

PPPEXT Working Group

Vivek

Kamath

INTERNET-DRAFT

Ashwin

Palekar

Category: Informational

Mark

Wodrich

[<draft-kamath-pppext-peapv0-00.txt>](#)

Microsoft

**25 October 2002**

## **Microsoft's PEAP version 0 (Implementation in Windows XP SP1)**

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC 2026](#).

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

Copyright (C) The Internet Society (2002). All Rights Reserved.

### Abstract

This specification documents the implementation of PEAP supported in Windows XP SP1. This implementation utilizes a version number of zero (0) and supports acknowledged and protected success and failure indications, using the EAP Extensions method, Type 33.



Table of Contents

<a href="#">1.</a>	Introduction .....	<a href="#">3</a>
<a href="#">1.1</a>	EAP encapsulation .....	<a href="#">3</a>
<a href="#">1.2</a>	Version field .....	<a href="#">3</a>
<a href="#">1.3</a>	EAP Extensions method .....	<a href="#">4</a>
<a href="#">2.</a>	Details of EAP extensions method .....	<a href="#">4</a>
<a href="#">2.1</a>	Extensions Request Packet .....	<a href="#">4</a>
<a href="#">2.2</a>	Extensions Response Packet .....	<a href="#">5</a>
<a href="#">2.3</a>	AVP format .....	<a href="#">6</a>
<a href="#">3.</a>	Security considerations .....	<a href="#">7</a>
<a href="#">3.1</a>	Authentication and integrity protection .....	<a href="#">7</a>
<a href="#">3.2</a>	Outcomes .....	<a href="#">7</a>
<a href="#">4.</a>	Normative references .....	<a href="#">8</a>
<a href="#">5.</a>	Informative references .....	<a href="#">9</a>
<a href="#">Appendix A</a>	- Examples .....	<a href="#">10</a>
	Acknowledgments .....	<a href="#">18</a>
	Author's Addresses .....	<a href="#">18</a>
	Intellectual Property Statement .....	<a href="#">18</a>
	Full Copyright Statement .....	<a href="#">19</a>



## **1. Introduction**

Microsoft's Windows XP SP1 implementation of PEAP version 0 differs in the following ways from the protocol specified in [[PEAP](#)].

- Version field
- EAP encapsulation
- Acknowledged and protected success and failure using EAP extensions method

PEAP protocol [[PEAP](#)] supports versioning and hence servers and clients can support multiple versions of the protocol. [[PEAP](#)] is currently at version 1. Each of these differences is explained in the sections that follow.

### **1.1. EAP encapsulation**

The [[PEAP](#)] specification requires that EAP packets be tunneled within a TLS channel in their entirety. However, the Windows XP SP1 implementation of PEAP does not include an EAP header on packets sent within the TLS channel, except for EAP Extension packets (Type 33), where the complete header is sent. As a result, for EAP Types other than 33, the Code, Identifier, and Length fields are not sent, but rather EAP packets sent within the PEAP tunnel begin with the Type field.

While the Code, Identifier and Length fields are not sent over the wire, they are reconstructed at the receiver prior to EAP processing. For example, the Code and Identifier fields of the tunneled EAP packet are assumed to be the same as the equivalent fields within the outer EAP header, and the Length field of the tunneled EAP packet is derived from the Length field of the PEAP packet. This has the following implications:

- [a] The Code field of the tunneled EAP packet is assumed to be the same as the Code field of the PEAP packet. This may not always be the case; for example, an EAP Success or Failure packet (Code 3 and 4) may be tunneled within a PEAP Request packet (Code 1). This means that the Windows XP SP1 implementation of PEAP is incapable of tunneling arbitrary EAP packets.
- [b] Since the full EAP header is sent for the EAP Extensions type (Type 33), but not for other Types, it is difficult for the implementation to distinguish an Extensions Request (Code 1) from an EAP Type 1 (Identity) Request packet. In practice, this implies that the Windows XP SP1 PEAP implementation can only support authentication using a single EAP method per session.



**1.2. Version Field**

[PEAP] is currently a work-in-progress. In order to allow for backward compatibility once the final specification of PEAP is completed, a version field of zero (0) is used, rather than the value of one (1) required for conformant implementations as specified in [PEAP].

Note that use of distinct version numbers enables backward compatibility once the final specification of PEAP is complete. Version negotiation takes place at the start of the conversation, with the authenticator indicating its highest supported version. The peer then responds with the highest version it supports. The conversation will then occur using the highest version supported by both parties. This behavior is illustrated in the last example in [Appendix A](#).

