EAP Working Group
INTERNET-DRAFT
Category: Informational
<draft-josefsson-pppext-eap-tls-eap-10.txt>
15 October 2004

Ashwin Palekar
Dan Simon
Microsoft Corporation
Joe Salowey
Hao Zhou
Glen Zorn
Cisco Systems
S. Josefsson
Extundo

Protected EAP Protocol (PEAP) Version 2

Copyright Notice

Abstract

The Extensible Authentication Protocol (EAP) provides support for multiple authentication methods. This document defines the Protected Extensible Authentication Protocol (PEAP) Version 2, which provides an encrypted and authenticated tunnel based on transport layer security (TLS) that encapsulates EAP authentication mechanisms. PEAPv2 uses TLS to protect against rogue authenticators, protect against various attacks on the confidentiality and integrity of the

inner EAP method exchange and provide EAP peer identity privacy. PEAPv2 also provides support for chaining multiple EAP mechanisms, cryptographic binding between authentications performed by inner EAP mechanisms and the tunnel, exchange of arbitrary parameters (TLVs), and fragmentation and reassembly.

Table of Contents

1. Introduction

The Extensible Authentication Protocol (EAP), defined in [RFC3748], provides support for multiple authentication methods. EAP over PPP, defined in [RFC3748], is typically deployed with leased lines or modem connections. [IEEE8021X] defines EAP over IEEE 802 local area networks (EAPOL).

Since its deployment, a number of weaknesses in EAP framework have become apparent. These include lack of support for:

- Identity protection
- Protected method negotiation
- Protected notification messages
- Protected termination messages
- Sequences of EAP methods
- Fragmentation and reassembly
- Exchange of arbitrary parameters in a secure channel
- Optimized re-authentication

In addition, some EAP methods lack the following features:

- Mutual authentication
- Resistance to dictionary attacks
- Adequate key generation

By wrapping the EAP protocol within TLS, Protected EAP (PEAP) Version 2 addresses deficiencies in EAP or EAP methods. Benefits of PEAP Version 2 include:

Identity protection
: By encrypting the identity exchange, and allowing client identity to be provided after negotiation of the TLS channel, PEAPv2 provides for identity protection.

Dictionary attack resistance
: By conducting the EAP conversation within a TLS channel, PEAPv2 protects EAP methods that might be subject to an offline dictionary attack were they to be conducted in the clear.

Protected negotiation
: Since within PEAPv2, the EAP conversation is authenticated, integrity and replay protected on a per-packet basis, the EAP method negotiation that occurs within PEAPv2 is protected, as are error messages sent within the TLS channel (TLS alerts or EAP Notification packets). EAP negotiation outside of PEAPv2 is not protected.

Header protection
: Within PEAPv2, TLS provides per-packet encryption, authentication, integrity and replay protection for the EAP conversation. As a result, the Type-Data field within PEAPv2 (including the EAP header of the EAP method within PEAPv2) is protected against modification. However, the EAP header of PEAPv2 itself is not protected against modification, including the Code, Identifier and Type fields.

Protected termination
: By sending success/failure indications within the TLS channel, PEAPv2 provides support for protected termination of the EAP conversation. This prevents an attacker from carrying out denial of service attacks by spoofing EAP Failure messages, or fooling the EAP peer into accepting a rogue NAS, by spoofing EAP Success messages.

Fragmentation and Reassembly
: Since EAP does not include support for fragmentation and reassembly, individual methods need to include this capability. By including support for fragmentation and reassembly within PEAPv2, methods leveraging PEAPv2 do not need to support this on their own.

Fast reconnect
: Where EAP is used for authentication in wireless networks, the authentication latency is a concern. As a result, it is valuable to be able to do a quick re-authentication on roaming between access points. PEAPv2 supports this capability by leveraging the TLS session resumption facility, and any EAP method running under PEAPv2 can take advantage of it.

Standard key establishment
: In order to provide keying material for a wide range of link layer ciphersuites, EAP methods need to provide keying material. Key derivation is complex. PEAPv2 provides for key establishment by relying on the widely implemented and well-reviewed TLS [RFC2246] key derivation mechanism. PEAPv2 provides keying material for any EAP method running within it. If EAP methods will also be deployed without external protection (e.g PEAPv2 or IPSec), then the EAP methods should follow the guidelines in Section 5.8 to prevent the man-in-the-middle attacks.

Sequencing of multiple EAP methods
: In order to enhance security, PEAPv2 implementations may choose to provide multi-factor authentication that validates different identities (for example user and machine identities) and/or uses different credentials of the same or different identities of the peer (e.g. user password and machine cert). PEAPv2 provides a standard way to chain different types of authentication mechanisms

supporting different types of credentials.

Protected exchange of arbitrary parameters
: Type-Length-Value (TLV) tuples provide a way to exchange arbitrary information between peer and EAP server within a secure channel. This information can include signaling parameters for EAP protocol, provisioning parameters, media specific and environment specific data, and authorization parameters. The advantage of using PEAP TLVs is that every EAP method does not have to be modified.

Credential provisioning
: PEAPv2 supports provisioning of certificate trust anchors by the server using TLVs and can be extended to support provisioning of other peer credentials.

Optimized for light weight devices
: In order to support peers that may not support certificate ciphersuites, and may not support provisioning of certificate trust anchors, PEAPv2 enables negotiation of other TLS ciphersuites.

Server unauthenticated tunnel provisioning mode
: In some cases, the peer may only support password credentials and may not be provisioned with a certificate trust anchor.

  In server unauthenticated tunnel provisioning mode, a PEAPv2 peer can authenticate using a password, in order to be provisioned with a pre-shared key and other credentials that can be used for subsequent authentication. In server unauthenticated tunnel provisioning mode the PEAPv2 peer only confirms possession of the private key corresponding to the public key contained within the server certificate, but does not otherwise validate the server certificate. As a result, it is possible for an attacker to act as a man-in-the-middle during the initial exchange in order to perform an offline dictionary attack, based on capture of the password-based authentication exchange.

  In PEAPv2, implementation of server unauthenticated tunnel provisioning mode is optional, and due to the security vulnerabilities introduced by this mode, it is not recommended for use with peers that support certificate validation and provisioning of certificate trust anchors.

1.1. Requirements Language

In this document, the key words "MAY", "MUST, "MUST NOT", "OPTIONAL", "RECOMMENDED", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [RFC2119].

1.2. Terminology

This document frequently uses the following terms:

Access Point
: A Network Access Server implementing 802.11.

Authenticator
: The end of the link initiating EAP authentication. This term is also used in [IEEE8021X] and has the same meaning in this document.

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 [IEEE8021X].

EAP server
: The entity that terminates the EAP authentication method with the peer. In the case where no backend authentication server is used, the EAP server is part of the Authenticator. In the case where the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.

Link layer ciphersuite
: The ciphersuite negotiated for use at the link layer.

NAS
: Short for "Network Access Server".

Peer
: The end of the link that responds to the authenticator. In [IEEE8021X], this end is known as the Supplicant.

TLS Ciphersuite
: The ciphersuite negotiated for protection of the PEAPv2 Part 2 conversation.

EAP Master key (MK)
: A key derived between the PEAPv2 client and server during the authentication conversation, and that is kept local to PEAPv2 and not exported or made available to a third party.

Master Session Key (MSK)
: Keying material that is derived between the EAP peer and server and exported by the EAP method. The MSK is at least 64 octets in length. In existing implementations, a AAA server acting as an EAP server transports the MSK to the authenticator.

Extended Master Session Key (EMSK)
: Additional keying material derived between the EAP client and server that is exported by the EAP method. The EMSK is at least 64 octets in length. The EMSK is not shared with the authenticator or any other third party. The EMSK is reserved for future uses that are not defined yet.

Type Length Value (TLV)
: The PEAPv2 protocol utilizes objects in Type Length Value (TLV) format. The TLV format is defined in Section 4 of this document.

1.3. Operational Model

In EAP, the EAP server may be implemented either within a Network Access Server (NAS) or on a backend authentication server. Where the EAP server resides on a NAS, the NAS is required to implement the desired EAP methods, and therefore needs to be upgraded to support each new EAP method.

One of the goals of EAP is to enable development of new authentication methods without requiring deployment of new code on the Network Access Server (NAS). Where a backend authentication server is deployed, the NAS acts as a "passthrough" and need not understand specific EAP methods.

This allows new EAP methods to be deployed on the EAP peer and backend authentication server, without the need to upgrade code residing on the NAS.

Figure 1 describes the relationship between the EAP peer, NAS and EAP server. As described in the figure, the EAP conversation occurs between the EAP peer and EAP server, "passing through" the NAS. In order for the conversation to proceed in the case where the NAS and EAP server reside on separate machines, the NAS and EAP server need to establish trust beforehand.

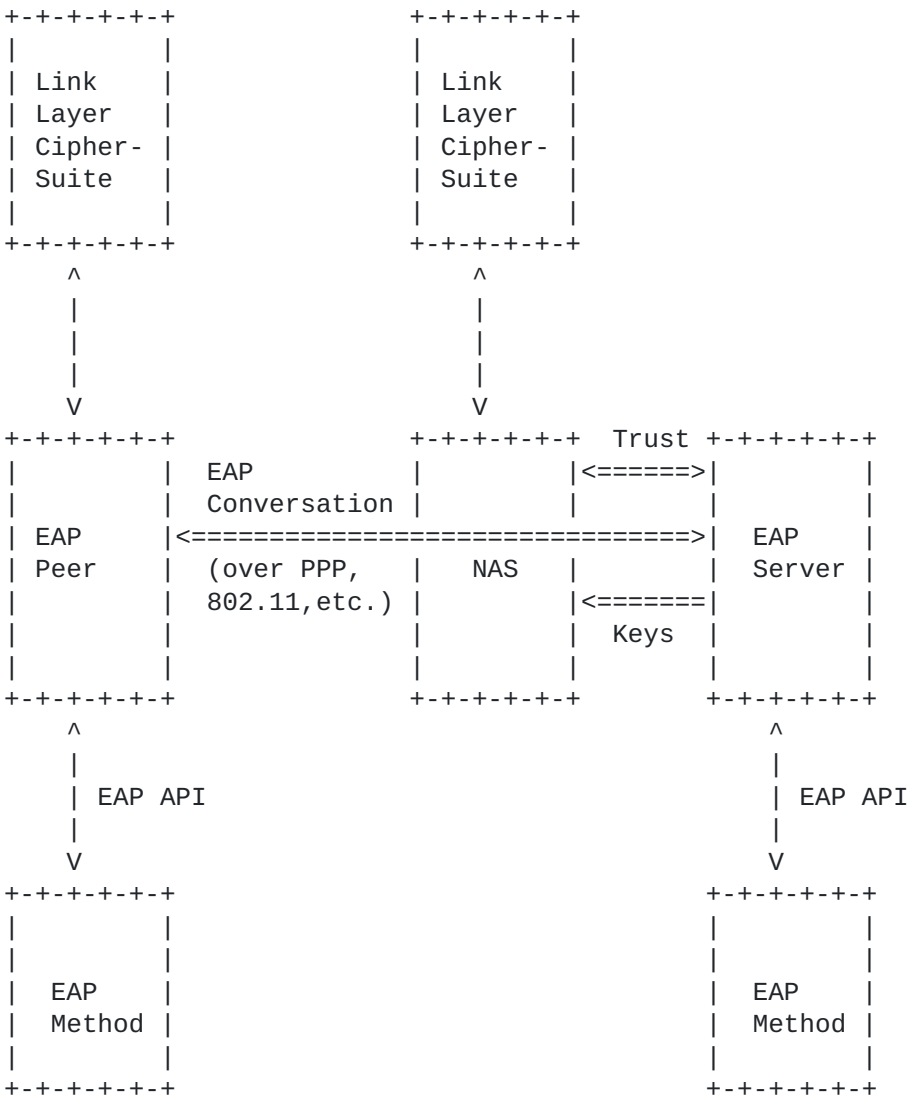


Figure 1 - Relationship between EAP client, backend authentication server and NAS.

