

Network Working Group
Request for Comments: 5609
Category: Informational

V. Fajardo, Ed.
Telcordia Technologies
Y. Ohba
Toshiba
R. Marin-Lopez
Univ. of Murcia
August 2009

State Machines for
the Protocol for Carrying Authentication for Network Access (PANA)

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 with the Extensible Authentication Protocol (EAP) state machines. The state machines and associated models are informative only. Implementations may achieve the same results using different methods.

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (<http://trustee.ietf.org/license-info>). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Interface between PANA and EAP	3
4. Document Authority	5
5. Notations	5
6. Common Rules	6
6.1. Common Procedures	6
6.2. Common Variables	9
6.3. Configurable Values	10
6.4. Common Message Initialization Rules	10
6.5. Common Retransmission Rules	10
6.6. Common State Transitions	11
7. PaC State Machine	12
7.1. Interface between PaC and EAP Peer	12
7.1.1. Delivering EAP Messages from PaC to EAP Peer	12
7.1.2. Delivering EAP Messages from EAP Peer to PaC	12
7.1.3. EAP Restart Notification from PaC to EAP Peer	13
7.1.4. EAP Authentication Result Notification from EAP Peer to PaC	13
7.1.5. Alternate Failure Notification from PaC to EAP Peer	13
7.2. Configurable Values	13
7.3. Variables	14
7.4. Procedures	15
7.5. PaC State Transition Table	15
8. PAA State Machine	21
8.1. Interface between PAA and EAP Authenticator	21
8.1.1. EAP Restart Notification from PAA to EAP Authenticator	21
8.1.2. Delivering EAP Responses from PAA to EAP Authenticator	22
8.1.3. Delivering EAP Messages from EAP Authenticator to PAA	22
8.1.4. EAP Authentication Result Notification from EAP Authenticator to PAA	22
8.2. Variables	23
8.3. Procedures	24
8.4. PAA State Transition Table	24
9. Implementation Considerations	29
9.1. PAA and PaC Interface to Service Management Entity	29
10. Security Considerations	29
11. Acknowledgments	29
12. References	29
12.1. Normative References	29
12.2. Informative References	30

1. Introduction

This document defines the state machines for the Protocol for Carrying Authentication for Network Access (PANA) [RFC5191]. 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. The state machines and associated models described in this document are informative only. Implementations may achieve similar results using different models and/or methods.

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. 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 [RFC4137]. Nevertheless, PANA state machines presented here have been coordinated with state machines shown by [RFC4137].

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. Terminology

This document reuses the terminology used in [RFC5191].

3. 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 interacts with an EAP state machine.

Two state machines are defined in this document: the PaC state machine (see Section 7) and the PAA state machine (see Section 8). 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.

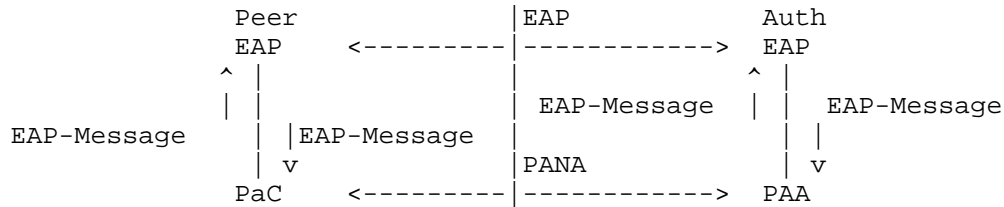


Figure 1: Interface between PANA and EAP

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 and PAA state machines present EAP messages to the EAP peer and authenticator state machines through the interface, respectively. The EAP peer and authenticator state machines process these messages and send EAP messages through the PaC and PAA state machines that are responsible for actually transmitting this message, respectively.

For example, [RFC4137] 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 [RFC4137] 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 [RFC4137], 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.

4. Document Authority

This document is intended to comply with the technical contents of any of the related documents ([RFC5191] and [RFC4137]). When there is a discrepancy, the related documents are considered authoritative and they take precedence over this document.

5. Notations

The following state transition tables are completed mostly based on the conventions specified in [RFC4137]. 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 any state in each state machine except those explicitly enumerated as exception states. The exit conditions of a wildcard state are evaluated after all other exit conditions 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. (Note that the procedures defined in [RFC4137] 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 executes 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 [RFC4137].

6. 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., "PCI", "PAR", "PAN", "PTR", "PTA", "PNR", "PNA".

