Internet Draft
Expiration: September 2003
File: <draft-jwalker-eap-archie-00.txt>
Expires: August 28, 2003

Jesse Walker
Intel Corporation
Russ Housley
Vigil Security
February 28, 2003

The EAP Archie Protocol

Status of this Memo

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

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

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

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

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

Copyright Notice

Abstract

This memo defines an EAP authentication protocol based on pre-shared keys, called Archie. This protocol performs mutual authentication and session key derivation based on static pre-shared key. Archie is designed as a native EAP method.

Table of Contents

1. Introduction

The Extensible Authentication Protocol (EAP), described in [4], provides a standard mechanism for support of additional authentication methods within PPP. EAP is also used within IEEE 802 networks through the IEEE 802.1X [8] framework.

EAP supports many authentication schemes, including smart cards, Kerberos, Public Key, One Time Passwords, TLS, and others. This memo defines a new authentication scheme, called the EAP-Archie. EAP-Archie performs mutual authentication and fresh session key derivation, based on a static pre-shared key.

1.1. Specification of requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [7].

1.2 Terminology

This document frequently uses the following terms:

Backend Authentication Server
: A backend authentication server is an entity that provides an authentication service to an authenticator. When used, this server typically executes EAP Methods for the Authenticator. [This terminology is also used in [IEEE.802-1X.2001].]

EAP Authenticator
: The end of the EAP link initiating the EAP authentication methods. [Note: This terminology is also used in [IEEE.802-1X.2001], and has the same meaning in this document].

EAP Peer
: The end of the EAP Link that responds to the authenticator. [Note: In [IEEE.802-1X.2001], this end is known as the Supplicant.]

EAP Server
: The entity that terminates the EAP authentication with the peer. In the case where there is no Backend Authentication Server, this term refers to the EAP Authenticator. Where the EAP Authenticator operates in pass-through, it refers to the Backend Authentication Server.

2. Protocol Overview

2.1. Overview of EAP-Archie

EAP-Archie is a native EAP authentication method. That is, EAP-Archie is an authentication protocol, and a stand-alone version of EAP-Archie outside of EAP is not defined.

EAP-Archie assumes a long-lived 512-bit secret shared between the EAP Peer and the EAP Server. This long-lived secret is called the pre-shared secret. The pre-shared secret has an internal structure. It consists of 2 128-bit keys and a 256-bit key, respectively called the key-confirmation key (KCK), the key-encryption key (KEK), and the key-derivation key (KDK). The protocol uses the key-confirmation key to mutually authenticate the EAP Peer and the EAP Server. The protocol uses the key-encryption key to distribute secret nonces used for session key derivation. The protocol uses the key-derivation key to derive a session key between the EAP Peer and the EAP Server. EAP-Archie assumes that the pre-shared secret is known only to the EAP Peer and EAP Server, and authentication may be compromised if the pre-shared secret has wider distribution.

EAP-Archie uses four messages consisting of two EAP request/response exchanges. As with all EAP methods, the EAP Server initiates the exchange. Figure 1 depicts the EAP-Archie message flow:

```
   EAP Peer                          EAP Server
   ========                          ==========
                                <--- EAP-Request/EAP-Type=Archie
```

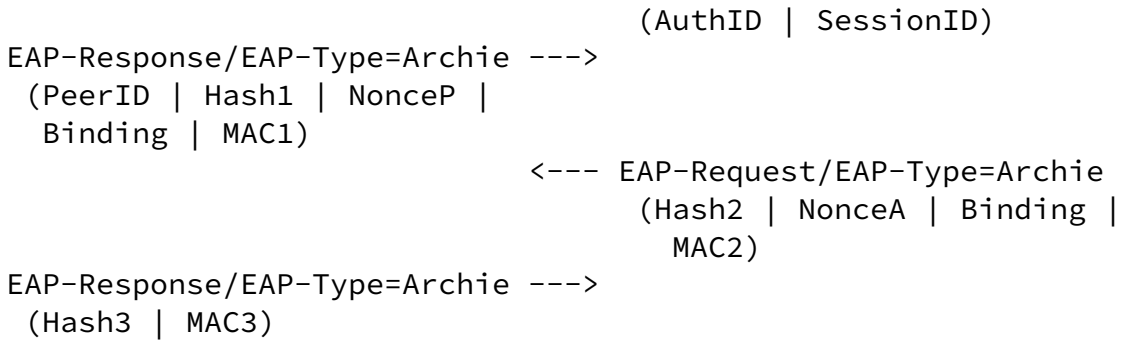


Figure 1. The EAP-Archie Message Flow

The first message is called an Archie Start message. The Archie Start message initiates the EAP-Archie exchange. The EAP Server sends the Archie Start message to the EAP Peer in an EAP-Request message of type Archie. The Archie Start message requests that the EAP Peer authenticate itself using EAP-Archie. The Archie Start message is unauthenticated but does convey the identity of the EAP Server, denoted by AuthID, and a challenge value, denoted SessionID. The

AuthID is an NAI [9] identifying the EAP Server, and SessionID is a 256-bit random value. The AuthID allows the EAP Peer to identify the pre-shared secret for this context. SessionID serves several purposes. First, since it is unpredictable, its use allows the EAP Server to detect replay attacks later in the protocol. Second, it provides a session identifier for the entire exchange.

The response to the Archie Start is called the Archie Response message. The EAP Peer sends the Archie Response message to the EAP Server in an EAP-Response message of type Archie. The Archie Response message authenticates the EAP Peer to the EAP Server, and requests the EAP Server to authenticate itself to the EAP Peer. This message conveys five values, called PeerID, Hash1, NonceP, Binding, and MAC1. PeerID is an NAI identifying the EAP Peer to the EAP Server. Hash1 is a 128-bit value giving the 16 most significant octets of the SHA1 digest value [5] of the Archie Start message. NonceP is a 256-bit random value selected by the EAP Peer, encrypted under the KEK using the AES Key Wrap [6]; the EAP Peer and EAP Server use the decrypted nonce value to derive a session key, as explained below. MAC1 is a message authentication code computed over the prior Archie Response message fields, using the KCK as the key and AES-CBC-MAC-96, defined in Section 3.1 below, as the MAC algorithm. PeerID allows the EAP

Server to identify the pre-shared secret to use in this protocol instance. Hash1 ties this Archie Response to a particular Archie Request and consequently proves that the Archie Response is live. If the unencrypted nonce value appears random, so will the encrypted value NonceP, and thus NonceP, being unpredictable, can serve as a challenge. Binding is the initial binding of the derived keys. It indicates the identifier the EAP Peer uses and its intended communication partner, which is typically some sort of Network Access Server. Finally, a correct MAC1 value authenticates the Response message and hence the EAP Peer to the EAP Server.