In PEAPv2, the conversation between the EAP peer and the EAP server is encrypted, authenticated, integrity and replay protected within a TLS channel.

As a result, where the NAS acts as a "passthrough" it does not have knowledge of the TLS master secret derived between the peer and the EAP server. In order to provide keying material for link-layer ciphersuites, the NAS obtains the master session key, which is derived from a one-way function of the TLS master secret as well as keying material provided by EAP methods protected within a TLS channel. This enables the NAS and EAP peer to subsequently derive transient session keys suitable for encrypting, authenticating and integrity protecting session data. However, the NAS cannot decrypt the PEAPv2 conversation or spoof session resumption, since this requires knowledge of the TLS master secret.

1.3.1. Sequences

EAP [RFC3748] prohibits use of multiple authentication methods within a single EAP conversation, except when tunneled methods such as PEAPv2 are used. This restriction was imposed in order to limit vulnerabilities to man-in-the-middle attacks as well as to ensure compatibility with existing EAP implementations.

Within PEAP these concerns are addressed since PEAPv2 includes support for cryptographic binding to address man-in-the-middle attacks, as well as version negotiation so as to enable backward compatibility with prior versions of PEAP.

Within this document, the term "sequence" refers to a series of EAP authentication methods run in sequence or TLV exchanges before or after EAP methods. The methods need not be distinct - for example, EAP-TLS could be run initially with machine credentials followed by the same protocol authenticating with user credentials.

PEAPv2 supports initiating additional EAP method(s) after a successful or a failed EAP method. The result of failure of a EAP method does not always imply a failure of the overall authentication. The overall result of authentication depends on the policy at EAP server and the peer. For example, successful authentication might require a successful machine authentication followed by a successful user authentication. Alternatively, if machine authentication fails, then user authentication can be attempted. PEAP does not support initiating multiple EAP methods simultaneously.

2. Protocol Overview

Protected EAP (PEAP) Version 2 is comprised of a two-part conversation:

[1] In Part 1, a TLS session is negotiated, with server authenticating to the client and optionally the client to the server. The negotiated key is then used to encrypt the rest of the conversation.

[2] In Part 2, within the TLS session, zero or more EAP methods are carried out. Part 2 completes with a success/failure indication protected by the TLS session or a protected error (TLS alert).

In the next two sections, we provide an overview of each of the parts of the PEAPv2 conversation.

2.1. PEAPv2 Part 1

2.1.1. Initial identity exchange

The PEAP conversation typically begins with an optional identity exchange. The authenticator will typically send an EAP-Request/Identity packet to the peer, and the peer will respond with an EAP-Response/Identity packet to the authenticator.

The initial identity exchange is used primarily to route the EAP conversation to the EAP server. Since the initial identity exchange is in the clear, the peer MAY decide to place a routing realm instead of its real name in the EAP-Response/Identity. The real identity of the peer can be established later in PEAPv2 part 2.

If the EAP server is known in advance (such as when all users authenticate against the same backend server infrastructure and roaming is not supported), or if the identity is otherwise determined (such as from the dialing phone number or client MAC address), then the EAP-Request/Response-identity exchange MAY be omitted.

Once the optional initial Identity Request/Response exchange is completed, while nominally the EAP conversation occurs between the authenticator and the peer, the authenticator MAY act as a passthrough device, with the EAP packets received from the peer being encapsulated for transmission to a backend authentication server. However, PEAP does not require a backend authentication server; if the authenticator implements PEAP, then it can authenticate local users.

In the discussion that follows, we will use the term "EAP server" to

denote the ultimate endpoint conversing with the peer.

2.1.2. TLS Session Establishment

In this section, the protocol is described, and in Appendix A, examples of protocol exchanges are provided. While this section and the examples in Appendix A often describe negotiation of a certificate-based ciphersuite within TLS, PEAPv2 supports negotiation of other ciphersuites (for example, ciphersuites that do not use certificates) or extensions. However, the conversation may slightly differ if other TLS ciphersuites or extensions are used.

Once having received the peer's Identity, and determined that PEAP authentication is to occur, the EAP server MUST respond with a PEAP/Start packet, which is an EAP-Request packet with EAP-Type=PEAP, the Start (S) bit set, the PEAP version as specified in Section 2.1.5, and optionally, the Server-Identifier TLV.

Assuming that the peer supports PEAP, the PEAP conversation will then begin, with the peer sending an EAP-Response packet with EAP-Type=PEAP. The Type-Data field of the EAP-Response Packet will encapsulate one or more TLS records containing the TLS handshake messages. As defined in [RFC2246], the TLS handshake is used to negotiate parameters and cryptographic keys and may take several roundtrips between the TLS client and server.

The version offered by the TLS client and server MUST be TLS v1.0 or later. PEAPv2 implementations need not necessarily support all TLS ciphersuites listed in [RFC2246]. Not all TLS ciphersuites are supported by available TLS tool kits and licenses may be required in some cases.

To ensure interoperability, PEAPv2 peers and servers MUST support the TLS v1.0 [RFC2246] mandatory-to-implement ciphersuite:

```
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
```

In addition, PEAPv2 servers SHOULD support and be able to negotiate all of the following TLS ciphersuites:

```
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
```

In addition, PEAPv2 peers SHOULD support at least one of the following TLS ciphersuites:

```
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
```

TLS as described in [RFC2246] supports compression as well as ciphersuite negotiation. Therefore during the PEAPv2 Part 1 conversation the PEAPv2 endpoints MAY request or negotiate TLS compression.

If the full TLS handshake is performed, then the first payload of PEAPv2 part 2 MAY be sent along with finished handshake message to reduce number of round trips.

Since after the TLS session is established, another complete EAP negotiation will occur and the peer will authenticate using a secondary mechanism, with PEAPv2 the client need not authenticate as part of TLS session establishment.

Note that since TLS client certificates are sent in the clear, if identity protection is required, then it is possible for the TLS authentication to be re-negotiated after the first server authentication. Alternatively, if identity protection is required, then it is possible to perform certificate authentication using a EAP method (for example: EAP-TLS) within the TLS session in PEAPv2 part 2.

To accomplish this, the server will typically not request a certificate in the server_hello, then after the server_finished message is sent, and before PEAP part 2, the server MAY send a TLS hello_request. This allows the client to perform client authentication by sending a client_hello if it wants to, or send a no_renegotiation alert to the server indicating that it wants to continue with PEAP part 2 instead. Assuming that the client permits renegotiation by sending a client_hello, then the server will respond with server_hello, a certificate and certificate_request messages. The client replies with certificate, client_key_exchange and certificate_verify messages. Since this re-negotiation occurs within the encrypted TLS channel, it does not reveal client certificate details.

2.1.3. Session Resumption

The purpose of the sessionId within the TLS protocol and the Server-Identifier TLV in PEAP is to allow for improved efficiency in the case where a client repeatedly attempts to authenticate to an EAP server within a short period of time. This capability is particularly useful for support of wireless roaming.

In order to help the peer choose a sessionID that belongs to the specific server, the EAP server MAY send an identifier (Server-Identifier TLV) that the peer can use as a hint. The Server-Identifier TLV MAY be sent in the first PEAP packet from the EAP server to the peer. In order to detect modification of the Server-Identifier TLV, the Server-Identifier TLV is included in calculation of the compound MAC.

It is left up to the peer whether to attempt to continue a previous session, thus shortening the PEAP Part 1 conversation. Typically the peer's decision will be made based on the time elapsed since the previous authentication attempt to that EAP server.

Based on the sessionId chosen by the peer, and the time elapsed since the previous authentication, the EAP server will decide whether to allow the continuation, or whether to choose a new session.

If the EAP server is resuming a previously established session, then it MUST include only a TLS change_cipher_spec message and a TLS finished handshake message after the server_hello message. The finished message contains the EAP server's authentication response to the peer.

If the preceding server_hello message sent by the EAP server in the preceding EAP-Request packet indicated the resumption of a previous session, then the peer MUST send only the change_cipher_spec and finished handshake messages. The finished message contains the peer's authentication response to the EAP server. The latter contains the EAP server's authentication response to the peer. The peer will verify the hash in order to authenticate the EAP server.

If authentication fails, then the peer and EAP-server MUST follow the error handling behavior specified in section 2.3.

Even if the session is successfully resumed with the same EAP server, the peer and EAP server MUST NOT assume that either will skip inner EAP methods. The peer may have roamed to a network which may use the same EAP server, but may require conformance with a different authentication policy, and therefore may require inner EAP authentication methods.

2.1.4. Version Negotiation

PEAP packets contain a three bit version field, which enables PEAP implementations to be backward compatible with previous versions of the protocol. This specification documents the PEAP version 2 protocol; implementations of this specification MUST use a version field set to 2. Version negotiation proceeds as follows:

[1] In the first EAP-Request sent with EAP-Type=PEAP, the EAP server MUST set the version field to the highest supported version number.

[2] If the EAP peer supports this version of the protocol, it MUST respond with an EAP-Response of EAP-Type=PEAP, and the version number proposed by the EAP server.

[3] If the EAP peer does not support this version, it responds with an EAP-Response of EAP-Type=PEAP and the highest supported version number.

[4] If the PEAP server does not support the version number proposed by the PEAP peer, it either starts a different EAP type or terminates the conversation by sending an EAP-Failure, depending on the server policy.

The version negotiation procedure guarantees that the EAP peer and server will agree to the latest version supported by both parties. If version negotiation fails, then use of PEAP will not be possible, and another mutually acceptable EAP method will need to be negotiated if authentication is to proceed.