6.1. 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. In the case that a non-key-generating EAP method is used but a PANA SA is required after successful authentication (generate_pana_sa() returns TRUE), Authorize() procedure must return FALSE.

```
void Tx:PANA_MESSAGE_NAME[flag](AVPs)
```

A procedure to send a PANA message to its peering PANA entity. The "flag" argument contains one or more flags (e.g., Tx:PAR[C]) to be set to the message, except for 'R' (Request) flag. The "AVPs" contains a list of names of optional AVPs to be inserted in the message, except for AUTH AVP.

This procedure includes the following action before actual transmission:

```
    if (flag==S)
        PANA_MESSAGE_NAME.S_flag=Set;
    if (flag==C)
        PANA_MESSAGE_NAME.C_flag=Set;
    if (flag==A)
        PANA_MESSAGE_NAME.A_flag=Set;
    if (flag==P)
        PANA_MESSAGE_NAME.P_flag=Set;
    PANA_MESSAGE_NAME.insert_avp(AVPs);
    if (key_available())
        PANA_MESSAGE_NAME.insert_avp("AUTH");
```

```
void TxEAP()
```

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

```
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. Note that RTX_MAX_NUM is assumed to be set to the same default value for all messages. However, implementations may also reset RTX_MAX_NUM in this procedure and its value may vary depending on the message that was sent.

```
void RtxTimerStop()
```

A procedure to stop the retransmission timer.

```
void SessionTimerReStart(TIMEOUT)
```

A procedure to (re)start the PANA session timer. TIMEOUT specifies the expiration time associated with the session timer. Expiration of TIMEOUT will trigger a SESS_TIMEOUT event.

```
void SessionTimerStop()
```

A procedure to stop the current 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_NAME1", "AVP_NAME2",...)
```

A procedure to insert AVPs for each specified AVP name in the list of AVP names in the PANA message. When an AVP name ends with "*", zero, one, or more AVPs are inserted; otherwise, one AVP is inserted.

```
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 generate_pana_sa()
```

A procedure to check whether the EAP method being used generates keys and that a PANA SA will be established on successful authentication. For the PaC, the procedure is also used to check and match the PRF and Integrity algorithm AVPs advertised by the PAA in PAR[S] message. For the PAA, it is used to indicate whether a PRF and Integrity algorithm AVPs will be sent in the PAR[S]. This procedure will return TRUE if a PANA SA will be generated. Otherwise, it returns FALSE.

```
boolean key_available()
```

A procedure to check whether the PANA session has a PANA_AUTH_KEY. If the state machine already has a PANA_AUTH_KEY, it returns TRUE. If the state machine does not have a PANA_AUTH_KEY, it tries to retrieve a Master Session Key (MSK) from the EAP entity. If an MSK is retrieved, it computes a PANA_AUTH_KEY from the MSK and returns TRUE. Otherwise, it returns FALSE.

6.2. Common Variables

PAR.RESULT_CODE

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

NONCE_SENT

This variable is set to TRUE to indicate that a Nonce-AVP has already been sent. Otherwise, it is set to FALSE.

RTX_COUNTER

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

Rx:PANA_MESSAGE_NAME[flag]

This event variable is set to TRUE when the specified PANA message is received from its peering PANA entity. The "flag" contains a flag (e.g., Rx:PAR[C]), except for 'R' (Request) flag.

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. This event variable can only be set while in the OPEN state.

TERMINATE

This event variable is set to TRUE when initiation of PANA session termination is triggered. This event variable can only be set while in the OPEN state.

PANA_PING

This event variable is set to TRUE when initiation of liveness test based on PANA-Notification exchange is triggered. This event variable can only be set while in the OPEN state.

SESS_TIMEOUT

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

LIFETIME_SESS_TIMEOUT

Configurable value used by the PaC and PAA to close or disconnect an established session in the access phase. This variable indicates the expiration of the session and is set to the value of Session-Lifetime AVP if present in the last PANA-Auth-Request message in the case of the PaC. Otherwise, it is assumed that the value is infinite and therefore has no expiration. Expiration of LIFETIME_SESS_TIMEOUT will cause the event variable SESS_TIMEOUT to be set.

ANY

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

6.3. Configurable Values

RTX_MAX_NUM

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

6.4. 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 Other message header flags are not set. They are set explicitly by specific state machine actions.
- o AVPs that are mandatory to be included in a message are inserted with appropriate values set.