The response to the Archie Response message is called the Archie Confirm message. The EAP Server sends the Archie Confirm message to the EAP Peer in an EAP-Request message of type Archie. The Archie Confirm message authenticates the EAP Server to the EAP Peer. The Archie Confirm message conveys four values, Hash2, NonceA, Binding, and MAC2. Hash2 is the 128-bit value giving the 16 most significant octets of the SHA1 digest value of the Archie Response message. NonceA is a 256-bit random value selected by the EAP Server and encrypted under the KEK using the AES Key Wrap; the EAP Peer and EAP Server use the decrypted nonce value to derive a session key, as explained below. Binding is a copy of field of the same name from the Archie Response message, and confirms the initial target binding. MAC2 is a message authentication code compute over the prior Archie Confirm fields, using the KCK as the key and AES-CBC-MAC-96 as the MAC algorithm. Hash2 ties the Confirm message to a particular EAP-

Archie Request/Response exchange, so proves that a Confirm with a valid MAC2 value is live and part of this protocol instance. MAC2, if valid, authenticates the Confirm message and hence the EAP Server to the EAP Peer.

The final message responds to the Archie Confirm, and is called the Archie Finish message. The EAP Peer sends the Archie Finish message to the EAP Server in an EAP-Response message of type Archie. This message consists of Hash3 and MAC3. Hash3 is a 128-bit value, being the 16 most significant octets of the SHA1 digest value of the Archie Confirm message. MAC3 is the AES-CBC-MAC-96 digest of Hash3 under the KCK. This message serves no cryptographic purpose, but is defined since every EAP Request requires an EAP Response. The Archie Finish message includes MAC3 to prevent undetected forgery attacks. It includes Hash3 to detect replays from other sessions.

The use of SessionID in EAP-Archie and the values Hash1, Hash2, and Hash3 allow for multiple simultaneous sessions between the EAP Peer and EAP Server.

2.2. Retry Behavior

As with other EAP protocols, the EAP Server is responsible for retry behavior. This means that if the EAP Server does not receive a reply from the peer, it MUST resend the EAP-Request for which it has not yet received an EAP-Response. However, the EAP Peer MUST NOT resend EAP-Response messages without first being prompted by the EAP Server.

For example, if the initial Archie Start message sent by the EAP Server were to be lost, then the EAP Peer would not receive this message, so would not respond to it. As a result, the EAP Server would resend the Archie Start message. Once the EAP Peer received the Archie Start message, it would send an EAP-Response conveying the Archie Response message. If the EAP-Response were to be lost, then the EAP Server would resend the initial Archie Start, triggering the EAP Peer to resend the EAP-Response.

As a result, it is possible that an EAP Peer will receive duplicate EAP-Request messages, and may send duplicate EAP-Responses. Both the EAP Peer and the EAP Server should be engineered to handle this possibility.

2.3. Fragmentation

EAP-Archie does not require fragmentation.

2.4. Identity Verification

EAP-Archie always performs mutual authentication. EAP-Archie uses a 512-bit long-lived pre-shared secret, identified to the EAP Peer by the EAP Server's NAI, and to the EAP Server by the EAP Peer's NAI. EAP-Archie uses the 128 most significant bits of the pre-shared secret serves as an authentication key.

The EAP Server considers the EAP Peer successfully authenticated if Hash1 and MAC1 from the Archie Response message are both valid. An EAP Server implementation of EAP-Archie MUST silently discard any EAP-Response messages of type Archie it receives with invalid Hash1 or MAC1 fields.

Similarly, the EAP Peer considers the EAP Server authenticated if Hash2 or MAC2 from the Archie Confirm message are valid. An EAP Peer implementation of EAP-Archie MUST silently discard any EAP-Request messages of type Archie it receives with invalid Hash2 or MAC2 fields.

The EAP Server MAY require that the Binding field from the Archie Response message be consistent with the verified identity. For instance, the Binding might identify the MAC address of the EAP Peer. If it knows the proper value for this, the EAP Server MAY declare the message a forgery if this MAC address does not match the EAP Peer's reported NAI.

2.5. Key Derivation

Each instance of EAP-Archie constitutes a session, and derives a session key (SK) from the KDK portion of the long-lived pre-shared secret. The derived key consists of 256 bits. This section describes how the SK is derived. It also describes how subordinate keys, if required, are derived.

EAP-Archie relies on a 512-bit pre-shared secret. EAP-Archie uses the second 128 most significant bits of the pre-shared secret as a KEK, and it uses the 512 least significant bits as a KDK.

When using EAP-Archie, the EAP Peer selects a random value called PeerNonce, wraps it using the AES Key Wrap algorithm under the KEK to produce a field called NonceP, and sends NonceP to the EAP Server in the Archie Response message. PeerNonce is 256-bits. The EAP Server uses KEK and the AES Key Wrap algorithm to recover PeerNonce.

Similarly, the EAP Server selects a random value called AuthNonce, wraps it using the AES Key Wrap algorithm under the KEK to produce a

field called NonceA, and sends NonceA to the EAP Peer in the Archie

Confirm message. AuthNonce is 256 bits. The EAP Peer uses the KEK and the AES Key Wrap algorithm to recover AuthNonce.

Once both the EAP Server and Peer have both AuthNonce and PeerNonce, they derive the SK using the KDK, AuthNonce, PeerNonce, and the text string "Archie session key", using a function called the Archie-PRF:

```
SK = Archie-PRF(KDK, "Archie session key" | AuthNonce | PeerNonce)
```

Here, the "|" symbol denotes string concatenation. Section 3.2 defines the Archie-PRF.