The PEAP version field is not protected by TLS and therefore can be modified in transit. In order to detect modification of the PEAP version which could occur as part of a "downgrade" attack, the peer and EAP server check if the version it sent during negotiation is same as the version claimed to be received by the other party. Each party uses the Crypto-Binding TLV to inform the other party of the version number it received during the PEAP version negotiation. The receiver of the Crypto-Binding TLV must verify that the version in the Crypto-Binding TLV matches the version it sent during PEAP version negotiation.

2.2. PEAPv2 Part 2

The second portion of the PEAPv2 conversation typically consists of a complete EAP conversation occurring within the TLS session negotiated in PEAPv2 Part 1, ending with protected termination using the Result TLV. PEAPv2 part 2 will occur only if establishment of a new TLS session in Part 1 is successful or a TLS session is successfully resumed in Part 1. In cases where a new TLS session is established in PEAPv2 part 1, the first payload of the part 2 conversation MAY be sent by the EAP server along with the finished message to save a round-trip.

Part 2 SHOULD NOT occur if the EAP Server authenticates unsuccessfully, and MUST NOT occur if establishment of the TLS session in part 1 was not successful Or a TLS fatal error has been

sent terminating the conversation.

Since all packets sent within the PEAPv2 Part 2 conversation occur after TLS session establishment, they are protected using the negotiated TLS ciphersuite. All EAP packets of the EAP conversation in part 2 including the EAP header of the inner EAP method are protected using the negotiated TLS ciphersuite.

Part 2 MAY NOT always include a EAP conversation within the TLS session, referred to in this document as inner EAP methods. However, Part 2 MUST always end with either protected termination or protected error termination (e.g. TLS alert).

Within Part 2, protected EAP conversation and protected termination packets are always carried within TLVs. There are TLVs defined for specific purposes such as carrying EAP-authentication messages and carrying cryptographic binding. New TLVs may be developed for other purposes.

2.2.1. Protected Conversation

Part 2 of the PEAPv2 conversation typically begins with the EAP server sending an optional EAP-Request/Identity packet to the peer, protected by the TLS ciphersuite negotiated in PEAPv2 Part 1. The peer responds with an EAP-Response/Identity packet to the EAP server, containing the peer's userId. Since this Identity Request/Response exchange is protected by the ciphersuite negotiated in TLS, it is not vulnerable to snooping or packet modification attacks.

After the TLS session-protected Identity exchange, the EAP server will then select authentication method(s) for the peer, and will send an EAP-Request with the Type field set to the initial method. As described in [RFC3748], the peer can NAK the suggested EAP method, suggesting an alternative. Since the NAK will be sent within the TLS channel, it is protected from snooping or packet modification. As a result, an attacker snooping on the exchange will be unable to inject NAKs in order to "negotiate down" the authentication method. An attacker will also not be able to determine which EAP method was negotiated.

The EAP conversation within the TLS protected session may involve a sequence of zero or more EAP authentication methods; it completes with the protected termination described in Section 2.2.2. Several TLVs may be included in each Request and Response. EAP methods are always encapsulated within EAP Payload-TLV.

In a typical EAP conversation, the result of the conversation is communicated by sending EAP Success or EAP Failure packets after the

EAP method is complete. The EAP Success or Failure packet is considered the last packet of the EAP conversation; and therefore cannot be used when sequences need to be supported. Hence, instead of using the EAP-success or EAP-failure packet, both peer and EAP server MUST use the Intermediate Result TLV to communicate the result.

In a typical EAP conversation, the EAP Success or EAP Failure is considered the last packet of the EAP conversation. Within PEAPv2, the EAP server can start another EAP method after success or failure of the previous EAP method inside the protected session.

In a sequence of more than one EAP authentication method, to make sure the same parties are involved in tunnel establishment and successful completion of previous inner EAP methods, before completing negotiation of the next EAP method, both peer and EAP server MUST use crypto binding (Crypto-Binding TLV).

The Intermediate-Result TLV is used to indicate the result of a individual successful EAP method, and the Result TLV is used to indicate result of the entire PEAP conversation.

The Intermediate-Result TLV and Crypto-Binding TLV MUST be sent after each EAP method that was successful. If the EAP method failed, or if the EAP method negotiation did not complete, then an Intermediate-Result TLV MAY be included, and the Crypto-Binding TLV MUST NOT be included. An exception is that the Crypto-Binding TLV MUST be sent along with a protected success/failure indication as described in Section 2.2.2.

If these TLVs are not sent after a successful EAP method, it should be considered a tunnel compromise error by peer and EAP server, resulting in terminating the conversation as described in Section 2.3.

A subsequent EAP conversation can be started after both TLVs are exchanged in a TLV packet. Alternatively, if a subsequent EAP conversation is being attempted, then in order to reduce round trips, both TLVs SHOULD be sent with the EAP-Payload of the first EAP packet of the next EAP conversation (for example, EAP- Identity or EAP- packet of the EAP method). Alternatively, if the next packet is the protected success/failure packet, then in order to reduce round trips, both TLVs MUST be sent with the protected success/failure packet.

If the EAP server sends a valid Crypto-Binding TLV to the peer, the peer MUST respond with a Crypto-Binding TLV. If the Crypto-Binding TLV is invalid, it should be considered a tunnel compromise error by

the peer. If the peer does not respond with a TLV packet containing the Crypto-Binding TLV, it MUST be considered a tunnel compromise error by the EAP server.

Within a PEAPv2 part 2 conversation, a peer MAY request the trusted root of a server certificate using a Server-Trusted-Root TLV, and the EAP server MAY respond with a Server-Trusted-Root TLV to the peer. The Server-Trusted-Root can be exchanged in regular authentication mode or server unauthenticated tunnel provisioning mode.

After the peer has determined that it has successfully authenticated the EAP server and determined that the tunnel and inner EAP methods were between the same peer and EAP server by validating the Crypto-Binding TLV, it MAY send one or more Server-Trusted-Root TLVs (marked as optional) to request the trusted root of server certificate from the EAP server. The peer may receive a response, but is not required to use the trusted root received from the EAP server.

If the EAP server has determined that it has successfully authenticated the peer and determined that the tunnel and inner EAP methods were between the same peer and EAP server by validating the Crypto-Binding TLV, then it MAY respond with the the server-trusted-root containing the PCKS#7 TLV.

2.2.2. Protected Termination

The PEAPv2 part 2 conversation is completed by exchanges of success/failure indications (Result TLV) within a TLV packet protected by the TLS session.

Even if Crypto-Binding TLVs have been exchanged in previous conversations, the Crypto-Binding TLV MUST be included in both protected success/failure (Result TLV) indications. If the TLVs are not included, or if the TLVs are invalid, it should be considered a tunnel compromise error, and the peer and EAP server MUST follow the rules described in Section 2.3 to abort the conversation.

The Result TLV is sent within the TLS channel. The PEAP client then replies with a Result TLV. The conversation concludes with the PEAP server sending a cleartext success/failure indication.

The only outcome which should be considered as successful authentication is when a Result TLV of Status=Success is answered by the peer with a Result TLV of Status=Success.

The combinations (Result TLV=Failure, Result TLV=Success), (Result TLV=Failure, Result TLV=Failure), (no TLVs exchange or no protected

success or failure) should be considered an authentication failure by both the peer and EAP server. Once the peer and EAP server consider that authentiation has failed, these are the last packets inside the protected tunnel. These combinations are considered an authentication failure regardless of whether a cleartext EAP Success or EAP Failure packet is subsequently sent.

If the EAP server wants authentication to fail, it sends the TLV response with Result TLV=Failure. If the EAP server sends a failure, the peer MUST respond with Result TLV=Failure and the Crypto-Binding TLV, without any other mandatory TLVs. The Crypto-Binding TLV is calculated using the key derivation formula in Section 2.5; if for some reason one or more inner EAP method MSKs were not derived, then these MSKs are assumed to be null.

If the EAP server has sent the success indication (Result TLV=Success), the peer is allowed to refuse to accept a Success message from the EAP server since the client's policy may require completion of certain EAP methods or the client may require credentials.

If the EAP server has sent a success indication (Result TLV=success), and the peer wants authentication to fail, it sends the TLV response with Result TLV=Failure and Crypto-Binding TLV.

After the EAP-server returns success, if the peer wants to request the EAP server to continue conversation, it sends a Result TLV=Success along with a Request-Action TLV with the appropriate action (e.g. Negotiate-EAP, or Process-TLV). If the Request-Action TLV is set to mandatory, then the EAP server MUST process the action, or return status=failure, closing the conversation inside the tunnel. If the Request-Action TLV is set to optional, then the EAP server can ignore the TLV and return Result TLV=Success again, closing the conversation inside the tunnel.

2.2.3. Provisioning of Credentials

PEAPv2 supports built-in provisioning of certificate trust anchors and can be extended to provisioning of other types of credentials. The following two provisioning modes are suported:

[a] Provisioning inside a server authenticated TLS tunnel.

After regular authentication in PEAP part 2, the peer and EAP server can use the Server-Trusted-Root TLV to request and provision peer credentials. The provisioning payload is exchanged after the peer and EAP server have determined that both have successfully authenticated each other (either thru TLS handshake and/or inner

EAP method), and the tunnel and inner EAP methods are between the same peers.

After the peer has determined that it has successfully authenticated the EAP server and determined that the tunnel and inner EAP methods were between the same peer and EAP server by validating the Crypto-Binding TLV, it MAY send one or more Server-Trusted-Root TLVs (marked as optional) to request credentials from the EAP server. The EAP server will send corresponding credentials in the Server-Trusted-Root TLVs if its internal policy has been satisfied. It may ignore the credential provisioning request or request additional authentication methods if its policy dictates. The peer may receive a credential, but is not required to use the credentials received from the EAP server.

[b] Provisioning inside a server unauthenticated TLS tunnel.

In some cases, peer lacks the credentials to authenticate the server in the TLS handshake. At the same time, bootstrapping the information to the peer out of band may be prohibitive from a deployment cost perspective. It can rely on the inner EAP method using existing credentials to authenticate the server. This provisioning mode provides ease of deployment at the cost of introducing man-in-the-middle vulnerabilities. As a result, implementation of the server unauthenticated tunnel provisioning mode is OPTIONAL.

In this provisioning mode, as part of PEAPv2 part 1, if the peer does not authenticate, or does not successfully authenticate the EAP server during TLS negotiation, it can decide to go into server unauthenticated tunnel provisioning mode. While this section describes negotiation of a certificate-based ciphersuite within TLS, PEAPv2 supports negotiation of other ciphersuites (for example, ciphersuites that do not use certificates such as anonymous DH) or extensions. However, the conversation may slightly differ if other TLS ciphersuites or extensions are used. For example, in a certificate based TLS handshake, the peer verifies that the EAP server possesses the private key corresponding to the public key contained in the certificate presented by the EAP server. However, the peer does not verify whether the certificate presented by the server chains to a provisioned trust anchor, as the peer may not be configured with a certificate trust anchor required to validate the server certificate. If the peer cannot verify that the server possesses the corresponding private key, or if the certificate presented by the server is unacceptable for any reason other than the lack of an appropriate trust anchor, the peer MUST NOT use this provisioning mode. Assuming that the server demonstrates possession of the

private key, the peer continues with establishment of the tunnel (PEAPv2 part 2). As a result, it is possible that the TLS channel (PEAPv2 part 2) may be terminated by an attacker.