6.5. Common Retransmission Rules

The state machines defined in this document assume that the PaC and the PAA cache the last transmitted answer message. This scheme is described in Section 5.2 of [RFC5191]. When the PaC or PAA receives a retransmitted or duplicate request, it would be able to resend the corresponding answer without any aid from the EAP layer. However, to simplify the state machine description, this caching scheme is

omitted in the state machines below. In the case that there is not a corresponding answer to a retransmitted request, the request will be handled by the corresponding state machine.

6.6. Common State Transitions

The following transitions can occur at any state with exemptions explicitly noted.

```

-----
State: ANY
-----

Exit Condition          Exit Action          Exit State
-----+-----+-----
- - - - - (Re-transmissions) - - - - -
RTX_TIMEOUT &&          Retransmit();        (no change)
RTX_COUNTER <
RTX_MAX_NUM
-----
- - - - - (Reach maximum number of transmissions) - - - - -
(RTX_TIMEOUT &&          Disconnect();         CLOSED
 RTX_COUNTER >=
 RTX_MAX_NUM) ||
SESS_TIMEOUT
-----

```

```

-----
State: ANY except INITIAL
-----

Exit Condition          Exit Action          Exit State
-----+-----+-----
- - - - - (liveness test initiated by peer) - - - - -
Rx:PNR[P]              Tx:PNA[P]();        (no change)
-----

```

```

-----
State: ANY except WAIT_PNA_PING
-----

Exit Condition          Exit Action          Exit State
-----+-----+-----
- - - - - (liveness test response) - - - - -
Rx:PNA[P]              None();              (no change)
-----

```

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

State: CLOSED

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Catch all event on closed state) - - - - -		
ANY	None();	CLOSED
- - - - -		

7. PaC State Machine

7.1. 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.

7.1.1. Delivering EAP Messages from PaC to EAP Peer

TxEAP() procedure in the PaC state machine serves as the mechanism to deliver EAP 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 [RFC4137], 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.

7.1.2. Delivering EAP Messages from EAP Peer to PaC

An EAP message 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 message to its lower layer. In the case where the EAP peer follows the EAP peer state machine defined in [RFC4137], EAP_RESPONSE event variable refers to eapResp variable of the EAP peer state machine and the EAP message is contained in eapRespData variable of the EAP peer state machine.

7.1.3. EAP Restart Notification from PaC to EAP Peer

The EAP peer state machine defined in [RFC4137] has an initialization procedure before receiving an EAP message. 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.

7.1.4. 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 [RFC4137], 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 an MSK is generated by the EAP authentication method in use, eapKeyAvailable variable is set to TRUE and eapKeyData variable contains the MSK. Note that EAP_SUCCESS and EAP_FAILURE event variables may be set to TRUE even before the PaC receives a PAR with a 'Complete' flag set from the PAA.

7.1.5. 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 [RFC4137], alt_reject() procedure sets altReject variable of the EAP peer state machine. Note that the EAP peer state machine in [RFC4137] also defines altAccept variable; however, it is never used in PANA in which EAP-Success messages are reliably delivered by the last PANA-Auth exchange.

7.2. Configurable Values

FAILED_SESS_TIMEOUT

This is a configurable value that allows the PaC to determine whether a PaC authentication and authorization phase has stalled without an explicit EAP success or failure notification.

7.3. Variables

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 an EAP conversation completes with success.

EAP_FAILURE

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

EAP_RESPONSE

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

EAP_RESP_TIMEOUT

This event variable is set to TRUE when the PaC that has passed an EAP message to the EAP layer does not receive a subsequent EAP message from the EAP layer in a given period. This provides a time limit for certain EAP methods where user interaction may be required.

EAP_DISCARD

This event variable is set to TRUE when the EAP peer indicates that it has silently discarded the last received EAP-Request. This event does not accompany any EAP message. In the case where the EAP peer follows the EAP peer state machine defined in [RFC4137], this event variable refers to `eapNoResp`. Note that this specification does not support silently discarding EAP messages. They are treated as fatal errors instead. This may have an impact on denial-of-service resistance.