The derived SK is uniquely identified by the combination of AuthID, PeerID, and SessionID, where AuthID is the identity the EAP Server presents in the Archie Request message, PeerID is the identity the EAP Peer presents in the Archie Response message, and SessionID is the random challenge value the EAP Server presents in the Archie Request message.

The derived SK can be used for many functions. One particular use is to derive fresh pairwise keys subordinate to the SK, for use between the EAP Peer and a Server device, such as a Network Access Server (NAS). This is done using the SK with the Archie-PRF. The fresh pairwise key between an EAP Peer with address AddrP and a Server with address AddrS is defined by:

```
key = Archie-PRF(SK, "Archie pairwise key" | AddrS | AddrP)
```

The fresh pairwise key is the 256 most significant bits of the output (octets 1 through 32). The addresses are use-specific. For example, MAC addresses or IP addresses could be used. The values for AddrS and AddrP for the fresh pairwise key stemming from the execution of an EAP-Archie protocol instance are taken from the Archie Response Bindings field. If they are needed, how the EAP Server learns AddrS and AddrP to derive additional fresh pairwise key later is dependent on a keying material update mechanism, and it is outside the scope of this document.

3. Cryptographic Algorithms

3.1. AES-CBC-MAC-128 and AES-CBC-MAC-96

This section reviews the definition of the AES-CBC-MAC.

Let K denote an AES key, and let AES-Encrypt denote the AES encrypt primitive. The AES-CBC-MAC-128 of a string S is defined using the following steps:

a. Let L denote the length of S in octets. Let L' = L + p, where p is the unique integer 0 <= p < 16 needed so that L' is a multiple of 16.
b. Let n = L'/16, and partition S into substrings, such that S = S[1] S[2] ... S[n-1] S'[n], with S[1], S[2], ..., S[n-1] consisting of 16 octets each and S'[n] consisting of 16 octets if p is 0 and 16-p octets otherwise. Let S[n] = S'[n]0^p, where 0^p denotes p octets of zero pad.
c. Set IV to 16 zero octets.
d. For i = 1 to n do
   IV = AES-Encrypt(K, S[i] XOR IV)
e. Return IV.

The EAP-Archie protocol uses a variant of AES-CBC-MAC-128 called AES-CBC-MAC-96. This variant differs from the AES-CBC-MAC-128 described above in only step e, which it replaces by:

e'. Return the most significant 12 octets of IV.

3.2 The Archie-PRF

This section defines the Archie-PRF function. The Archie-PRF always generates 512 bits of output.

Let K denote a key. Archie-PRF of a string S is defined by the following steps:

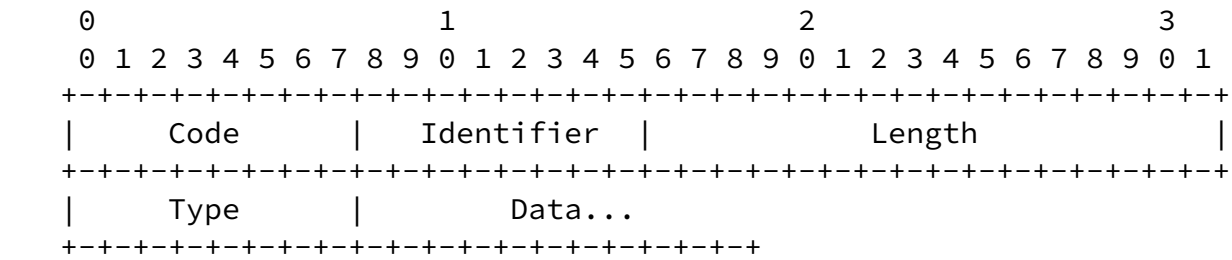
a. Output is initialized to the null string.
b. For i = 1 to 4 do
   Output = Output | AES-CBC-MAC-128(K, S | i | 64)
c. Return Output

In step b, the loop index i is encoded as a single octet, and the numeric value 64 is encoded as the octet 0x40.

4. Detailed Description of EAP-Archie

4.1. Archie Message Format Summary

A summary of the Archie Request/Response packet format is shown below. The fields are transmitted from left to right.



Code

1 - Request
2 - Response

Identifier

The identifier field is one octet and aids in matching responses with requests.

Length

The Length field is two octets and indicates the length of the EAP message including the Code, Identifier, Length, Type, and Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and should be ignored on reception.