The PEAPv2 Part 2 conversation is unchanged in this mode, except that the peer will only accept an EAP method supporting mutual authentication and key derivation that is compatible with its initial credentials (such as a password-based EAP method). The peer then uses the Crypto-Binding TLV to validate that the same server terminates both the TLS channel and the successfully completed EAP method, thereby verifying that the exchange was not subject to a man-in-the-middle attack. Assuming that the Crypto-Binding TLV exchange is successful, the peer will request and the server will subsequently provide a trusted root, using the Server-Trusted-Root TLV.

Once the initial provisioning exchange completes, the peer is expected to use the provisioned credentials in subsequent PEAPv2 authentications, and SHOULD NOT use this provisioning mode.

PEAPv2 servers implementing this provisioning mode MUST support the following additional ciphersuites, beyond those specified in Section 2.1.2:

TLS_DH_anon_WITH_AES_128_CBC_SHA

PEAPv2 peers implementing this provisioning mode MAY support the following additional ciphersuites, beyond those specified in Section 2.1.2:

TLS_DH_anon_WITH_AES_128_CBC_SHA

2.3. Error Handling

PEAPv2 does not have its own error message capabilities since:

[1] Errors in TLS layer are communicated via TLS alert messages.

[2] Errors within the EAP conversation in PEAPv2 Part 2 are expected to be handled by individual EAP methods.

[3] Violation of the TLV rules for inner-TLVs are handled using Result-TLVs together with the Error-Code TLV.

If an error occurs at any point in the TLS layer, the EAP server SHOULD send a TLS alert message instead of the next EAP-request packet to the peer. The EAP server SHOULD send an EAP-Request packet with EAP-Type=PEAP, encapsulating a TLS record containing the

appropriate TLS alert message. The EAP server SHOULD send a TLS alert message rather than immediately terminating the conversation so as to allow the peer to inform the user of the cause of the failure and possibly allow for a restart of the conversation. To ensure that the peer receives the TLS alert message, the EAP server MUST wait for the peer to reply with an EAP-Response packet.

The EAP-Response packet sent by the peer MAY encapsulate a TLS client_hello handshake message, in which case the EAP server MAY allow the PEAPv2 conversation to be restarted, or it MAY contain an EAP-Response packet with EAP-Type=PEAP and no data, in which case the PEAPv2 server MUST send an EAP-Failure packet, and terminate the conversation.

It is up to the EAP server whether to allow restarts, and if so, how many times the conversation can be restarted. An EAP server implementing restart capability SHOULD impose a limit on the number of restarts, so as to protect against denial of service attacks.

If an error occurs at any point in the TLS layer, the peer SHOULD send a TLS alert message instead of the next EAP-response packet to the EAP server. The peer SHOULD send an EAP-Response packet with EAP-Type=PEAP, encapsulating a TLS record containing the appropriate TLS alert message. The EAP server may restart the conversation by sending a EAP-Request packet encapsulating the TLS hello_request_handshake message, in which case the peer MAY allow the PEAPv2 conversation to be restarted; or the EAP server can response with EAP Failure.

Any time the peer or the EAP server finds an error when processing the sequence of exchanges, such as a violation of TLV rules, it should send a Result TLV of failure and Error-Code TLV=Unexpected_TLVs_Exchanged (a Fatal error), and terminate the tunnel. This is usually due to an implementation problem and is considered an fatal error. The party that receives the Error-Code TLV=Unexpected_TLVs_Exchanged should terminate the tunnel.

If a tunnel compromise error (see Section 2.2) is detected by the Peer or EAP server, the party SHOULD send a Result TLV of failure without a Crypto-Binding TLV, and Error-Code TLV=Tunnel-compromise-error (a Fatal error), and terminate the tunnel. The party that receives the Error-Code TLV=Tunnel-compromise error should terminate the tunnel.

2.4. Fragmentation

A single TLS record may be up to 16384 octets in length, but a TLS message may span multiple TLS records, and a TLS certificate message

may in principle be as long as 16MB.

The group of PEAPv2 messages sent in a single round may thus be larger than the PPP MTU size, the maximum RADIUS packet size of 4096 octets, or even the Multilink Maximum Received Reconstructed Unit (MRRU).

As described in [RFC1990], the multilink MRRU is negotiated via the Multilink MRRU LCP option, which includes an MRRU length field of two octets, and thus can support MRRUs as large as 64 KB.

However, note that in order to protect against reassembly lockup and denial of service attacks, it may be desirable for an implementation to set a maximum size for one such group of TLS messages. Since a typical certificate chain is rarely longer than a few thousand octets, and no other field is likely to be anywhere near as long, a reasonable choice of maximum acceptable message length might be 64 KB.

If this value is chosen, then fragmentation can be handled via the multilink PPP fragmentation mechanisms described in [RFC1990]. While this is desirable, EAP methods are used in other applications such as [IEEE80211] and there may be cases in which multilink or the MRRU LCP option cannot be negotiated. As a result, a PEAPv2 implementation MUST provide its own support for fragmentation and reassembly.

Since EAP is an ACK-NAK protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field as is provided in IPv4.

PEAPv2 fragmentation support is provided through addition of flag bits within the EAP-Response and EAP-Request packets, as well as a TLV Message Length field of four octets. Flags include the Length included (L), More fragments (M), and PEAP Start (S) bits. The L flag is set to indicate the presence of the four octet TLV Message Length field, and MUST be set only for the first fragment of a fragmented TLV message or set of messages.

The TLV Message Length field in the PEAPv2 header is not protected, and hence can be modified by a attacker. The TLS record length in the TLS data is protected. Hence, if the TLV Message length received in the first packet (with L bit set) is greater or less than the total size of TLS messages received including multiple fragments, then the TLV message length should be ignored.

In order to protect against reassembly lockup and denial of service

attacks, it may be desirable for an implementation to set a maximum size for one group of Outer-TLV messages. Since a typical certificate chain is rarely longer than a few thousand octets, and no other field is likely to be anywhere near as long, a reasonable choice of maximum acceptable message length for all the Outer-TLVs in a group of messages might be 64 KB.

The M flag is set on all but the last fragment. The S flag is set only within the PEAP start message sent from the EAP server to the peer. The TLV Message Length field is four octets, and provides the total length of the TLV message or set of messages that is being fragmented; this simplifies buffer allocation.

When a peer receives an EAP-Request packet with the M bit set, it MUST respond with an EAP-Response with EAP-Type=PEAP and no data. This serves as a fragment ACK. The EAP server MUST wait until it receives the EAP-Response before sending another fragment. In order to prevent errors in processing of fragments, the EAP server MUST increment the Identifier field for each fragment contained within an EAP-Request, and the peer MUST include this Identifier value in the fragment ACK contained within the EAP-Response. Retransmitted fragments will contain the same Identifier value.

Similarly, when the EAP server receives an EAP-Response with the M bit set, it MUST respond with an EAP-Request with EAP-Type=PEAP and no TLS data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors in the processing of fragments, the EAP server MUST increment the Identifier value for each fragment ACK contained within an EAP-Request, and the peer MUST include this Identifier value in the subsequent fragment contained within an EAP-Response.

2.5. Key Derivation

Since the normal TLS keys are used in the handshake, and therefore should not be used in a different context, new keys must be derived from the TLS master secret to protect the conversation within the PEAPv2 tunnel.

Instead of deriving keys specific to link layer ciphersuites, EAP methods provides a Master Session Key (MSK) used to derive keys in a link layer specific manner. The method used to extract ciphering keys from the MSK is beyond the scope of this document.

PEAPv2 also derives an Extended Master Session Key (EMSK) which is reserved for use in deriving keys in other ciphering applications. This draft also does not discuss the format of the attributes used to

communicate the master session keys from the backend authentication server to the NAS; examples of such attributes are provided in [RFC2548].

PEAPv2 combines key material from the TLS exchange with key material from inner key generating EAP methods to provide stronger keys and to bind inner authentication mechanisms to the TLS tunnel. Both the peer and EAP server MUST derive compound MAC and compound session keys using the procedure described below.

The input for the cryptographic binding includes the following:

[a] The PEAPv2 tunnel key (TK) is calculated using the first 40 octets of the (secret) key material generated as described in the EAP-TLS algorithm ([RFC2716] Section 3.5). More explicitly, the TK is the first 40 octets of the PRF as defined in [RFC2716]:

PRF(master secret,"client EAP encryption", random)

Where random is the concatenation of client_hello.random and server_hello.random

[b] The first 32 octets of the MSK provided by each successful inner EAP method ;for each successful EAP method completed within the tunnel.

ISK1..ISKn are the MSK portion of the EAP keying material obtained from methods 1 to n. The ISKj shall be the first 32 octets of the generated MSK of the jth EAP method. If the MSK length is less than 32 octets, it shall be padded with 0x00's to ensure the MSK is 32 octets. Similarly, if no keying material is provided for the EAP method, then ISKj shall be set to zero (e.g. 32 octets of 0x00).

The PRF algorithm is based on PRF+ from IKEv2 shown below ("|" denotes concatenation)

K = Key, S = Seed, LEN = output length, represented as binary in a single octet.

PRF (K,S,LEN) = T1 | T2 | T3 | T4 | ... where:

```
T1 = HMAC-SHA1(K, S | LEN | 0x01)
T2 = HMAC-SHA1 (K, T1 | S | LEN | 0x02)
T3 = HMAC-SHA1 (K, T2 | S | LEN | 0x03)
T4 = HMAC-SHA1 (K, T3 | S | LEN | 0x04)
  ...
```

The intermediate combined key is generated after each successful EAP

method inside the tunnel.

Generating the intermediate combined key:

S-IPMK0 = TK
for j = 1 to k do

IPMKj = PRF+(S-IPMK(j-1),"Inner Methods Compound Keys " | ISKj, 60);
Where S-IPMKj are the first 40 octets of IPMKj
and CMKj are the last 20 octets of IPMKj used to generate
the intermediate Compound MACs

k = the last successful EAP method inside the tunnel at the point where the combined MAC key is derived.

Each IPMKj output is 60 octets. The first 40 octets are used as the key input to the succeeding IPMK(j+1) derivation and the latter 20 octets are used as the key, CMKj, used to generate the intermediate Crypto-Binding Compound MAC value at the jth EAP method.

Compound Session Key derivation:

The compound session key (CSK) is derived on both the peer and EAP server.