7.4. Procedures

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()

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

void EAP_RespTimerStop()

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

7.5. PaC State Transition Table

 State: INITIAL (Initial State)

Initialization Action:

NONCE_SENT=Unset;
 RTX_COUNTER=0;
 RtxTimerStop();

Exit Condition	Exit Action	Exit State
-----+-----+----- - - - - - (PaC-initiated Handshake) - - - - - AUTH_USER	Tx:PCI[](); RtxTimerStart(); SessionTimerReStart (FAILED_SESS_TIMEOUT);	INITIAL
-----	-----	-----

```

- - - - - (PAA-initiated Handshake, not optimized) - - - - -
Rx:PAR[S] &&                EAP_Restart();                WAIT_PAA
!PAR.exist_avp              SessionTimerReStart
("EAP-Payload")            (FAILED_SESS_TIMEOUT);
                            if (generate_pana_sa())
                                Tx:PAN[S]("PRF-Algorithm",
                                           "Integrity-Algorithm");
                            else
                                Tx:PAN[S]();
- - - - -

- - - - - (PAA-initiated Handshake, optimized) - - - - -
Rx:PAR[S] &&                EAP_Restart();                INITIAL
PAR.exist_avp              TxEAP();
("EAP-Payload") &&         SessionTimerReStart
eap_piggyback()            (FAILED_SESS_TIMEOUT);

Rx:PAR[S] &&                EAP_Restart();                WAIT_EAP_MSG
PAR.exist_avp              TxEAP();
("EAP-Payload") &&         SessionTimerReStart
!eap_piggyback()           (FAILED_SESS_TIMEOUT);
                            if (generate_pana_sa())
                                Tx:PAN[S]("PRF-Algorithm",
                                           "Integrity-Algorithm");
                            else
                                Tx:PAN[S]();

EAP_RESPONSE               if (generate_pana_sa())        WAIT_PAA
                            Tx:PAN[S]("EAP-Payload",
                                        "PRF-Algorithm",
                                        "Integrity-Algorithm");
                            else
                                Tx:PAN[S]("EAP-Payload");
- - - - -

```



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

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PAR-PAN exchange) - - - - -		
Rx:PAR[] && !eap_piggyback()	RtxTimerStop(); TxEAP(); EAP_RespTimerStart(); if (NONCE_SENT==Unset) { NONCE_SENT=Set; Tx:PAN[]("Nonce"); } else Tx:PAN[]();	WAIT_EAP_MSG
Rx:PAR[] && eap_piggyback()	RtxTimerStop(); TxEAP(); EAP_RespTimerStart();	WAIT_EAP_MSG
Rx:PAN[]	RtxTimerStop();	WAIT_PAA

- - - - - (PANA result) - - - - -		
Rx:PAR[C] && PAR.RESULT_CODE== PANA_SUCCESS	TxEAP();	WAIT_EAP_RESULT
Rx:PAR[C] && PAR.RESULT_CODE!= PANA_SUCCESS	if (PAR.exist_avp ("EAP-Payload")) TxEAP(); else alt_reject();	WAIT_EAP_RESULT_ CLOSE

 State: WAIT_EAP_MSG

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (Return PAN/PAR from EAP) - - - - -		
EAP_RESPONSE && eap_piggyback()	EAP_RespTimerStop() if (NONCE_SENT==Unset) { Tx: PAN[]("EAP-Payload", "Nonce"); NONCE_SENT=Set; } else Tx: PAN[]("EAP-Payload");	WAIT_PAA
EAP_RESPONSE && !eap_piggyback()	EAP_RespTimerStop() Tx: PAR[]("EAP-Payload"); RtxTimerStart();	WAIT_PAA
EAP_RESP_TIMEOUT && eap_piggyback()	Tx: PAN[]();	WAIT_PAA
EAP_DISCARD && eap_piggyback()	Tx: PAN[](); SessionTimerStop(); Disconnect();	CLOSED
EAP_FAILURE (EAP_DISCARD && !eap_piggyback())	SessionTimerStop(); Disconnect();	CLOSED

 State: WAIT_EAP_RESULT

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (EAP Result) - - - - -		
EAP_SUCCESS	if (PAR.exist_avp ("Key-Id")) Tx: PAN[C]("Key-Id"); else Tx: PAN[C](); Authorize(); SessionTimerReStart (LIFETIME_SESS_TIMEOUT);	OPEN

```

EAP_FAILURE          Tx: PAN[C]();
                    SessionTimerStop();
                    Disconnect();

```

```

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

```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (EAP Result) - - - - -		
EAP_SUCCESS EAP_FAILURE	if (EAP_SUCCESS && PAR.exist_avp("Key-Id")) Tx: PAN[C]("Key-Id"); else Tx: PAN[C](); SessionTimerStop(); Disconnect();	CLOSED

```

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