Type

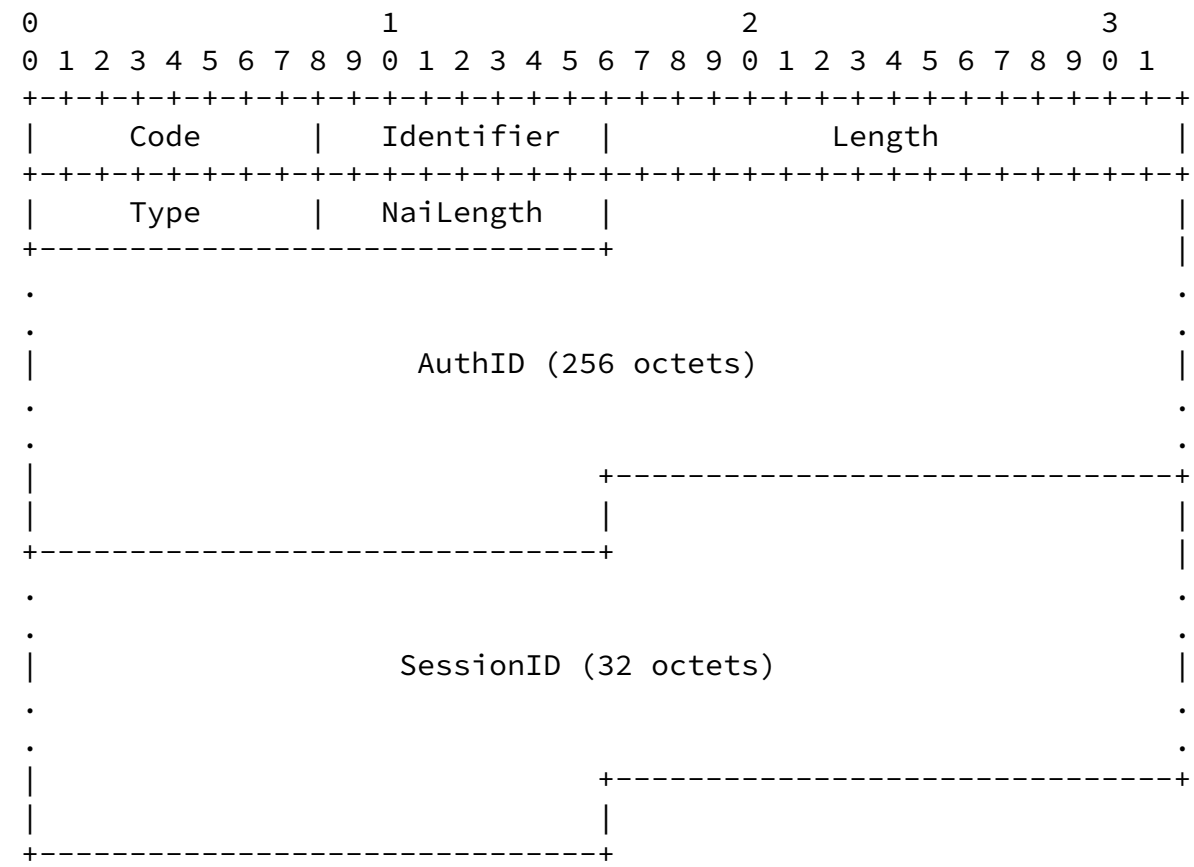
TBS - Archie

Data

The format of the Data field is determined by the Code field.

4.2. Archie Request Message

A summary of the Archie Request message format is shown below. The fields are transmitted from left to right.



Code

1

Identifier

The Identifier field is one octet and aids in matching

Responses with Requests. The Identifier field MUST be changed on each Request message.

Length

The Length field is two octets and indicates the length of the EAP message including the Code, Identifier, Length, Type, NaiLength, AuthID, and SessionID fields. Its value is 294.

Type

TBS - Archie

NaiLength

The number of octets used in the AuthID field. The value 0

means the AuthID is the full 256 octets in length.

AuthID

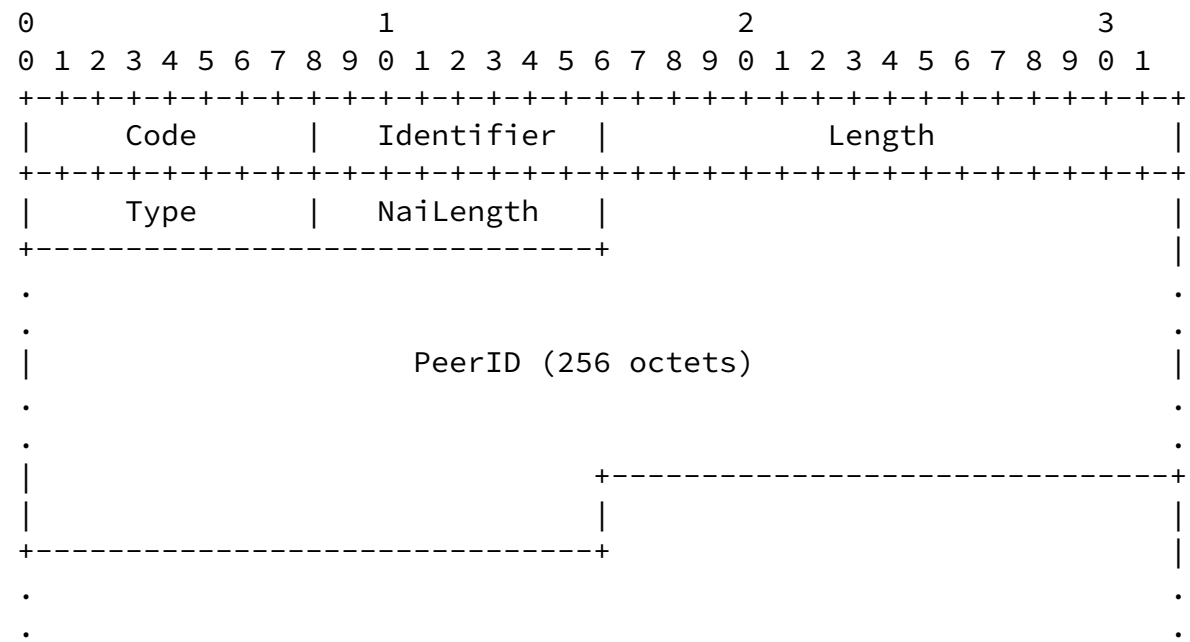
The AuthID field gives the NAI the EAP Server uses to identify itself to the EAP Peer. The first NaiLength octets of this field are non-zero, while any remaining octets of the AuthID field MUST be zero.

