetime and additional filtering parameters.

9.2. Multicast Traffic

In general, binding a UDP socket to a multicast address and/or port is system dependent. In most systems, a socket can be bound to any address and a specific port. This allows the socket to receive all packets destined for the local host (on all it's local addresses) for that port. If the host subscribes to a multicast addresses then this socket will also receive multicast traffic as well. In some systems, this would also result in the socket receiving all multicast traffic even though it has subscribed to only one multicast address. This is because most physical interfaces has either multicast traffic enabled or disabled and does not provide specific address filtering. Normally, it is not possible to filter out specific traffic on a socket from the user level. Most environments provides lower layer filtering that allows the use of only one socket to receive both unicast and specific multicast address. However it might introduce portability problems.

10. Security Considerations

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

11. IANA Considerations

This document has no actions for IANA.

[12.](#) Acknowledgments

This work was started from state machines originally made by Dan Forsberg.

13. References

13.1. Normative References

- [I-D.ietf-pana-pana]
Forsberg, D., "Protocol for Carrying Authentication for Network Access (PANA)", [draft-ietf-pana-pana-10](#) (work in progress), July 2005.
- [I-D.ietf-eap-statemachine]
Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", [draft-ietf-eap-statemachine-06](#) (work in progress), December 2004.
- [I-D.ietf-pana-mobopts]
Forsberg, D., "PANA Mobility Optimizations", [draft-ietf-pana-mobopts-00](#) (work in progress), January 2005.

13.2. Informative References

- [RFC4058] Yegin, A., Ohba, Y., Penno, R., Tsirtsis, G., and C. Wang, "Protocol for Carrying Authentication for Network Access (PANA) Requirements", [RFC 4058](#), May 2005.
- [I-D.ietf-pana-snmpp]
Mghazli, Y., "SNMP usage for PAA-EP interface", [draft-ietf-pana-snmpp-04](#) (work in progress), July 2005.

Authors' Addresses

Victor Fajardo
Toshiba America Research, Inc.
1 Telcordia Drive
Piscataway, NJ 08854
USA

Phone: +1 732 699 5368
Email: vfajardo@tari.toshiba.com

Yoshihiro Ohba
Toshiba America Research, Inc.
1 Telcordia Drive
Piscataway, NJ 08854
USA

Phone: +1 732 699 5305
Email: yohba@tari.toshiba.com

Rafa Marin Lopez
University of Murcia
30071 Murcia
Spain

Email: rafa@dif.um.es

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights 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; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat 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 implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