CSK = PRF+(S-IPMKn, "Session Key Generating Function", OutputLength)

The output length of the CSK must be at least 128 bytes. The first 64 octets are taken and the MSK and the second 64 octets are taken as the EMSK. The MSK and EMSK are described in [RFC3748].

2.6. Ciphersuite Negotiation

Since TLS supports TLS ciphersuite negotiation, peers completing the TLS negotiation will also have selected a TLS ciphersuite, which includes key strength, encryption and hashing methods. However, unlike in [RFC2716], within PEAPv2, the negotiated TLS ciphersuite relates only to the mechanism by which the PEAPv2 Part 2 conversation will be protected, and has no relationship to link layer security mechanisms negotiated within the PPP Encryption Control Protocol (ECP) [RFC1968] or within IEEE 802.11 [IEEE80211].

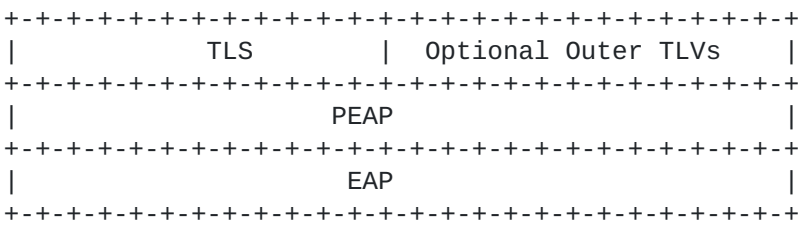
As a result, this specification currently does not support secure negotiation of link layer ciphersuites.

3. PEAPv2 Protocol Description

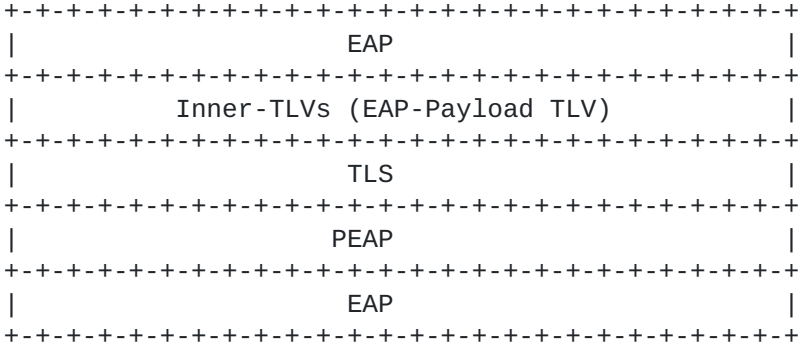
3.1. PEAPv2 Protocol Layers

PEAPv2 packets may include TLVs both inside and outside the TLS tunnel. The term "Outer TLVs" is used to refer to optional TLVs outside the TLS tunnel, which are only allowed in the first two messages in the PEAPv2 protocol. That is the first EAP server to peer message and first peer to EAP server message. If the message is fragmented, the whole set of messages is counted as one message. The term "Inner TLVs" is used to refer to TLVs sent within the TLS tunnel.

In PEAPv2 Part 1, Outer TLVs are used to help establishing the TLS tunnel, but no Inner TLVs are used. Therefore the layering of PEAPv2 Part 1 is as follows:

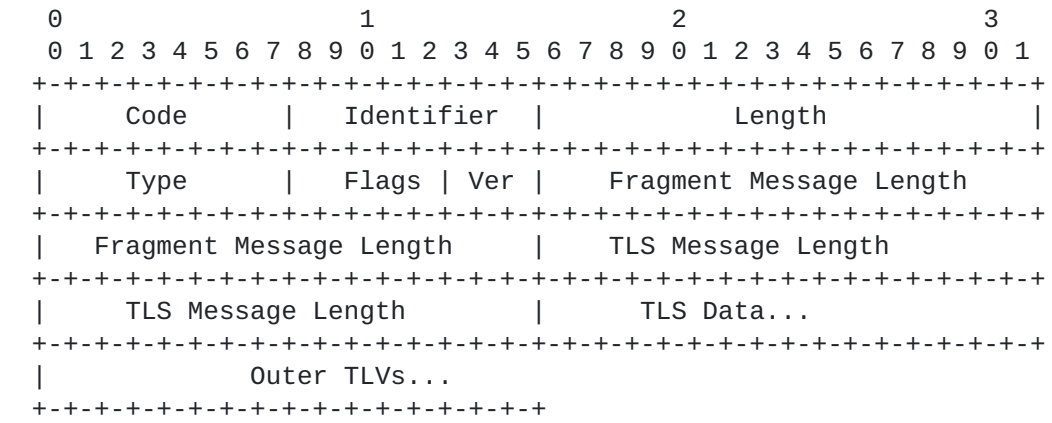


In PEAPv2 Part 2, TLS records may encapsulate zero or more Inner TLVs, but no Outer TLVs. EAP packets (including EAP header fields) used within tunneled EAP authentication methods are carried within Inner TLVs. Therefore the layering of PEAPv2 Part 2 is as follows:



3.2. PEAPv2 Packet Format

A summary of the PEAPv2 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. The Identifier field MUST be changed on each Request packet. The Identifier field in a Response packet MUST match the Identifier field from the corresponding Request.

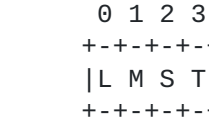
Length

The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, Flags, Version, Fragmented Length, TLS Message Length, TLS Data, and Outer-TLV fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and should be ignored on reception.

Type

25 - PEAP

Flags

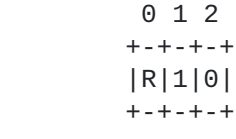


L = Length included

M = More fragments
S = PEAP start
T = TLS Length included
R = Reserved (must be zero)

The L bit (Fragmented Message Length included) is set to indicate the presence of the four octet Fragmented Message Length field, and MUST be set for the first fragment of a fragmented PEAP message or set of messages. The M bit (more fragments) is set on all but the last fragment. The S bit (PEAP start) is set in a PEAP Start message. This differentiates the PEAP Start message from a fragment acknowledgment. The T bit (TLS Message Length included) is set to indicate the presence of the four octet TLS Message Length field, and MUST only be set for packet that contains Out-TLVs. It can be used to calculate the start of the Outer-TLVs.

Version



R = Reserved (must be zero)

Fragmented Message Length

The Fragmented Message Length field is four octets, and is present only if the L bit is set. This field provides the total length of the data after the Fragmented Message Length field in the PEAP message or set of messages that is being fragmented.

TLS Message Length

The TLS Message Length field is four octets, and is present only if the T bit is set. This field provides the total length of the TLS Data in the PEAP message. Data after this length of TLS data are the Outer TLVs.

TLS Data

The TLS data consists of the encapsulated packet in TLS record format.

Outer TLVs

The Outer-TLVs consists of the optional data used to help

establishing the TLS tunnel in TLV format. The start of the Outer-TLV can be derived from the EAP Length field and TLS Length field.

4. TLVs

The TLVs used within PEAPv2 are standard Type-Length-Value (TLV) objects. The TLV objects could be used to carry arbitrary parameters between EAP peer and EAP server. Possible uses for TLV objects include: language and character set for Notification messages and cryptographic binding.

The EAP peer may not necessarily implement all the TLVs supported by the EAP server; and hence to allow for interoperability, TLVs allow an EAP server to discover if a TLV is supported by the EAP peer, using the NAK TLV. The PEAPv2 packet does not have to contain any TLVs, nor need it contain any mandatory TLVs.

The mandatory bit in a TLV indicates whether support of the TLV is required. If the peer or server does not support the TLV, it MUST send a NAK TLV in response, and all the other TLVs in the message MUST be ignored. If an EAP peer or server finds an unsupported TLV which is marked as optional, it can ignore the unsupported TLV. It MUST NOT send an NAK TLV.

Note that a peer or server may support a TLV with the mandatory bit set, but may not understand the contents. The appropriate response to a supported TLV with content that is not understood is defined by the TLV specification.

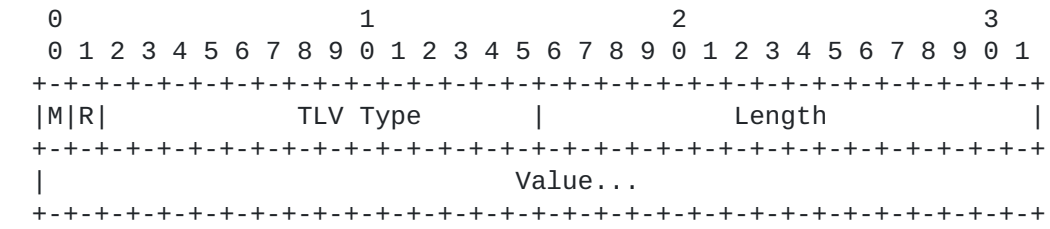
Outer-TLVs SHOULD NOT be included in messages after the first two Outer-TLV messages sent by the peer and EAP server respectively. A single Outer-TLV message may be fragmented in multiple PEAP packets.

All Outer-TLVs MUST NOT have the mandatory bit set. If an Outer-TLV has the mandatory bit set, then the packet MUST be ignored.

PEAPv2 implementations MUST support TLVs, as well as processing of mandatory/optional settings on the TLV.

4.1. TLV Format

TLVs are defined as described below. The fields are transmitted from left to right.



M

0 - Optional TLV
1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

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

0 - Reserved
1 - Reserved
2 - Reserved
3 - Result-TLV - Acknowledged Result
4 - NAK-TLV
5 - Error-Code TLV
6 - Connection-Binding TLV
7 - Vendor-Specific TLV
8 - URI-TLV
9 - EAP-Payload TLV
10 - Intermediate-Result TLV
11 - Reserved
12 - Crypto-Binding TLV
13 - Calling-Station-Id TLV
14 - Called-Station-Id TLV
15 - NAS-Port-Type TLV
16 - Server-Identifier TLV
17 - Identity-Type TLV
18 - Server-Trusted-Root TLV
19 - Request-Action TLV
20 - PKCS#7 TLV

Length

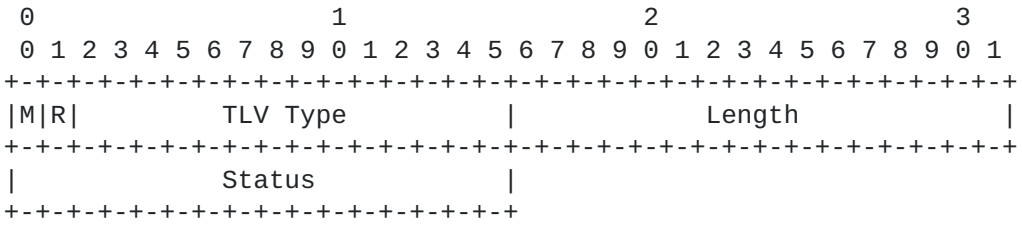
The length of the Value field in octets.