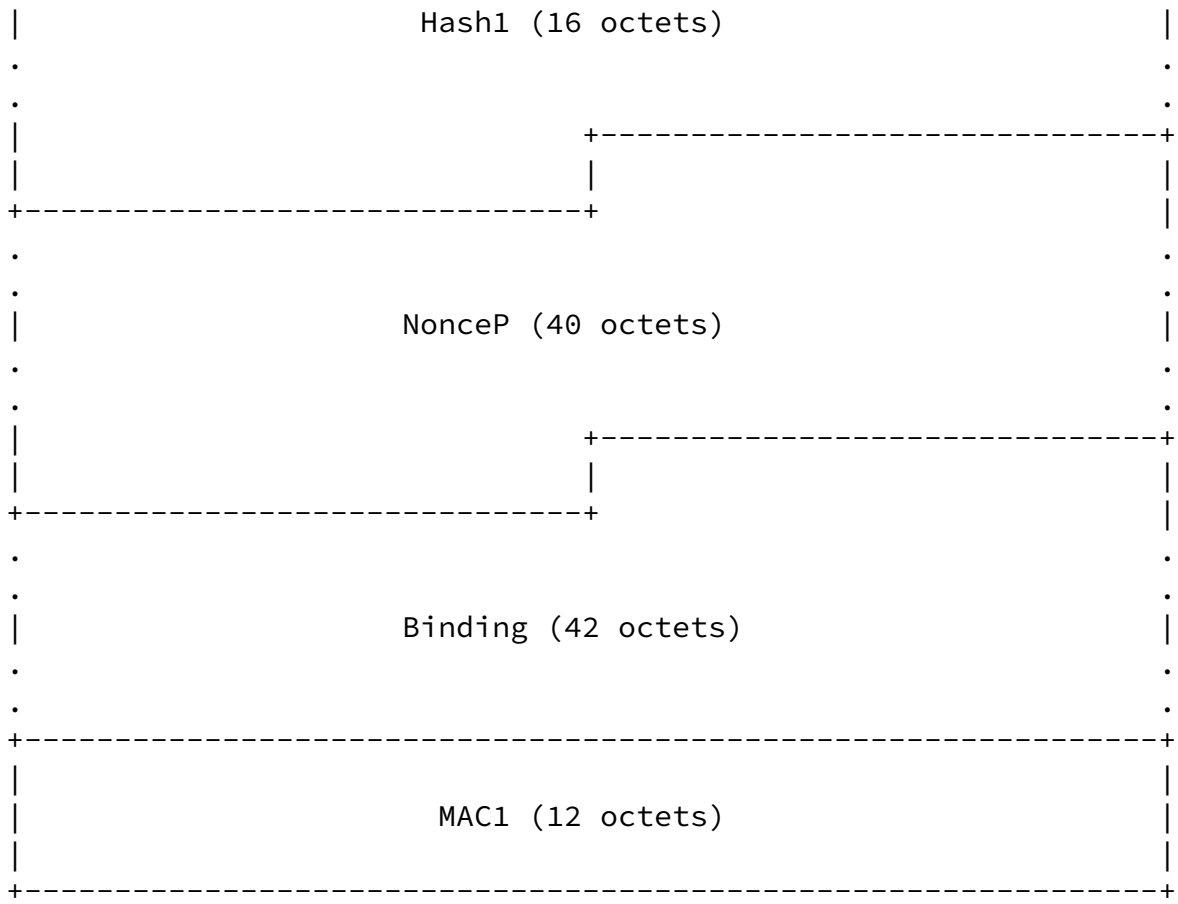
SessionID

The SessionID field is 32 octets. The EAP Server MUST generate this randomly. This value represents the session id for this instance of the EAP-Archie protocol.

4.3. Archie Response Message

A summary of the Archie Response message format is shown below. The fields are transmitted from left to right.


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Code
Identifier
Length
Type
NaiLength
PeerID (256 octets)



Code

2

Identifier

The Identifier field is one octet and MUST match the Identifier field from the corresponding request.

Length

The Length field is two octets and indicates the length of the EAP message including the Code, Identifier, Length, Type, NaiLength, PeerID, Hash1, NonceP, Binding, and MAC1 fields. Its value is 372.

Type

TBS - Archie

NaiLength

The number of octets used in the PeerID field. The value 0 means the PeerID is the full 256 octets in length.

PeerID

The PeerID field gives the NAI the EAP Peer uses to identify itself to the EAP Server. The first NaiLength octets of this field are non-zero, while any remaining octets of the PeerID field MUST be zero.

Hash1

The Hash1 field is 16 octets. The value of this field MUST be the first 16 octets of the SHA1 digest of the Archie Request to which this Archie Response message responds.

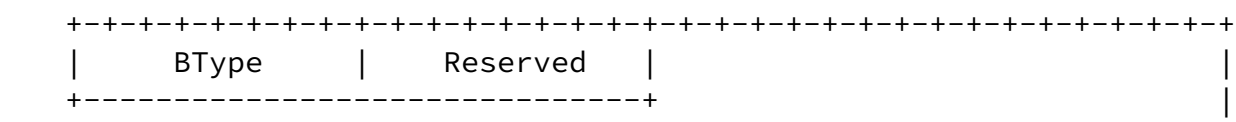
NonceP

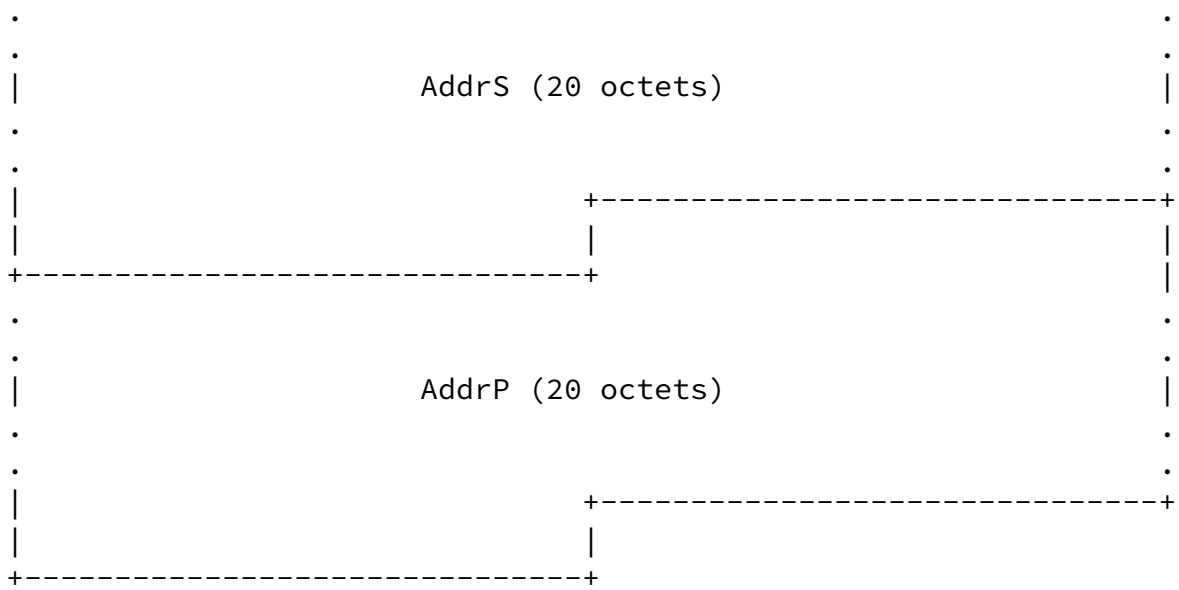
The NonceP field is 40 octets. The EAP Peer MUST generate its value as follows:

a. Generate a 256 bit (32 octet) random value.
b. Use the AES Key Wrap algorithm and the KEK portion of the long-lived pre-shared secret it shares with the EAP Server to encrypt the random value.

