IEEE P802.11  
Wireless LANs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The Public Key Authentication Exchange | | | | |
| Date: 2015-04-04 | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | Aruba Networks | 1322 Crossman avenue, Sunnyvale, California, United States of America | +1 408 227 4500 | dharkins at arubanetworks dot com |
|  |  |  |  |  |

Abstract

This submission describes a public key-based authentication protocol that runs pre-association and can be used to establish a shared PMK. Authentication can be server-side only or mutual.

***Instruct the editor to create new subsection at the end of 8.4.1, replacing A with the next number in the sequence of subsection numbers:***

**8.4.1.A Hashed Identity field**

The format of the Hashed Identity field is shown in Table <ANA-1> (Hashed Identity field).

|  |  |  |
| --- | --- | --- |
| Length of hashes | Recipient Hashed Identity | Sender Hashed Identity |

Octets: 1 variable variable

The Length of hashes field indicates the length, in octets, of the entire field.

Each Hashed Identity field is the digest output from a cryptographic hash function on a public key that identifies the recipient and sender, respectively, of a frame carrying the Hashed Identity field.

***Instruct the editor to modify section 8.6.16.1 as indicated:***

**8.6.16.1 Self-protected Action fields**

**Table 8-349—Self-protected Action field values**

|  |  |
| --- | --- |
| Self-protected Action field value | Description |
| <ANA-2> | Public Key Authentication Request |
| <ANA-3> | Public Key Authentication Response |
| <ANA-4> | Public Key Authentication Confirm |
| <ANA-4+1>-255 | Reserved |

***Instruct the editor to create a new subsection in 8.6.16 and replace A in the following, and all sub-subsections, with the next number in the sequence of subsection numbers:***

**8.6.16.A Public Key Authentication Request format**

**8.6.16.A.1 Public Key Authentication Request frame self protection**

Protection of this frame is provided by authenticating this frame, and encrypting a portion of it, by use of a secret known only to the transmitter and receiver of the frame.

**8.6.16.A.2 Public Key Authentication Request frame details**

The PKAUTH Request frame is used to initiate a protocol that authenticates two STAs using public key cryptography. The format of the PKAUTH Request frame is shown in Table <ANA-5> (PKAUTH Request frame Action field format).

|  |  |  |
| --- | --- | --- |
| Order | Information | Notes |
| 1 | Category |  |
| 2 | Self-protected Action | Value of <ANA-2> from table 8-349 |
| 3 | Finite Cyclic group |  |
| 4 | Hashed Identity | The hashed identites |
| 5 | Element | A field element from a finite field encoded as described in 11.6.12.2 (PKEX Messages). |
| 6 | FILS Wrapped Data |  |

**8.6.16.B Public Key Authentication Response**

**8.6.16.B.1 Public Key Authentication Response frame self protection**

Protection of this frame is provided by authenticating this frame, and encrypting a portion of it, by use of a secret known only to the transmitter and receiver of the frame.

**8.6.16.B.2 Public Key Authentication Response frame details**

The PKAUTH Response frame is used to engage in a protocol that authenticates two STAs using public key cryptography. The format of the PKAUTH Response frame is shown in Table <ANA-6> (PKAUTH Response frame Action field format).

|  |  |  |
| --- | --- | --- |
| Order | Information | Notes |
| 1 | Category |  |
| 2 | Self-protected Action | Value of <ANA-3> from table 8-349 |
| 3 | Finite Cyclic group |  |
| 4 | Hashed Identity | Hashed identities |
| 5 | FILS Wrapped Data | Wrapped handshaking data |
| 6 | FILS Wrapped Data | Wrapped authentication data |

**8.6.16.C Public Key Authentication Confirm frame**

**8.6.16.C.1 Public Key Authentication Confirm frame self protection**

Protection of this frame is provided by authenticating this frame, and encrypting a portion of it, by use of a secret known only to the transmitter and receiver of the frame.