Value

The value of the TLV.

4.2. Result TLV

The Result TLV provides support for acknowledged success and failure messages within PEAPv2. PEAPv2 implementations MUST support this TLV, which cannot be responded to with a NAK TLV. If the Status field does not contain one of the known values, then the peer or EAP server MUST drop the connection. The Result TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

3

Length

2

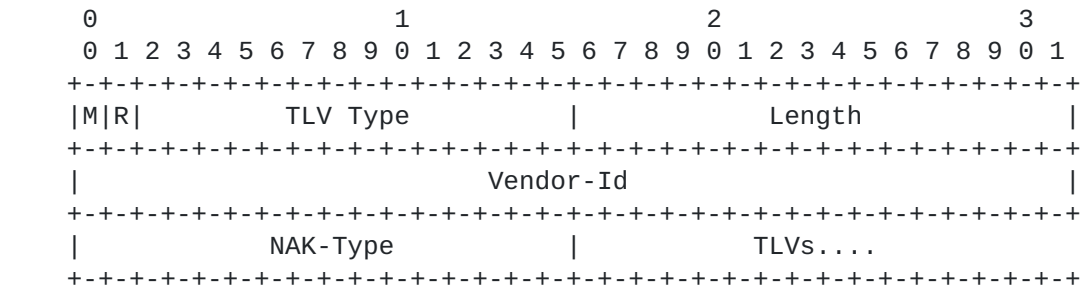
Status

The Status field is two octets. Values include:

1 - Success
2 - Failure

4.3. NAK TLV

The NAK TLV allows a peer to detect TLVs that are not supported by the other peer. A TLV packet can contain 0 or more NAK TLVs. PEAPv2 implementations MUST support the NAK TLV, which cannot be responded to with a NAK TLV. The NAK TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

4

Length

>=6

Vendor-Id

The Vendor-Id field is four octets, and contains the Vendor-Id of the TLV that was not supported. The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order. The Vendor-Id field MUST be zero for TLVs that are not Vendor-Specific TLVs. For Vendor-Specific TLVs, the Vendor-ID MUST be set to the SMI code.

NAK-Type

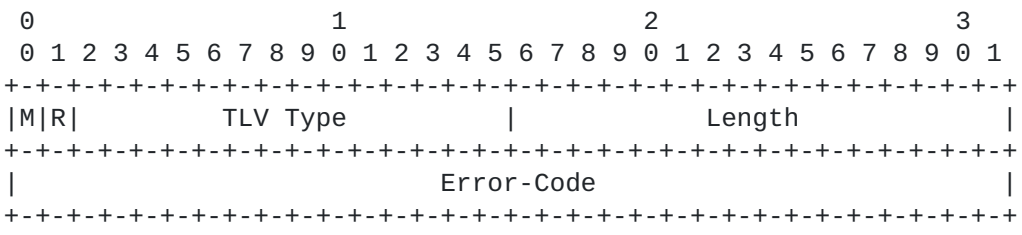
The NAK-Type field is two octets. The field contains the Type of the TLV that was not supported. A TLV of this Type MUST have been included in the previous packet.

TLVs

This field contains a list of TLVs, each of which MUST NOT have the mandatory bit set. These optional TLVs can be used in the future to communicate why the offending TLV was determined to be unsupported.

4.4. Error-Code TLV

The Error-Code TLV allows a PEAPv2 peer or server to indicate errors to the other party. A TLV packet can contain 0 or more Error TLVs. Error-Code TLVs MUST be marked as Mandatory. PEAPv2 implementations MUST support the Error-Code TLV, which cannot be responded to with a NAK TLV. The Error-Code TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

5

Length

4

Error-Code

The Error-Code field is four octets. Error Codes 1-999 represent successful outcomes (informative messages), 1000-1999 represent warnings, and codes 2000-2999 represent fatal errors. If an Error-Code TLV with a fatal error has been sent, then the conversation must be terminated.

Currently defined values for Error-Code include:

2001	Tunnel_Compromise_Error
2002	Unexpected_TLVs_Exchanged

4.5. Crypto-Binding TLV

The Crypto-Binding TLV is used prove that both peers participated in the sequence of authentications (specifically the TLS session and

inner EAP methods that generate keys).

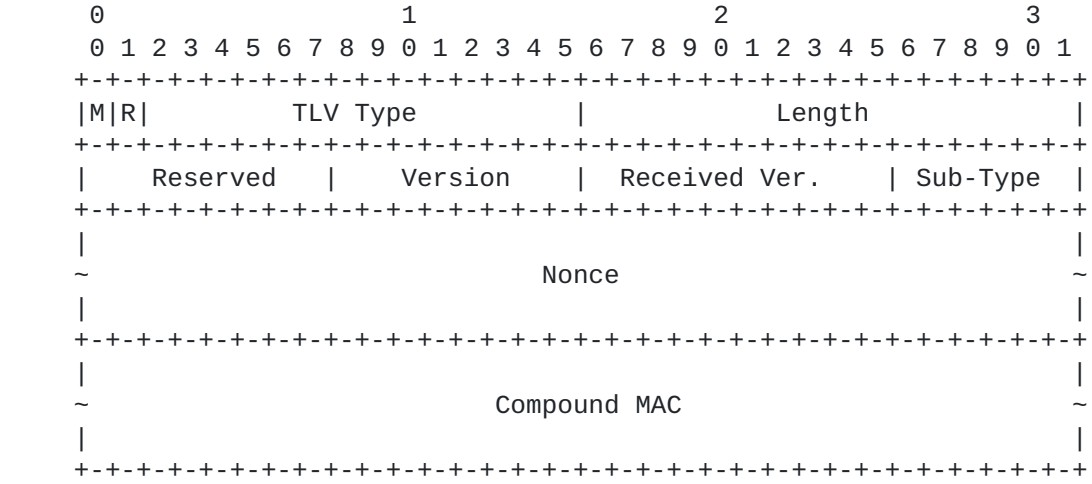
Both the Binding Request (B1) and Binding Response (B2) use the same packet format. However the Sub-Type indicates whether it is B1 or B2.

The Crypto-Binding TLV MUST be used to perform Cryptographic Binding after each successful EAP method in a sequence of EAP methods is complete in PEAPv2 part 2. The Crypto-Binding TLV can also be used during Protected Termination.

The Crypto-Binding TLV must have the version number received during the PEAP version negotiation. The receiver of the Crypto-Binding TLV must verify that the version in the Crypto-Binding TLV matches the version it sent during the PEAP version negotiation. If this check fails then the TLV is invalid.

The receiver of the Crypto-Binding TLV must verify that the subtype is not set to any value other than the ones allowed. If this check fails then the TLV is invalid.

This message format is used for the Binding Request (B1) and also the Binding Response. This uses TLV type CRYPTO_BINDING_TLV. PEAPv2 implementations MUST support this TLV and this TLV cannot be responded to with a NAK TLV. The Crypto-Binding TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

12

Length

56

Reserved

Reserved, set to zero (0)

Version

The Version field is a single octet, which is set to the version of Crypto-Binding TLV. For the Crypto-Binding TLV defined in this specification, it is set to two (2).

Received Version

The Received Version field is a single octet and MUST be set to the PEAP version number received during version negotiation. Note that this field only provides protection against downgrade attacks where a version of PEAP requiring support for this TLV is required on both sides (such as PEAPv2 or a more recent version).

Sub-Type

The Sub-Type field is two octets. Possible values include:

0 - Binding Request
1 - Binding Response

Nonce

The Nonce field is 32 octets. It contains a 256 bit nonce that is temporally unique, used for compound MAC key derivation at each end. This is the S_NONCE for the B1 message and a C_NONCE for the B2 message.

Compound MAC

The Compound MAC field is 20 octets. This can be the Server MAC (B1_MAC) or the Client MAC (B2_MAC). It is computed using the HMAC-SHA1-160 keyed MAC that provides 160 bits of output using the CMK key. The MAC is computed over the buffer created after

concatenating these fields in the following order:

[a] The entire Crypto-Binding TLV attribute with the MAC field zeroed out.

[b] The EAP Type sent by the other party in the first PEAP message.

[c] All the Outer-TLVs from the first PEAP message sent by EAP-server to peer. If a single PEAP message is fragmented into multiple PEAP packets; then the Outer-TLVs in all the fragments of that message MUST be included.

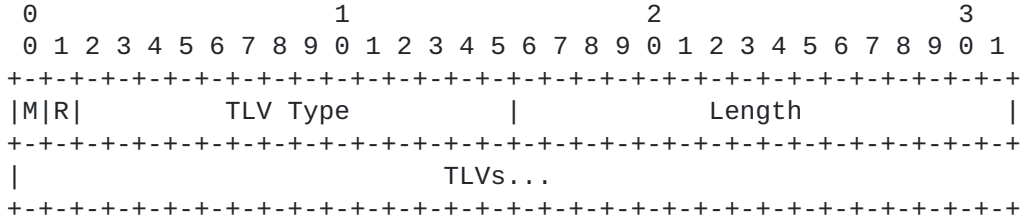
[d] All the Outer-TLVs from the first PEAP message sent by the peer to the EAP server. If a single PEAP message is fragmented into multiple PEAP packets, then the Outer-TLVs in all the fragments of that message MUST be included.

4.6. Connection-Binding TLV

The Connection-Binding TLV allows for connection specific information to be sent by the peer to the AAA server. This TLV should be logged by the EAP or AAA server. The AAA or EAP server should not deny access if there i s a mismatch between the value sent through the AAA protocol and this TLV.

The format of this TLV is defined for the layer that defines the parameters. The format of the value sent by the peer to the EAP server may be different from the format of the corresponding value sent through the AAA protocol. For example, the connection binding TLV may contain the 802.11 MAC Address or SSID.

PEAP implementations MAY support this TLV and this TLV MUST NOT be responded to with a NAK TLV. The Connection-Binding TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

6

Length

>=0

TLVs...

The field contains a list of TLVs, each in the same format defined in Section 4.3, with the optional bit set. These TLVs contain information on the identity of the peer and authenticator (layer 2 or IP addresses); the media used to connect the peer and authenticator (NAS-Port-Type); and/or the service the client is trying to access on the gateway (SSID). The format of these TLVs will be defined in a separate draft.

4.7. Vendor-Specific TLV

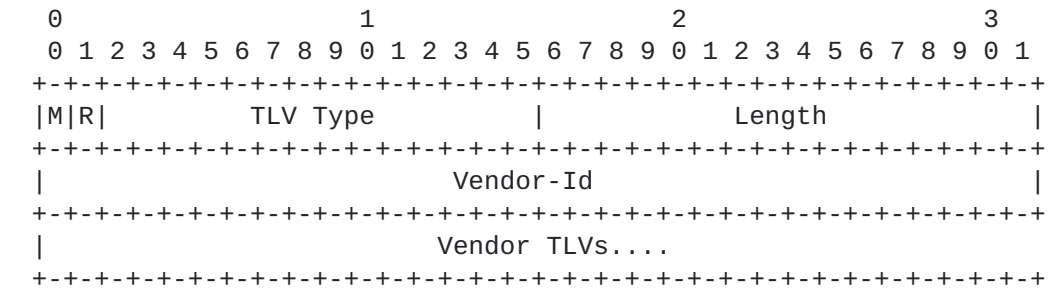
The Vendor-Specific TLV is available to allow vendors to support their own extended attributes not suitable for general usage.