Binding

The Binding field is 42 octets. It has its own internal structure:





BType

Btype identifies the type of address in the binding structure. The specified values include

0 - Not present (no binding information)
1 - 802 MAC addresses
2 - IPv4 addresses
3 - IPv6 addresses
4 - IPv4 transport address
5 - IPv6 transport address
6-255 - reserved

Reserved

0

AddrS

Addr identifies address of the party with whom the EAP Peer wants to communicate, typically a NAS device. This field assumes a different format depending of the value of the BType field. The specified formats include:

BType	AddrS Value
0	AddrS is zero
1	The 802 MAC address in octets 1 through 6 of AddrS, and octets 7 through 20 are zero.
2	The IPv4 IP address in octets 1 through 4 of AddrS, and octets 5 through 20 are zero.
3	The IPv6 IP address in octets 1 through 16 of AddrS, and octets 17 through 20 are zero.
4	The the IPv4 IP address in octets 1 through 4 of AddrS, the transport protocol number in octet 5, and the port number, if required, in octets 7 and 8; octets 6 and 9 through 20 are zero. If a port number is not required, the octets 7 and 8 are encoded as zero.
5	The IPv6 IP address in octets 1 through 16 of AddrS, the transport protocol number in octet 17, and the port number, if required, in octets 19 and 20. Octets 18 is zero. If a port number is not required, the octets 19 and 20 are encoded as zero.

AddrP

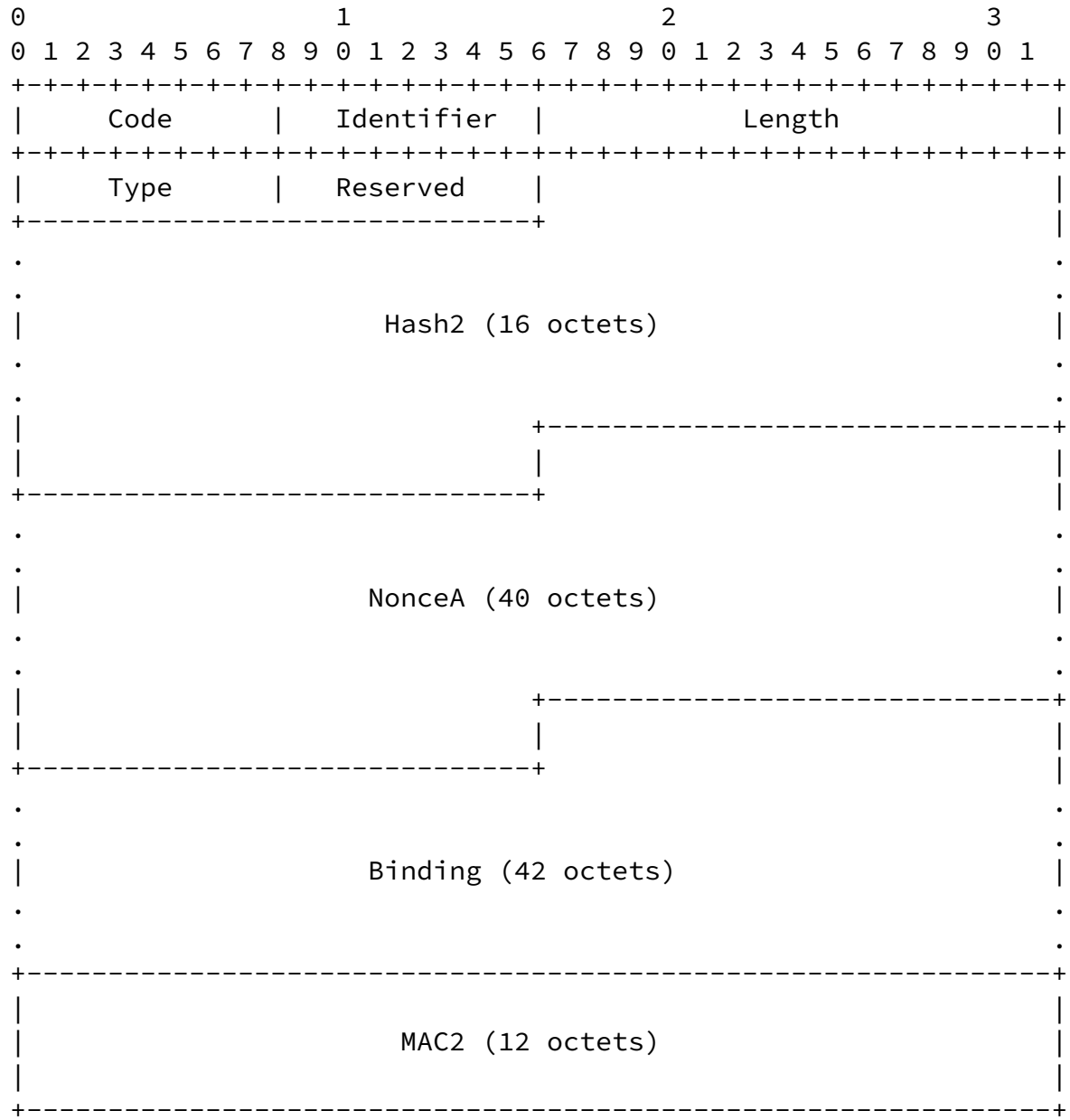
Addr identifies address the EAP Peer wants to use with the targeted system, typically a NAS device. This field assumes a different format depending of the value of the BType field. The formatting rules for the AddrS field apply to this field verbatim.