**1.3. EAP extensions method**

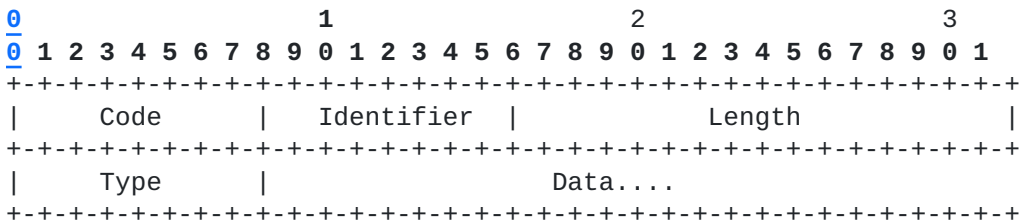
The [PEAP] termination mechanism (sending an encrypted EAP Success or EAP Failure packet) does not function correctly with Authenticators implementing [IEEE8021X]. Since IEEE 802.1X authenticators "manufacture" a clear-text EAP Success based on receipt of a RADIUS Access-Accept, or a clear-text EAP Failure based on receipt of a RADIUS Access-Reject, unless an acknowledged success/failure indication is used, the PEAP Supplicant may never receive a protected success/failure indication. This leaves the PEAP Supplicant open to attack. As a result, the Windows XP SP1 PEAP implementation supports acknowledged and protected success/failure indications, using the EAP Extensions method (Type 33).

**2. Details of EAP extensions method**

The packet formats for the EAP Extensions method (type 33) follow.

**2.1. Extensions Request Packet**

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



Code





Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field MUST be changed on each Request packet.

Length

The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, and Data fields.

Type

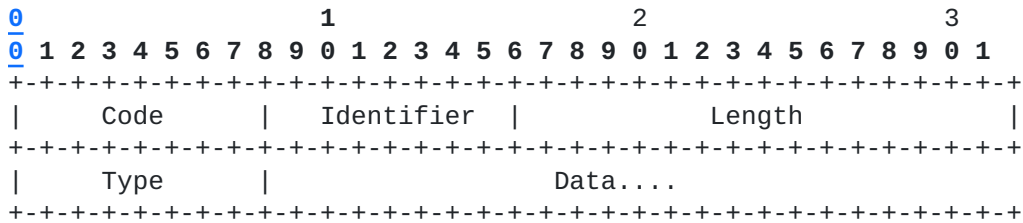
33 - EAP Extensions

Data

The Data field is of variable length, and contains Attribute-Value Pairs (AVPs).

**2.2. Extensions Response Packet**

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



Code

2

Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field MUST be changed on each Request packet.

Length

The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, and Data fields.

Type



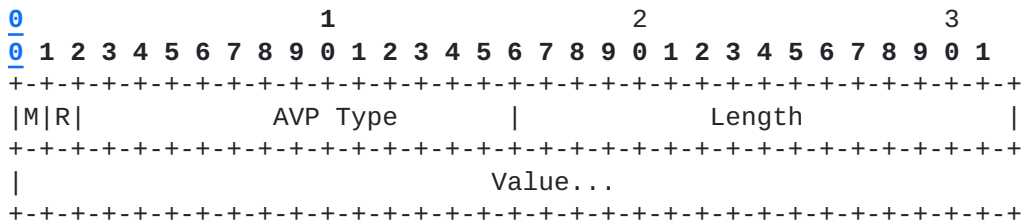
33 - EAP Extensions

Data

The Data field is of variable length, and contains Attribute-Value Pairs (AVPs).

**2.3. AVP format**

Extensions AVPs are defined as follows:



M

- 0 - Non-mandatory AVP
- 1 - Mandatory AVP

R

Reserved, set to zero (0)

AVP Type

A 14-bit field, denoting the attribute type. Allocated AVP Types include:

- 0 - Reserved
- 1 - Reserved
- 2 - Reserved
- 3 - Acknowledged Result

Length

The length of the value field in octets.

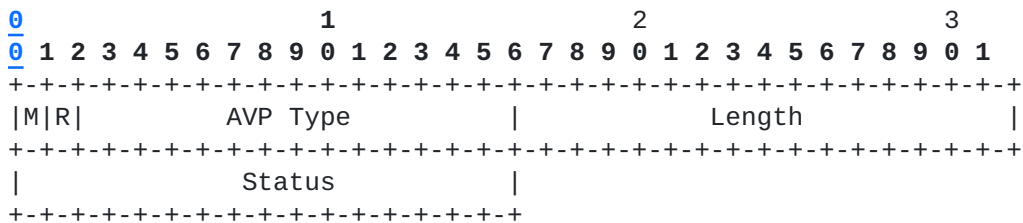
Value

The value of the attribute.