A Vendor-Specific-TLV attribute can contain one or more TLVs, referred to as Vendor TLVs. The TLV-type of the Vendor-TLV will be defined by the vendor. All the Vendor TLVs inside a single Vendor-Specific TLV belong to the same vendor.

PEAPv2 implementations MUST support the Vendor-Specific TLV, and this TLV MUST NOT be responded to with a NAK TLV. PEAPv2 implementations MAY NOT support the Vendor TLVs inside in the Vendor-Specific TLV, and can respond to the Vendor TLVs with a NAK TLV containing the appropriate Vendor-ID and Vendor-TLV type.

Vendor TLVs may be optional or mandatory. Vendor TLVs sent in the protected success and failure packets MUST be marked as optional. If Vendor TLVs sent in protected success/failure packets are marked as Mandatory, then the peer or EAP server MUST drop the connection.

The Vendor-Specific TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

7

Length

>=4

Vendor-Id

The Vendor-Id field is four octets. The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order. The Vendor-Id MUST be zero for TLVs that are not Vendor-Specific TLVs. For Vendor-Specific TLVs, the Vendor-ID MUST be set to the SMI code.

Vendor TLVs

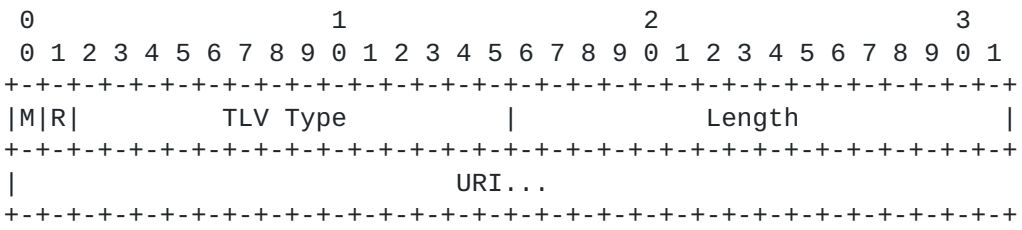
This field is of indefinite length. It contains vendor-specific TLVs, in a format defined by the vendor.

4.8. URI TLV

The URI TLV allows a server to send a URI to the client to refer it to a resource. The TLV contains a URI in the format specified in RFC2396 with UTF-8 encoding. Interpretation of the value of the URI is outside the scope of this document.

If a packet contains multiple URI TLVs, then the client SHOULD select the first TLV it can implement, and ignore the others. If the client

is unable to implement any of the URI TLVs, then it MAY ignore the error. PEAP implementations MAY support this TLV; and this TLV cannot be responded to with a NAK TLV. The URI TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

8

Length

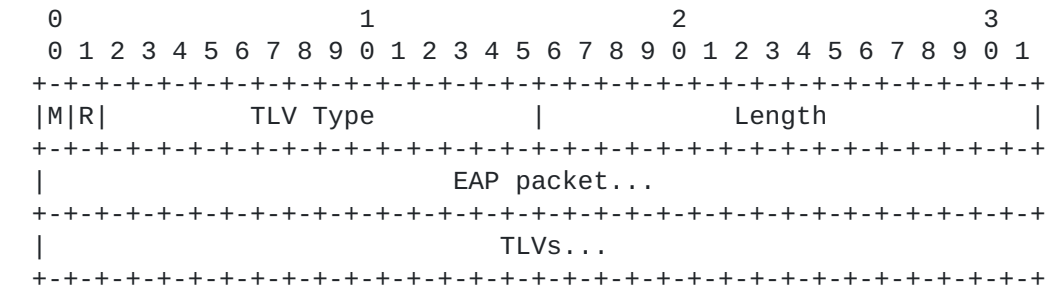
>=0

URI

This field is of indefinite length, and conforms to the format specified in [RFC2396].

4.9. EAP-Payload TLV

To allow piggybacking EAP request and response with other TLVs, the EAP Payload TLV is defined, which includes an encapsulated EAP packet and 0 or more TLVs. PEAPv2 implementations MUST support this TLV, which cannot be responded to with a NAK TLV. The EAP-Payload TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

9

Length

>=0

EAP packet

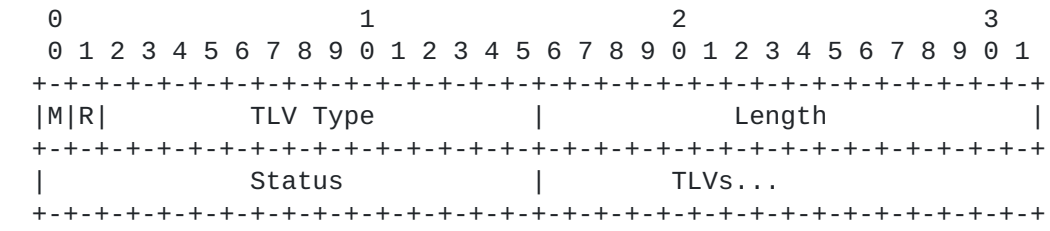
This field contains a complete EAP packet, including the EAP header (Code, Identifier, Length, Type) fields. The length of this field is determined by the Length field of the encapsulated EAP packet.

TLVs...

This (optional) field contains a list of TLVs associated with the EAP packet field. The TLVs utilize the same format described Section 4.3, and MUST NOT have the mandatory bit set. The total length of this field is equal to the Length field of the EAP-Payload-TLV, minus the Length field in the EAP header of the EAP packet field.

4.10. Intermediate Result TLV

The Intermediate Result TLV provides support for acknowledged intermediate Success and Failure messages within EAP. PEAPv2 implementations MUST support this TLV, which cannot be responded to with a NAK TLV. The Intermediate Result TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

10

Length

>=2

Status

The Status field is two octets. Values include:

1 - Success
2 - Failure

TLVs

This (optional) field is of indeterminate length, and contains the TLVs associated with the Intermediate Result TLV, in the same format as described in Section 4.3. The TLVs in this field MUST NOT have the mandatory bit set.

4.11. Reserved TLVs

TLV type 11 is reserved due to use in previous implementations. PEAPv2 implementations MAY NOT support this TLV, which MUST be marked as OPTIONAL. This TLV MUST NOT be responded to with a NAK TLV.

4.12. Calling-Station-ID TLV

This TLV allows a peer to send information to EAP server about the call originator. This TLV MAY be included in the Connection-Binding-

TLV.

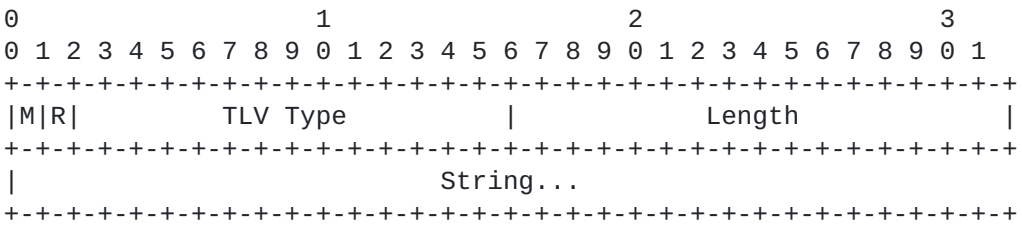
For dial-up, the Called-Station-ID TLV contains the phone number of the peer. For use with IEEE 802.1X, the MAC address of the peer is included, as specified in [RFC3580].

For VPN, this attribute is used to send the IPv4 or IPV6 address of the interface of the peer used to initiate the VPN in ASCII format. Where the Fully Qualified Domain Name (FQDN) of the VPN client is known, it SHOULD be appended, separated from the address with a " " (space). Example: "12.20.2.3 vpnserver.company.com".

As described in [RFC3748] Section 7.15, this TLV SHOULD be logged by the EAP or AAA server, and MAY be used for comparison with information gathered by other means.

However, since the format of this TLV may not match the format of the information gathered by other means, if an EAP server or AAA server supports the capability to deny access based on a mismatch, spurious authentication failures may occur. As a result, implementations SHOULD allow the administrator to disable this check.

PEAP implementations MAY support this TLV and this TLV MUST NOT be responded to with a NAK TLV. The Calling-Station-ID TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

13

Length

>=0

String

The field should be the same as the value of the Calling-Station-ID attribute in [RFC2865].

4.13. Called-Station-ID TLV

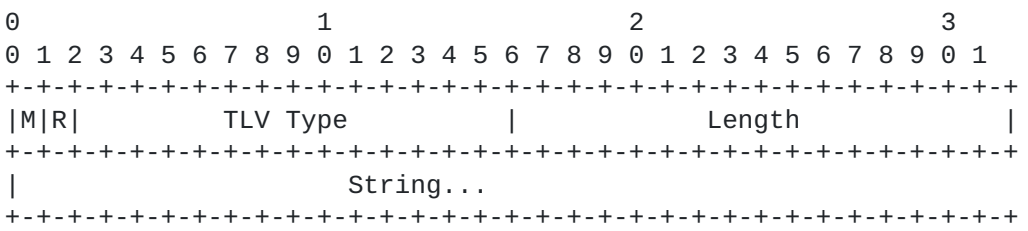
This TLV allows a peer to send information to EAP server about the NAS it called. This TLV MAY be included in the Connection-Binding-TLV.

For dial-up, the Calling-Station-ID TLV contains the phone number called by the peer. For use with IEEE 802.1X, the MAC address of the NAS is included, as specified in [RFC3580].

For VPN, this attribute is used to send the IPv4 or IPv6 address of VPN server in ASCII format. Where the Fully Qualified Domain Name (FQDN) of the VPN server is known, it SHOULD be appended, separated from the address with a " " (space). Example: "12.20.2.3 vpnserver.company.com".

This TLV SHOULD be logged by the EAP or AAA server, and MAY be used for comparison with information gathered by other means. However, since the format of this TLV may not match the format of the information gathered by other means, if an EAP server or AAA server supports the capability to deny access based on a mismatch, spurious authentication failures may occur. As a result, implementations SHOULD allow the administrator to disable this check.

PEAP implementations MAY support this TLV, and this TLV MUST NOT be responded to with a NAK TLV. The Called-Station-ID TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

14

Length

>=0

String

The field should be the same as the value of the Called-Station-ID attribute in [RFC2865].