MAC1

The MAC1 field is 12 octets. The value of this field MUST be the AES-CBC-MAC-96 of the Code, Identifier, Length, Type, NaiLength, PeerID, Hash1, NonceP, and Binding fields, using the KCK portion of the long-lived pre-shared secret as the authentication key.

4.4. Archie Confirm Message

A summary of the Archie Confirm message format is shown below. The fields are transmitted from left to right.



Code

1

Identifier

The Identifier field is one octet and aids in matching Responses with Requests. The Identifier field MUST be changed on each Request message.

Length

The Length field is two octets and indicates the length of the

EAP message including the Code, Identifier, Length, Type, Reserved, Hash2, NonceA, Binding, and MAC2 fields. Its value is 116.

Type

TBS - Archie

Reserved

0

Hash2

The Hash2 field is 16 octets. The value of this field MUST be the first 16 octets of the SHA1 digest of the Archie Response to which this Archie Confirm message responds.

NonceA

The NonceA field is 40 octets. The EAP Server MUST generate its value as follows:

a. Generate a 256 bit (32 octet) random value.
b. Use the AES Key Wrap algorithm and the KEK portion of the long-lived pre-shared secret it shares with the EAP Server to encrypt the random value.

Binding

The Binding field is 42 octets. It is encoded as in the Archie Response message. The EAP Server SHOULD copy the value of this field from the Response field of the same name. The EAP Server

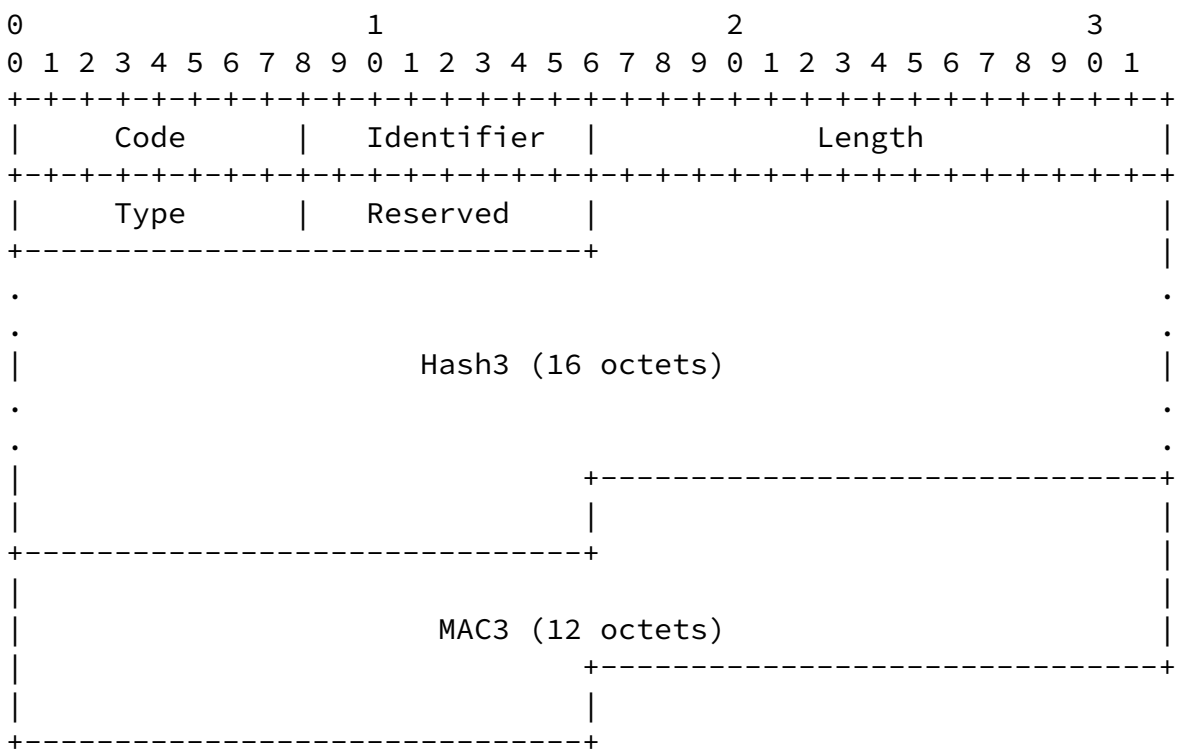
MAY alter the AddrS or AddrP values to indicate something wrong with the Response message Binding values (e.g., the AddrS MAC address does not belong to the NAS that forwarded the Response message), but the Confirm Binding BType MUST be identical to that of the Response Binding BType.

MAC2

The MAC2 field is 12 octets. The value of this field MUST be the AES-CBC-MAC-96 of the Code, Identifier, Length, Type, Reserved, Hash2, NonceA, and Binding fields, using the KCK portion of the long-lived pre-shared secret as the authentication key.

4.5. Archie Finish Message

A summary of the Archie Finish message format is shown below. The fields are transmitted from left to right.



Code

2

Identifier

The Identifier field is one octet and MUST match the Identifier field from the corresponding request.

Length

The Length field is two octets and indicates the length of the EAP message including the Code, Identifier, Length, Type, Reserved, Hash3, MAC3 fields. Its value is 34.

Type

TBS - Archie

Reserved

0

Hash3

The Hash3 field is 16 octets. The value of this field MUST be the first 16 octets of the SHA1 digest of the Archie Confirm to which this Archie Finish message responds.

MAC3

The MAC3 field is 12 octets. The value of this field MUST be the AES-CBC-MAC-96 of the Code, Identifier, Length, Type, Reserved, and Hash3 fields, using the KCK portion of the long-lived pre-shared secret as the authentication key.

5. IANA Considerations

This document introduces two new IANA considerations.

First, it requires IANA to allocate a new EAP Type for EAP-Archie.

Second, it requires IANA to manage the space of bindings supported by the protocol.

6. Security Considerations

6.1. Intended use

EAP-Archie is designed to be useful over an insecure network. The protocol implements mechanisms to protect an EAP-Archie exchange from both active and passive attack.