**8.6.16.C.2 Public Key Authentication Confirm frame details**

The PKAUTH Confirm frame is used to engage in a protocol that authenticates two STAs using public key cryptography. The format of the PKAUTH Response frame is shown in Table <ANA-7> (PKAUTH Confirm frame Action field format).

|  |  |  |
| --- | --- | --- |
| Order | Information | Notes |
| 1 | Category |  |
| 2 | Self-protected Action | Value of <ANA-4> from table 8-349 |
| 3 | Finite Cyclic group |  |
| 4 | Hashed Identity | Hashed Identities |
| 5 | FILS Wrapped Data | Wrapped authentication data |

**11.6.A Public Key Authentication protocol**

The Public Key Authentication (PKAUTH) protocol allows two STAs to authenticate each other using trusted public keys. The protocol is a 3 message handshake. It runs pre-association using self-protected Action frames and consists of a Public Key Authentication request, sent by a STA deemed the initiator to another STA deemed the responder, a Pubilc Key Authentication response, sent by the responder to the initiator, and a Public Key Authentication confirm, sent by the initiator to the responder.

The protocol always provides authentication of the responder to the initiator and, optionally, authentication of the initiator back to the responder—i.e. mutual authentication. Authentication is performed by trusted public keys, called Identity Keys. It is assumed that the STA initiating the PKA has obtained, and trusts, the public key of the peer. For optional mutual authentication it is assumed the responding STA has, and trusts, the public key of the initiating STA. How these public keys are obtained is outside the scope of this specification but may be accomplished by a protocol such as the Public Key Exchange (PKEX).

The PKAUTH achieves perfect forward secrecy (PFS) through an exchange of ephemeral public keys. The ephemeral public keys must be in the same group as the peers’ Identity Keys.

To facilitate authentication, PKAUTH uses cryptographic primitives to provide hashing, using the SHA2 family of hash functions, and to provide an authenticated encryption component, using AES-SIV. The particular hash algorithm and the length of the key used by AES-SIV depends on the length, in bits, of the prime defining the group used by the STAs’ public keys:

SHA256, AES-SIV-128: len(p) <= 256

SHA384, AES-SIV-192: 256 < len(p) <= 384

SHA512, AES-SIV-256: 384 < len(p)

Where len(p) is the length, in bits, of the prime, p.

**11.6.A.1 Public Key Authentication request**

A STA initiates the PKAUTH by sending a Public Key Authentication request to the STA it wishes to authenticate. The Public Key Authentication request contains identifiers indicating the Identity Key of the target to whom the Public Key Authentication request is being sent and the Identity Key of the sender of the Public Key Authentication request. It also contains an ephemeral public key and a proof-of-possession of the private key of the ephemeral public key.

To initiate PKAUTH, a STA shall:

1. Select the finite cyclic group from which its Identity Key has been generated and generates an ephemeral key pair in the same group.
2. Perform the Diffie-Hellman key exchange using its ephemeral private key and the public Identity Key of the peer it wishes to authenticate in order to generate a shared secret, *W*, and use the KDF from section 11.6.1.7.2 (Key derivation function) to generate a shared key, *k*:

*W* = scalar-op(*initiator-ephem-priv*, *Responder-Pub-ID*)

*k* = KDF-z(F(*W*), “PKAUTH First Intermediate Key”, *group*)

Where scalar-op() is defined in section 11.3.4 (Finite Cyclic groups), *initiator-ephem-priv* is the private portion of the ephemeral keypair, *Responder-Pub-ID* is the public Identity Key of the peer, z is the length of the digest of the hash function per 11.6.A, F() is the mapping function from 11.3.4 (Finite Cyclic groups), “PKAUTH First Intermediate Key” is an unterminated ASCII string, and *group* is the numeric identifier of the group in which the public keys have been created.

1. Generate a random nonce, *ni*, whose length is equivalent to the length of the digest of the hash function used per 11.6.A above.
2. Wrap *ni* using AES-SIV with key k with the body.

The Public Key Authentication request frame (see section 8.6.16.A.2) is constructed by assigning the finite cyclic group of the public keys to the finite cyclic group field, the peer’s public Identity Key and STA’s public Identity key are hashed using the hash algorithm per 11.6.A above and copied to the Recipient and Sender Hashed Identity portions, respectively, of the Hashed Identities field with a length of the Hashed Identities field set to twice the length of the hash’s digest, and the STA’s ephemeral public key is copied into the Element field. The nonce ni is wrapped with AES-SIV using key *k* with the body of the frame from Finite Cyclic Group field to the Hashed Identities field, inclusive, and the STA’s MAC address as separate components of associated data. The wrapped nonce is then copied into the FILS wrapped data field of the frame.