**2.3.1. Result AVP**

The Result AVP provides support for acknowledged Success and Failure within EAP. It is defined as follows:



M

1 - Mandatory AVP

R

Reserved, set to zero (0)

AVP Type

3 - Result

Length

2

Status

The status field is two octets. Values include:

- 1 - Success
- 2 - Failure

**3. Security considerations**

**3.1. Authentication and integrity protection**

The EAP Extension method is presumed to run before or after an EAP method that supports mutual authentication and establishes a protected channel. PEAP is such a method, and as a result the acknowledged Success and Failure messages are always protected.

Note however, that [[IEEE8021X](#)] manufactures clear-text EAP Success and EAP Failure messages, so that even though the Result AVP will be protected, this will be followed by a clear-text EAP Success or EAP



Failure packet.

### **3.2. Outcomes**

Within the Microsoft PEAP Version 0 implementation, support for the EAP Extensions method and the Result AVP is required. The only outcome which should be considered a successful authentication is when an EAP Request of Type=Extensions with Result AVP of Status=Success is answered by an EAP Response of Type=Extensions with Result AVP of Status=Success. All other combinations (Extensions Success, Extensions Failure), (Extensions Failure, Extensions Success), (Extensions Failure, Extensions Failure), (No extensions exchange) should be considered failed authentications, both by the EAP Peer and EAP Server. This is true regardless of whether an EAP Success or EAP Failure packet is subsequently sent, either in clear-text or within the PEAP tunnel. Because the EAP Extensions method is protected within the PEAP channel, its messages cannot be spoofed, whereas clear-text Success and Failure messages can be sent by an attacker.

While the [[PEAP](#)] specification permits a tunneled EAP Success or Failure packet to be sent as the last message, this is not possible within the Windows XP SP1 implementation, which can only tunnel EAP packets of codes Request or Response within PEAP. Since the [[IEEE8021X](#)] specification requires that the switch or access point "manufacture" a clear-text EAP Success packet when an Access-Accept is received from the backend authentication server, and a clear-text EAP Failure packet when an Access-Reject is received. As a result, a tunneled EAP Success or Failure packet, if sent as the last message, would be thrown away by conformant [[IEEE 8021X](#)] implementations, and replaced with clear-text. This problem is being addressed within the IEEE 802.1aa revision to IEEE 802.1X, but the fix may take a while to move through the standards process and be implemented in commercial products.

### **4. Normative references**

- [PEAP] Andersson, H., et al. "Protected EAP Protocol", Internet draft (work in progress), [draft-josefsson-pppext-eap-tls-eap-02.txt](#), February 2002.
- [RFC1661] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 51, [RFC 1661](#), July 1994.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.

[RFC2284] Blunk, L., Vollbrecht, J., "PPP Extensible Authentication Protocol (EAP)", [RFC 2284](#), March 1998.

Kamath, Palekar & Wodrich      Informational  
8]

[Page



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

## **5. Informative references**

[IEEE80211] Information technology - Telecommunications and  
information exchange between systems - Local and  
metropolitan area networks - Specific Requirements Part  
11: Wireless LAN Medium Access Control (MAC) and  
Physical Layer (PHY) Specifications, IEEE Std.  
802.11-1999, 1999.



Appendix A - Examples

In the case where an identity exchange occurs within PEAP Part 1, the conversation will appear as follows:

```
Authenticating Peer      Authenticator
-----
EAP-Response/
Identity (MyID) ->      <- EAP-Request/
                          Identity

EAP-Response/
EAP-Type=PEAP, V=0      <- EAP-Request/
(TLS client_hello)->    EAP-Type=PEAP, V=0
                          (PEAP Start, S bit set)

EAP-Response/
EAP-Type=PEAP, V=0      <- EAP-Request/
(TLS client_hello)->    EAP-Type=PEAP, V=0
                          (TLS server_hello,
                          TLS certificate,
                          [TLS server_key_exchange,]
                          [TLS certificate_request,]
                          TLS server_hello_done)

EAP-Response/
EAP-Type=PEAP, V=0      <- EAP-Request/
([TLS certificate,]      EAP-Type=PEAP, V=0
 TLS client_key_exchange,
 [TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->      (TLS change_cipher_spec,
                          TLS finished)

EAP-Response/
EAP-Type=PEAP ->

TLS channel established
(messages sent within the TLS channel)

EAP-Response/
Identity (MyID) ->      <- EAP-Request/
                          Identity

EAP-Response/
Identity (MyID) ->      <- EAP-Request/
                          EAP-Type=X
```