6.2. Mechanism

EAP-Archie relies on a 512-bit pre-shared secret. The protocol does not restrict how the secret is constructed nor address how it comes into being. However, the protocol does not increase the entropy of the shared secret in any way; if the shared secret does not provide 256 bits of entropy, then the SK and any derived keys can have less entropy as well.

6.3. Security Claims

EAP-Archie provides:

a. mutual authentication,
b. integrity protection,
c. replay protection,

d. key derivation,
e. key strength,
f. dictionary attack resistance, and
g. man-in-the-middle resistance.

The Archie Response and Archie Confirm messages accomplish mutual authentication. The Archie Response conveys Hash1, a digest of the Archie Request message, and the Archie Request message contains an unpredictable challenge value, SessionID. The Archie Response MIC1 value is computed using the KCK portion of the pre-shared secret. It is infeasible for any computationally bounded adversary to construct the appropriate MIC1 value without the KCK, so by assumption, the

Archie Response message can only be constructed by the EAP Peer. It is similarly infeasible for any computationally bounded adversary to construct the correct MIC2 value in the Archie Confirm message without the KCK, so this must be from the EAP Server. Note that the challenge value in the Response message is implicit through the use of the Archie Response NonceP and Archie Confirm Hash2 fields.

Integrity protection is accomplished through the use of the AES-CBC-MAC-96 and the KCK.

Replay protection is accomplished through the SessionID and NonceP fields, and through Hash1, Hash2, and Hash3. If the EAP Server chooses SessionID randomly, then it is unpredictable and an adversary therefore has one chance in 2^256 in guessing the value and having the appropriate EAP Peer precomputing a Response message with the correct Hash1 value for it. The EAP Server can distinguish replayed Response and Finish messages by erroneous Hash1 and Hash3 values. Similarly, if the EAP Peer selects the value encrypted into NonceP randomly, an adversary has at most one chance in 2^256 in selecting the right NonceP value, so can detect replayed Archie Confirm messages by invalid Hash2 values.

EAP-Archie derives both a fresh session key SK, and further subordinate keys as needed. EAP-Archie does not require that any of these keys are used, however.

The key strength of the derived keys is at most the strength of the KDK portion of the pre-shared secret.

EAP-Archie's resistance to on- and off-line dictionary attack depends on the key strength of the KCK and KEK portions of the pre-share secret.

EAP-Archie defends against man-in-the-middle attack if the KCK portion of the pre-shared key is strong enough to resist attack. This can be verified using the argument showing how EAP-Archie

accomplishes mutual authentication.

6.4. Key Strength

The effective key strength of the derived keys is at most that of the

KDK. The key strength cannot exceed 256 bits, because the KDK is only 256 bits in length.

6.5. Key Hierarchy

EAP-Archie uses a three-level key hierarchy.

Level 1 is the pre-shared secret. This is a 512-bit key partitioned into three pieces: the 128-bit key confirmation key (KCK) used for mutual authentication, the 128-bit key encryption key (KEK) used for nonce hiding, and the 256-bit key derivation key (KDK), which is used to derive the next level of the hierarchy.

The second level of the hierarchy is the session key SK. The session key is derived from the KCK and the nonces using the Archie-PRF. Because it includes nonces in its derivation, the master key is fresh key if either nonce is random. The session key is known only to the EAP Peer and EAP Server.

The final level of the hierarchy are the derived keys, which are bound to particular EAP Peer/Server pairs. Derived Keys are derived from the session key and addressing information for the communicating parties. Since the session key is fresh, each derived key is fresh. Only the first 256 bits of a derived key is shared with the communicating parties. EAP-Archie does not derive additional keys for authentication and IVs.

6.6. Vulnerabilities

The Archie Request message is vulnerable to spoofing. The Archie Request message could be acknowledged, but this would not eliminate the problem, as the first message of any session establishment protocol such as EAP-Archie is subject to replay.

EAP-Archie cannot provide mutual authentication if the 128 most significant bits of the pre-shared key are known to any party other than the EAP Server and the EAP Peer.

EAP-Archie cannot guarantee that the derived session key is secure unless the last 256 bits of the pre-shared key are known only to the EAP Server and the EAP Peer.

If the pre-shared secret is derived from a low-entropy quantity such

as a password, then EAP-Archie is subject to both on-line dictionary attacks, and the EAP-Archie session key is subject to off-line dictionary attack.

To defend against on-line dictionary attack, implementations SHOULD keep track of the number of EAP-Archie messages received with invalid message authentication codes throughout the pre-shared secret lifetime, and SHOULD force the pre-shared secret to be changed if the number exceeds some threshold.

The protocol is not secure unless SessionID is random in the sense of being unpredictable to any computationally bound adversary. If SessionID is not unpredictable, then an attacker can masquerade as the EAP Server to the EAP Peer for the purpose of generating the Archie Response. The attacker can then replay the Archie Response to the EAP Server at a later time, when the Server actually would issue SessionID.

7. Acknowledgements

EAP-Archie evolved from an earlier unpublished Archie protocol defined by Bob Moskowitz, Doug Whiting, Greg Chesson, Jesse Walker, Russ Housley, and Thomas Hardjono.

8. References

[1] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[2] Sklower, K., Lloyd, B., McGregor, G., Carr, D. and T. Coradetti, "The PPP Multilink Protocol (MP)", RFC 1990, August 1996.

[3] Simpson, W., Editor, "PPP LCP Extensions", RFC 1570, January 1994.

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

[5] National Bureau of Standards, "Secure Hash Standard", FIPS PUB 180-1 (April 1995).

[6] Housley, R., "Advance Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[7] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

[8] IEEE STD 802.1X, Standards for Local and Metropolitan Area Networks: Port Based Access Control, June 14, 2001

[9] Aboba, B., and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

9. Author Addresses

Jesse R. Walker
Intel Corporation
2111 N.E. 25th Avenue
Hillsboro, OR 97214
USA
jesse.walker@intel.com

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
housley@vigilsec.com

10. Full Copyright Statement

will not be revoked by the Internet Society or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS 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."

11. Expiration Date

This memo is filed as <draft-jwalker-eap-archie-00.txt>, and expires September 2003.