```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (liveness test initiated by PaC) - - - - -		
PANA_PING	Tx: PNR[P](); RtxTimerStart();	WAIT_PNA_PING
-----+-----+-----		
- - - - - (re-authentication initiated by PaC) - - - - -		
REAUTH	NONCE_SENT=Unset; Tx: PNR[A](); RtxTimerStart();	WAIT_PNA_REAUTH

```

- - - - - (re-authentication initiated by PAA)- - - - -
Rx:PAR[]      EAP_RespTimerStart();      WAIT_EAP_MSG
               TxEAP();
               if (!eap_piggyback())
                 Tx:PAN[]("Nonce");
               else
                 NONCE_SENT=Unset;
               SessionTimerRestart
                 (FAILED_SESS_TIMEOUT);

```

```

- - - - - (Session termination initiated by PAA) - - - - -
Rx:PTR[]      Tx:PTA[]();                CLOSED
               SessionTimerStop();
               Disconnect();

```

```

- - - - - (Session termination initiated by PaC) - - - - -
TERMINATE     Tx:PTR[]();                SESS_TERM
               RtxTimerStart();
               SessionTimerStop();

```

State: WAIT_PNA_REAUTH

Exit Condition	Exit Action	Exit State
- - - - - (re-authentication initiated by PaC) - - - - -		
Rx:PNA[A]	RtxTimerStop(); SessionTimerRestart (FAILED_SESS_TIMEOUT);	WAIT_PAA
- - - - - (Session termination initiated by PAA) - - - - -		
Rx:PTR[]	RtxTimerStop(); Tx:PTA[](); SessionTimerStop(); Disconnect();	CLOSED

State: WAIT_PNA_PING

Exit Condition	Exit Action	Exit State
- - - - - (liveness test initiated by PaC) - - - - -		
Rx:PNA[P]	RtxTimerStop();	OPEN

```

- - - - - (re-authentication initiated by PAA)- - - - -
Rx:PAR[]      RtxTimerStop();          WAIT_EAP_MSG
               EAP_RespTimerStart();
               TxEAP();
               if (!eap_piggyback())
                 Tx:PAN[]("Nonce");
               else
                 NONCE_SENT=Unset;
               SessionTimerReStart
                 (FAILED_SESS_TIMEOUT);

```

```

- - - - - (Session termination initiated by PAA) - - - - -
Rx:PTR[]      RtxTimerStop();          CLOSED
               Tx:PTA[]();
               SessionTimerStop();
               Disconnect();

```

```

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

```

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

8. PAA State Machine

8.1. 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.

8.1.1. EAP Restart Notification from PAA to EAP Authenticator

An EAP authenticator state machine defined in [RFC4137] 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 authenticator. The event notification is done via EAP_Restart() procedure in the initialization action of the PAA state machine.

8.1.2. 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 [RFC4137], 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.

8.1.3. 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 [RFC4137], 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.

8.1.4. 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 [RFC4137], 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 an MSK if the MSK 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 and EAP_FAILURE event variables as a trigger to send a PAR message to the PaC.

8.2. Variables

OPTIMIZED_INIT

This variable indicates whether the PAA is able to piggyback an EAP-Request in the initial PANA-Auth-Request. Otherwise, it is set to FALSE.

PAC_FOUND

This variable is set to TRUE as a result of a PAA-initiated handshake.

REAUTH_TIMEOUT

This event variable is set to TRUE to indicate that the PAA initiates a re-authentication with the PaC. The re-authentication timeout should be set to a value less than the session timeout carried in the Session-Lifetime AVP if present.

EAP_SUCCESS

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

EAP_DISCARD

This event variable is set to TRUE when the EAP authenticator indicates that it has silently discarded the last received EAP-Response message. This event does not accompany any EAP message. In the case where the EAP authenticator follows the EAP authenticator state machines defined in [RFC4137], this event variable refers to eapNoReq.

8.3. Procedures

boolean new_key_available()

This is a procedure to check whether the PANA session has a new PANA_AUTH_KEY. If the state machine already has a PANA_AUTH_KEY, it returns FALSE. If the state machine does not have a PANA_AUTH_KEY, it tries to retrieve an MSK from the EAP entity. If an MSK has been retrieved, it computes a PANA_AUTH_KEY from the MSK and returns TRUE. Otherwise, it returns FALSE.