EAP-Response/  
EAP-Type=X or NAK ->

<- EAP-Request/  
EAP-Type=X

EAP-Response/  
EAP-Type=X ->

<- EAP-Request/  
EAP-Type=Extensions  
Result=Success

EAP-Response/  
EAP-Type=Extensions  
Result=Success ->

TLS channel torn down  
(messages sent in clear-text)

<- EAP-Success

In the case where the PEAP fragmentation is required, the conversation will appear as follows:

Authenticating Peer  
-----

Authenticator  
-----  
<- EAP-Request/  
Identity

EAP-Response/  
Identity (MyID) ->

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(PEAP Start, S bit set)

EAP-Response/  
EAP-Type=PEAP, V=0  
(TLS client\_hello)->

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(TLS server\_hello,  
TLS certificate,  
[TLS server\_key\_exchange,]  
[TLS certificate\_request,]  
TLS server\_hello\_done)  
(Fragment 1: L, M bits set)

EAP-Response/  
EAP-Type=PEAP, V=0 ->

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(Fragment 2: M bit set)



EAP-Response/  
EAP-Type=PEAP, V=0 ->

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(Fragment 3)

EAP-Response/  
EAP-Type=PEAP, V=0  
([TLS certificate,]  
TLS client\_key\_exchange,  
[TLS certificate\_verify,]  
TLS change\_cipher\_spec,  
TLS finished)  
(Fragment 1: L, M bits set)->

<- EAP-Request/  
EAP-Type=PEAP, V=0

EAP-Response/  
EAP-Type=PEAP, V=0  
(Fragment 2)->

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(TLS change\_cipher\_spec,  
TLS finished)

EAP-Response/  
EAP-Type=PEAP, V=0 ->

TLS channel established  
(messages sent within the TLS channel)

<- EAP-Request/  
Identity

EAP-Response/  
Identity (MyID) ->

<- EAP-Request/  
EAP-Type=X

EAP-Response/  
EAP-Type=X or NAK ->

<- EAP-Request/  
EAP-Type=X

EAP-Response/  
EAP-Type=X ->

<- EAP-Request/  
EAP-Type=Extensions  
Result=Success

EAP-Response/  
EAP-Type=Extensions





Result=Success ->

TLS channel torn down  
(messages sent in clear-text)

<- EAP-Success

In the case where the server authenticates to the client successfully in PEAP Part 1, but the client fails to authenticate to the server in PEAP Part 2, the conversation will appear as follows:

```
Authenticating Peer      Authenticator
-----
EAP-Response/
Identity (MyID) ->

EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->

EAP-Response/
EAP-Type=PEAP, V=0
([TLS certificate,
 TLS client_key_exchange,
 [TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->

EAP-Response/
EAP-Type=PEAP, V=0 ->

TLS channel established
(messages sent within the TLS channel)
```

<- EAP-Request/  
Identity

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(PEAP Start, S bit set)

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(TLS server\_hello,  
TLS certificate,  
[TLS server\_key\_exchange,]  
[TLS certificate\_request,]  
TLS server\_hello\_done)

<- EAP-Request/  
EAP-Type=PEAP, V=0  
(TLS change\_cipher\_spec,  
TLS finished)



```

EAP-Response/
Identity (MyID) ->
<- EAP-Request/
Identity

EAP-Response/
EAP-Type=X or NAK ->
<- EAP-Request/
EAP-Type=X

EAP-Response/
EAP-Type=X ->
<- EAP-Request/
EAP-Type=X

EAP-Response/
EAP-Type=Extensions
Result=Failure ->
<- EAP-Request/
EAP-Type=Extensions
Result=Failure

(TLS session cache entry flushed)
TLS channel torn down
(messages sent in clear-text)
```

<- EAP-Failure

In the case where server authentication is unsuccessful in PEAP Part 1, the conversation will appear as follows:

```

Authenticating Peer      Authenticator
-----
EAP-Response/
Identity (MyID) ->
<- EAP-Request/
Identity

EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->
<- EAP-Request/
EAP-Type=PEAP, V=0
(PCAP Start)

EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->
<- EAP-Request/
EAP-Type=PEAP, V=0
(TLS server_hello,
TLS certificate,
[TLS server_key_exchange,]
TLS server_hello_done)

EAP-Response/
```



```
EAP-Type=PEAP, V=0
(TLS client_key_exchange,
 [TLS certificate_verify,]
  TLS change_cipher_spec,
  TLS finished) ->
<- EAP-Request/
EAP-Type=PEAP, V=0
(TLS change_cipher_spec,
 TLS finished)
```

```
EAP-Response/
EAP-Type=PEAP, V=0
(TLS change_cipher_spec,
 TLS finished)
```

```
<- EAP-Request/
EAP-Type=PEAP, V=0
EAP-Response/
EAP-Type=PEAP, V=0
(TLS Alert message) ->
```

```
<- EAP-Failure
(TLS session cache entry flushed)
```