The Public Key Authentication request shall be transmitted to the responder. If the initiator knows the responder’s public Identity Key but not the responder’s MAC address, the frame shall be broadcast. The initiator may retransmit the Public Key Authentication request if a response is not received and it may abandon PKAUTH after a number of retransmissions. The frequency of retransmission and the number of retransmissions before abandonment are outside the scope of this standard.

Upon receipt of a Public Key Authenntication request, a STA first checks whether its hashed identity is in the Recipient Hashed Identity portion of the Hashed Identity field. If not, the STA shall silenetly drop the frame. If it is, the STA checks whether the indicated finite cyclic group is the group used with its Identity Key. If it differs, the STA shall silently drop the frame. Otherwise, the STA assumes the role of responder and processes the frame.

To process a Public Key Authentication request, a STA shall:

1. Check whether it:
2. Has a public Identity Key of the initiator that hashes to the Sender Hashed Identity; and,
3. Wishes to do mutual authentication

If both these conditions are true the responder authenticates the initiator and performs mutual authentication below, if either of are false it does not.

1. Extract the initiator’s ephemeral public key from the frame and validate it. If it is not valid, the frame shall be silently dropped and PKAUTH shall be unsuccessfully terminated. Otherwise, the STA performs a Diffie-Hellman key exchange to generate the shared secret W, followed by the KDF to generate k:

*W* = scalar-op(*responder-id-priv*, *Initiator-Ephem-Pub*)

*k* = KDF-z(F(*W*), “PKAUTH First Intermediate Key”, *group*)

Where *responder-id-priv* is the private portion of the responder’s Identity Key, *Initiator-Ephem-Pub* is the initiator’s ephemeral public key, and the rest of the terms are as above.

1. Unwrap the initiator’s nonce, *ni*, using AES-SIV with k as the key using the body of the frame from the Finite Cyclic Group field to the Hashed Identities field, inclusive, and the initiator’s MAC address as separate components of associated data. If AES-SIV returns failure, the frame shall be dropped and PKAUTH shall terminate unsuccessfully. Otherwise, processing completes successfully.

If Processing completes successfully, the responder STA shall generate a Public Key Authentication response.

**11.6.A.2 Public Key Authentication response**

After processing a Public Key Authentication request, a responder STA shall:

1. Generate an ephemeral key pair in the group indicated in the Finite Cyclic group field of the Public Key Authentication request and compute the following:

*X* = scalar-op(*respond-ephem-priv*, *Initiator-Ephem-Pub*)

*S* = elem-op(*W*, *X*)

Where *respond-ephem-priv* is the private portion of the responder’s ephemeral key and elem-op() is defined in section 11.3.4 (Finite Cyclic groups). If mutual authentication is being performed, in addition do the following:

*Y* = scalar-op(*respond-ephem-priv*, *Initiator-Identity-Pub*)

*Z* = scalar-op(*respond-id-priv*, *Initiator-Identity-Pub*)

*S* = elem-op(*S*, *Y*)

*S* = elem-op(*S*, *Z*)

Where *Initiator-Identity-Pub* is the initiator’s public Identity Key.

1. Generate a random nonce, *nr*, whose length is equivalent to the length of the digest of the hash function used per 11.6.A above, and an authentication token, *rauth*, as follows:

*rauth* = Hash(*n*i || *nr* || F(*Initiator-Ephem-Pub*) || F(*Responder-Ephem-Pub*) ||

[ F(*Initiator-Id-Pub*) || ] F(*Responder-Id-Pub*) || 0)

Where || indicates concatentation, *ni* and *nr* are the initiator and responder nonces, respectively, 0 is a single octet with a zero value, the information in brackets [] is present when the responder is performing mutual authentication and absent when it is not, Hash() is the hash algorithm per 11.6.A above.

1. Generate another shared key, *r*:

*r* = KDF-z(Hash(*ni* || *nr*), “PKAUTH Shared Key”, F(*S*))

Where “PKAUTH Shared Key” is an unterminated ASCII string and the rest of the terms are as above.

1. Wrap a concatentation of the initiator’s nonce, the responder’s nonce, and the responder’s ephemeral public key with AES-SIV using *k* as the key using the body of the frame from the Finite Cyclic Group field to the Hashed Identities field, inclusive, and the responder’s MAC address as separate components of associated data.