8.4. PAA State Transition Table

 State: INITIAL (Initial State)

Initialization Action:

```

OPTIMIZED_INIT=Set|Unset;
NONCE_SENT=Unset;
RTX_COUNTER=0;
RtxTimerStop();
    
```

Exit Condition	Exit Action	Exit State
----- - - - - - (PCI and PAA initiated PANA) - - - - - (Rx:PCI[] PAC_FOUND)	<pre> if (OPTIMIZED_INIT == Set) { EAP_Restart(); SessionTimerReStart (FAILED_SESS_TIMEOUT); } else { if (generate_ana_sa()) Tx:PAR[S]("PRF-Algorithm", "Integrity-Algorithm"); else Tx:PAR[S](); </pre>	INITIAL


```

    }

EAP_REQUEST      if (generate_pana_sa())      INITIAL
                  Tx:PAR[S]("EAP-Payload",
                  "PRF-Algorithm",
                  "Integrity-Algorithm");
                  else
                  Tx:PAR[S]("EAP-Payload");
                  RtxTimerStart();
- - - - -
- - - - - (PAN Handling) - - - - -
Rx: PAN[S] &&      if (PAN.exist_avp          WAIT_EAP_MSG
((OPTIMIZED_INIT ==
  Unset) ||
PAN.exist_avp
  ("EAP-Payload"))
                  TxEAP();
                  else {
                  EAP_Restart();
                  SessionTimerReStart
                    (FAILED_SESS_TIMEOUT);
                  }

Rx: PAN[S] &&      None();                      WAIT_PAN_OR_PAR
(OPTIMIZED_INIT ==
  Set) &&
! PAN.exist_avp
  ("EAP-Payload")
- - - - -
- - - - -
State: WAIT_EAP_MSG
- - - - -

```

Exit Condition	Exit Action	Exit State
EAP_REQUEST	<pre> if (NONCE_SENT==Unset) { Tx:PAR[]("Nonce", "EAP-Payload"); NONCE_SENT=Set; } else Tx:PAR[]("EAP-Payload"); RtxTimerStart(); </pre>	WAIT_PAN_OR_PAR
EAP_FAILURE	<pre> PAR.RESULT_CODE = PANA_AUTHENTICATION_ </pre>	WAIT_FAIL_PAN

```

        REJECTED;
Tx:PAR[C] ("EAP-Payload");
RtxTimerStart();
SessionTimerStop();

```

```

EAP_SUCCESS &&
Authorize()
    PAR.RESULT_CODE =          WAIT_SUCC_PAN
    PANA_SUCCESS;
    if (new_key_available())
        Tx:PAR[C] ("EAP-Payload",
                   "Key-Id");
    else
        Tx:PAR[C] ("EAP-Payload");
    RtxTimerStart();

```

```

EAP_SUCCESS &&
!Authorize()
    PAR.RESULT_CODE =          WAIT_FAIL_PAN
    PANA_AUTHORIZATION_
    REJECTED;
    if (new_key_available())
        Tx:PAR[C] ("EAP-Payload",
                   "Key-Id");
    else
        Tx:PAR[C] ("EAP-Payload");
    RtxTimerStart();

```

```

-----
- - - - - (Receiving EAP-Timeout or invalid message) - - - - -
EAP_TIMEOUT ||      SessionTimerStop();          CLOSED
EAP_DISCARD      Disconnect();
-----

```

```

-----
State: WAIT_SUCC_PAN
-----

```

Event/Condition	Action	Exit State
Rx: PAN[C]	RtxTimerStop(); SessionTimerReStart (LIFETIME_SESS_TIMEOUT);	OPEN

```

-----
State: WAIT_FAIL_PAN
-----

```

Exit Condition	Exit Action	Exit State
	(PAN Processing)	

```

Rx: PAN[C]          RtxTimerStop();          CLOSED
                   Disconnect();
-----