In the case where a previously established session is being resumed, the EAP server supports TLS session cache flushing for unsuccessful PEAP Part 2 authentications and both sides authenticate successfully, the conversation will appear as follows:

```
Authenticating Peer      Authenticator
-----
<- EAP-Request/
Identity
EAP-Response/
Identity (MyID) ->
<- EAP-Request/
EAP-Type=PEAP, V=0
(PEAP Start)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->
<- EAP-Request/
EAP-Type=PEAP, V=0
(TLS server_hello,
 TLS change_cipher_spec
 TLS finished)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS change_cipher_spec,
```



```
TLS finished) ->
                <- EAP-Request/
                   EAP-Type=Extensions
                   Result=Success
EAP-Response/
EAP-Type=Extensions
Result=Success ->
TLS channel torn down
(messages sent in clear-text)

                <- EAP-Success
```

In the case where a previously established session is being resumed, and the server authenticates to the client successfully but the client fails to authenticate to the server, the conversation will appear as follows:

```
Authenticating Peer   Authenticator
-----
                        <- EAP-Request/
                           Identity
EAP-Response/
Identity (MyID) ->
                        <- EAP-Request/
                           EAP-Request/
                           EAP-Type=PEAP, V=0
                           (TLS Start)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello) ->
                        <- EAP-Request/
                           EAP-Type=PEAP, V=0
                           (TLS server_hello,
                            TLS change_cipher_spec,
                            TLS finished)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS change_cipher_spec,
 TLS finished) ->
                        <- EAP-Request
                           EAP-Type=PEAP, V=0
                           (TLS Alert message)
EAP-Response
EAP-Type=PEAP, V=0 ->
                        <- EAP-Failure
                           (TLS session cache entry flushed)
```

In the case where a previously established session is being resumed,





and the server authentication is unsuccessful,  
the conversation will appear as follows:

```
Authenticating Peer      Authenticator
-----
EAP-Response/           <- EAP-Request/
Identity (MyID) ->      Identity

EAP-Response/           <- EAP-Request/
Identity (MyID) ->      EAP-Request/
                        EAP-Type=PEAP, V=0
                        (TLS Start)

EAP-Response/           <- EAP-Request/
EAP-Type=PEAP, V=0      EAP-Type=PEAP, V=0
(TLS client_hello)->    (TLS server_hello,
                        TLS change_cipher_spec,
                        TLS finished)

EAP-Response/           <- EAP-Request/
EAP-Type=PEAP, V=0      EAP-Type=PEAP, V=0
(TLS change_cipher_spec, TLS finished)

EAP-Response/           <- EAP-Request/
EAP-Type=PEAP, V=0      EAP-Type=PEAP, V=0
(TLS Alert message) ->

EAP-Response/           <- EAP-Failure
(TLS session cache entry flushed)
```

In the case where the peer and authenticator have mismatched PEAP versions (e.g. the peer has a pre-standard implementation with version 0, and the authenticator has a Version 1 implementation, but the authentication is unsuccessful, the conversation will occur as follows:

```
Authenticating Peer      Authenticator
-----
EAP-Response/           <- EAP-Request/
Identity (MyID) ->      Identity

EAP-Response/           <- EAP-Request/
Identity (MyID) ->      EAP-Request/
                        EAP-Type=PEAP, V=1
```



```
(TLS Start)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->
    <- EAP-Request/
    EAP-Type=PEAP, V=0
    (TLS server_hello,
    TLS change_cipher_spec,
    TLS finished)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS change_cipher_spec,
TLS finished)
    <- EAP-Request/
    EAP-Type=PEAP, V=0
EAP-Response/
EAP-Type=PEAP, V=0
(TLS Alert message) ->
(TLS session cache entry flushed)
    <- EAP-Failure
```

#### Acknowledgments

Thanks to Narendra Gidwani of Microsoft for useful discussions of this problem space.

#### Author Addresses

Vivek Kamath  
Ashwin Palekar  
Mark Wodrich  
Microsoft Corporation  
One Microsoft Way  
Redmond, WA 98052

Phone: +1 425 882 8080  
EMail: {vivek, ashwinp, markwo}@microsoft.com

#### Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-



related documentation can be found in [BCP-11](#). Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested Party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

#### Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved. This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in Part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and 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."

#### Expiration Date

This memo is filed as [<draft-kamath-pppext-peapv0-00.txt>](#), and expires April 23, 2002.