The responder indicates it is doing mutual authentication by including a hash of the initiator’s Identity Key in the Recipient Hashed Identity portion of the Hashed Identity field of the Public Key Authentication response frame (see section 8.6.16.B.2). When it is not doing mutual authentication that portion of the field is set to all zeros. The rest of the Public Key Authentication response frame is constructed by assigning the finite cyclic group of the public keys to the finite cyclic group field, the responder’s public Identity Key is hashed using the hash algorithm per 11.6.A above and copied to the Sender Hashed Identity portion of the Hashed Identities field, the wrapped nonces and ephemeral key are copied to the first FILS Wrapped Data field. The authentication token, *rauth*, is wrapped with AES-SIV using *r* as the key and body of the frame from the Finite Cyclic Group to the Hashed Identities field, inclusive, and the responder’s MAC address as separate components of associated data. The wrapped *rauth* token is copied to the second FILS Wrapped Data field.

The Public Key Authentication response frame shall then be transmitted to the initiator.

Upon receipt of a Public Key Authentication response frame, a STA first checks whether it has any state with which to process it—i.e. has the STA previously sent a Public Key Authentication request frame to this peer identified by the Sender Hashed Identity portion of the Hashed Identities field. If not, it shall silently drop the frame. If so, the initiator processes the Public Key Authentication response frame as follows:

1. If a hash of its Identity Key is in the Recipient Hashed Identity portion of the Hashed Identities field it performs mutual authentication, if the field is all zeros it does not perform mutual authentication. In all other cases, the frame shall be silently dropped.
2. Use AES-SIV with *k* to unwrap the two nonces and the responder’s ephemeral public key. If unwrapping returns a failure the frame shall be silently dropped. Otherwise, it checks whether its nonce is contained in the unwrapped data. If not, the frame shall be silently dropped.
3. The responder’s ephemeral key shall validated, if it is not valid the frame shall be silently dropped. Otherwise, the initiator shall generate the following:

*X* = scalar-op(*initiator-ephem-priv*, *Responder-Ephem-Pub*)

*S* = elem-op(*W*, *X*)

If mutual authentication is being performed, in addition do the following:

*Y* = scalar-op(*initiator-identity-priv*, *Responder-Ephem-Pub*)

*Z* = scalar-op(*initiator-identity-priv*, *Resonder-Identity-Pub*)

*S* = elem-op(*S*, *Y*)

*S* = elem-op(*S*, *Z*)

1. Generate another shared key, *r*:

*r* = KDF-z(Hash(*ni* || *nr*), “PKAUTH Shared Key”, F(*S*))

1. Unwrap *rauth* using AES-SIV with *r* as the key with the body of the frame from the Finite Cyclic Group to the Hashed Identities field, and the responder’s MAC address as separate components of associated data. If unwrapping returns a failure, the frame shall be dropped and PKAUTH shall terminate unsuccessfully.
2. Generate a verifier for *rauth* and ensure they match. If they do not match, the frame shall be dropped and PKAUTH shall terminate unsuccessfully. Otherwise, the peer has been authenticated.

If the peer has been successfully authenticated, the initiator proceeds to Public Key Authentication Confirmation.

**11.6.A.3 Public Key Authentication Confirmation**

First, the initiator generates an authenticating token, iauth, as follows:

*iauth* = Hash(*nr* || *ni* || F(*Responder-Ephem-Pub*) || F(*Initiator-Ephem-Pub*) ||

F(*Responder-Id-Pub*) [ || F(*Initiator-Id-Pub*) ] || 1)

Where the information in brackets [] is present when the responder is performing mutual authentication and absent when it is not, and 1 is a single octet with the value one. It then wraps *iauth* with AES-SIV using *r* as the key.

It then constructs a Public Key Authentication Cofirmation frame (see section 8.6.16.C.2): it assigns the finite cyclic group of the public keys to the finite cyclic group field, puts a hash of the responder’s Identity Key in the Recipient Hashed Identity portion of the Hashed Identity field, if mutual authentication is performed it puts a hash of its own Identity Key in the Sender Hashed Identity portion, otherwise it fills that portion with all zeros. Finally, the authentication token, iauth, is wrapped with AES-SIV using r as the key with the body of the frame from the Finite Cyclic Group field to the Hashed Identiies field, inclusive, and the initiator’s MAC address as separate compoents of associated data. The initiator then copies the wrapped *iauth* into the FILS Wrapped Data field.

The Public Key Authentication Confirmation frame is transmitted to the responder.

**References:**