```

```

State: OPEN
-----

```

Event/Condition	Action	Exit State
-----+-----+-----		
- - - - - (re-authentication initiated by PaC) - - - - -		
Rx: PNR[A]	NONCE_SENT=Unset; EAP_Restart(); Tx: PNA[A]();	WAIT_EAP_MSG
-----+-----+-----		
- - - - - (re-authentication initiated by PAA) - - - - -		
REAUTH REAUTH_TIMEOUT	NONCE_SENT=Unset; EAP_Restart();	WAIT_EAP_MSG
-----+-----+-----		
- - (liveness test based on PNR-PNA exchange initiated by PAA) - -		
PANA_PING	Tx: PNR[P](); RtxTimerStart();	WAIT_PNA_PING
-----+-----+-----		
- - - - - (Session termination initiated from PAA) - - - - -		
TERMINATE	Tx: PTR[](); SessionTimerStop(); RtxTimerStart();	SESS_TERM
-----+-----+-----		
- - - - - (Session termination initiated from PaC) - - - - -		
Rx: PTR[]	Tx: PTA[](); SessionTimerStop(); Disconnect();	CLOSED
-----+-----+-----		

```

State: WAIT_PNA_PING
-----

```

Exit Condition	Exit Action	Exit State
-----+-----+-----		
- - - - - (PNA processing) - - - - -		
Rx: PNA[P]	RtxTimerStop();	OPEN
-----+-----+-----		
- - - - - (re-authentication initiated by PaC) - - - - -		
Rx: PNR[A]	RtxTimerStop(); NONCE_SENT=Unset;	WAIT_EAP_MSG

```

        EAP_Restart();
        Tx:PNA[A]();
-----
- - - - - (Session termination initiated from PaC)- - - - -
Rx:PTR[]          RtxTimerStop();          CLOSED
                  Tx:PTA[]();
                  SessionTimerStop();
                  Disconnect();
-----

```

```

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

```

Exit Condition	Exit Action	Exit State
- - - - - (PAR Processing)- - - - -		
Rx:PAR[]	TxEAP(); RtxTimerStop(); Tx:PAN[]();	WAIT_EAP_MSG
- - - - - (Pass EAP Response to the EAP authenticator)- - - - -		
Rx:PAN[] && PAN.exist_avp ("EAP-Payload")	TxEAP(); RtxTimerStop();	WAIT_EAP_MSG
- - - - - (PAN without an EAP response) - - - - -		
Rx:PAN[] && !PAN.exist_avp ("EAP-Payload")	RtxTimerStop();	WAIT_PAN_OR_PAR
- - - - - (EAP retransmission) - - - - -		
EAP_REQUEST	RtxTimerStop(); Tx:PAR[]("EAP-Payload"); RtxTimerStart();	WAIT_PAN_OR_PAR
- - - - - (EAP authentication timeout or failure)- - - - -		
EAP_FAILURE EAP_TIMEOUT EAP_DISCARD	RtxTimerStop(); SessionTimerStop(); Disconnect();	CLOSED

```

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

```

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

```

- - - - - (PTA processing) - - - - -
Rx:PTA[]      RtxTimerStop();      CLOSED
              Disconnect();
- - - - -

```

9. Implementation Considerations

9.1. PAA and PaC Interface to Service Management Entity

In general, it is assumed that each device or network equipment has a PANA protocol stack available for use by other modules within the device or network equipment. One such module is the Service Management Entity (SME). The SME is a generic term for modules that manage different services (including network protocols) that are installed on a device or equipment. To integrate the PANA protocol with the SME, 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. This interface should include common procedures such as startup, shutdown, and re-authenticate signals. It should also provide for extracting keying material. For the PAA, the SME-PANA interface should also provide a method for communicating filtering parameters to the Enforcement Point(s) when cryptographic filtering is used. The filtering parameters include keying material used for bootstrapping secured transport such as IPsec. When a PAA device interacts with the backend authentication server using a AAA protocol, its SME may also provide an interface to the AAA protocol to obtain authorization parameters such as the authorization lifetime and additional filtering parameters.

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. Acknowledgments

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

12. References

12.1. Normative References

[RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

12.2. Informative References

[RFC4137] Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba,
"State Machines for Extensible Authentication Protocol
(EAP) Peer and Authenticator", RFC 4137, August 2005.

Authors' Addresses

Victor Fajardo (editor)
Telcordia Technologies
1 Telcordia Drive
Piscataway, NJ 08854
USA

Phone: +1 732 699 5368
EMail: vfajardo@research.telcordia.com

Yoshihiro Ohba
Toshiba Corporate Research and Development Center
1 Komukai-Toshiba-cho, Saiwai-ku
Kawasaki, Kanagawa 212-8582
Japan

Phone: +81 44 549 2230
EMail: yoshihiro.ohba@toshiba.co.jp

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Facultad de Informatica
Murcia 30100
Spain

Phone: +34 868 888 501
EMail: rafa@um.es