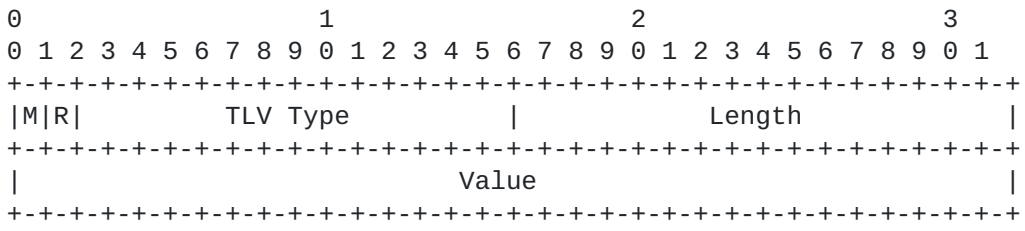
4.14. NAS-Port-Type TLV

This TLV allows a peer to send information to EAP server about the type of physical connection used by the peer to connect to NAS. This TLV MAY be included in the Connection-Binding-TLV.

The value of this field is the same as the value of NAS-Port-Type attribute in [RFC2865].

This TLV SHOULD be logged by the EAP or AAA server and MAY be used for comparison with information gathered by other means. However, since the format of this TLV may not match the format of the information gathered by other means, if an EAP server or AAA server supports the capability to deny access based on a mismatch, spurious authentication failures may occur. As a result, implementations SHOULD allow the administrator to disable this check.

PEAP implementations MAY support this TLV; and this TLV MUST NOT be responded to with a NAK TLV. The NAS-Port-Type TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

15

Length

4

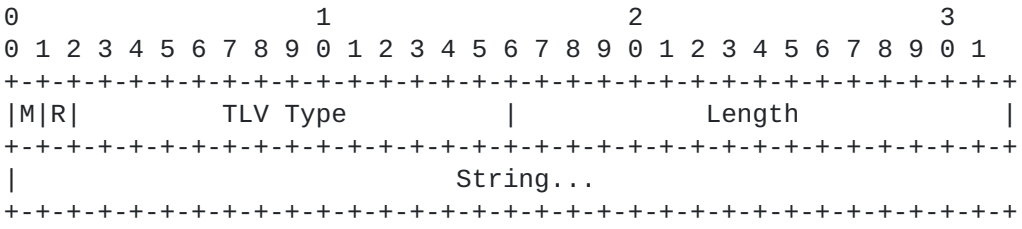
Value

The Value field is four octets. Values are the same as for the NAS-Port-Type attribute defined in [RFC2865].

4.15. Server-Identifier TLV

This TLV allows a EAP-Server to send a hint to the EAP peer to help the EAP peer select the appropriate sessionID for session resumption. The field is a string sent by the EAP server, and the field should be treated as a opaque string by the peer. During a full-tls-handshake, the EAP-peer MAY keep track of this field and the corresponding sessionID, and use it as a hint to select the appropriate sessionID during session resumption.

PEAP implementations MAY support this TLV and this TLV MUST NOT be responded to with a NAK TLV. The Server-Identifier TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

16

Length

>=0

String

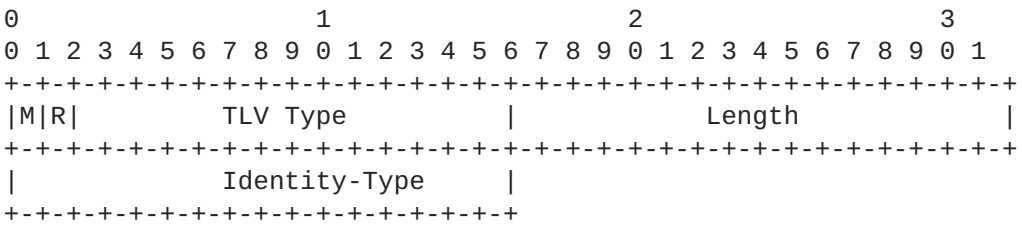
Contains an identifier sent by the EAP server.

4.16. Identity-Type TLV

The Identity-Type TLV allows an EAP-server to send a hint to help the EAP-peer select the right type of identity; for example; user or machine.

PEAPv2 implementations MAY support this TLV, which cannot be responded to with a NAK TLV.

If the Identity-type field does not contain one of the known values or if the EAP peer does not have an identity corresponding to the identity-type, then the peer MUST ignore the value. The Identity-Type TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

17

Length

2

Identity-Type

The Identity-Type field is two octets. Values include:

1 - User
2 - Machine

4.17. Server-Trusted-Root TLV

The Server-Trusted-Root TLV allows the peer to send a request to the EAP server for a trusted root in PKCS#7 format.

The Server-Trusted-Root TLV is always marked as optional, and cannot be responded to with a NAK TLV. PEAPv2 server implementations that claim to support provisioning MUST support Server-Trusted-Root TLV, PKCS#7 TLV, and the PKCS#7-Server-Certificate-Root credential format defined in this TLV. PEAPv2 peer implementations MAY NOT support this TLV.

The Server-Trusted-Root TLV can only be sent as an inner TLV (inside PEAP part 2 conversation), in both server unauthenticated tunnel provisioning mode, and the regular authentication process.

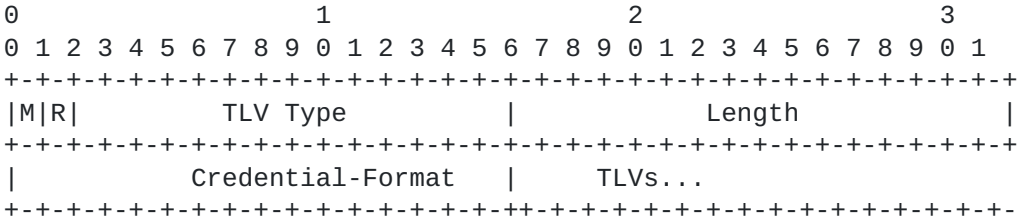
The peer MUST NOT request, or accept the trusted root sent inside the

Server-Root credential TLV by EAP-server until it has completed authentication of EAP server, and validated the Crypto-Binding TLV. The peer may receive a trusted root, but is not required to use the trusted root received from the EAP server.

If the EAP server sets credential-format to PKCS#7-Server-Certificate-Root, then the Server-Trusted-Root TLV MUST contain the root of the certificate chain of the certificate issued to the EAP server packages in a PKCS#7 TLV. If the Server certificate is a self-signed certificate, then the root is the self-signed certificate. In this case, the EAP server does not have to sign the certificate inside the PCKS#7 TLV since it does not necessarily have to private key for it.

If the Server-Trusted-Root TLV credential format does not contain one of the known values, then the EAP-server MUST ignore the value.

The Server-Trusted-Root TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

18

Length

>=2

Credential-Format

The Credential-Format field is two octets. Values include:

1 - PKCS#7-Server-Certificate-Root.

TLVs

This field is of indefinite length. It contains TLVs associated with the certificate-request.

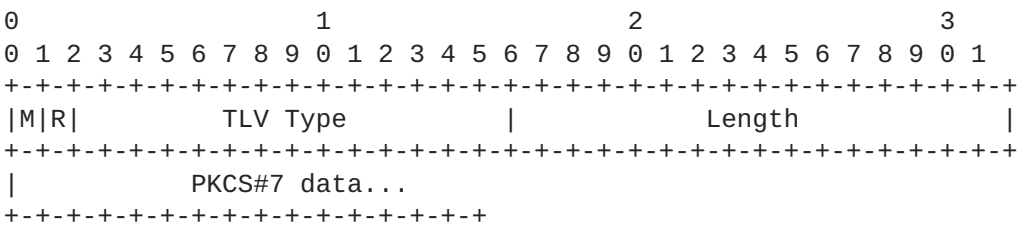
4.18. PKCS#7 TLV

The PKCS#7 TLV contains the PKCS #7 wrapped X.509 certificate. This field contains a certificate or certificate chain in PKCS#7 [RFC2315] format requested by the peer.

The PKCS#7 TLV is always marked as optional, which cannot be responded to with a NAK TLV. PEAPv2 server implementations that claim to support provisioning MUST support this TLV. PEAPv2 peer implementations MAY NOT support this TLV.

If the PKCS#7 TLV contains a certificate or certificate chain that is not acceptable to the peer, then peer MUST ignore the value.

The PKCS#7 TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

20

Length

>=0

PKCS#7 Data

PKCS #7 wrapped X.509 certificate. This field contains a

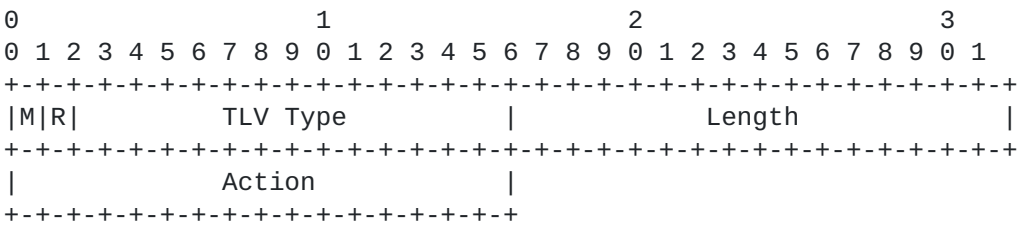
certificate or certificate chain in PKCS#7 [RFC2315] format.

4.19. Request-Action TLV

The Request-Action TLV MAY be sent by the peer along with acknowledged failure. It allows the peer to request the EAP server to negotiate EAP methods or process TLVs specified in the failure packet. The server MAY ignore this TLV.

PEAPv2 implementations MUST support this TLV, which cannot be responded to with a NAK TLV.

The Request-Action TLV is defined as follows:



M

1 - Mandatory TLV 0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

19

Length

2

Action

The Action field is two octets. Values include:

1 - Process-TLV
2 - Negotiate-EAP

4.20. TLV Rules

To save round trips, multiple TLVs can be sent in the single PEAPv2 packet. However, multiple EAP Payload TLVs within one single PEAPv2 packet is not supported in this version and MUST NOT be sent. If the peer or EAP server receives multiple EAP Payload TLVs, then it MUST drop the connection.

The following table defines the meaning of the table entries in the sections below:

0	This TLV MUST NOT be present in the packet.
0+	Zero or more instances of this TLV MAY be present in packet.
0-1	Zero or one instance of this TLV MAY be present in packet.
1	Exactly one instance of this TLV MUST be present in packet.

4.20.1. Outer TLVs

The following table provides a guide to which TLVs may be included in the PEAPv2 packet outside the TLS channel, which kind of packets, and in what quantity:

Request	Response	Success	Failure	TLV in unencrypted-TLVs field
0-1	0	0	0	Server-Identifier TLV
0+	0+	0	0	Vendor-Specific TLV

Outer-TLVs MUST be marked as optional. Vendor-TLVs inside Vendor-Specific TLV MUST be marked as optional when included in Outer TLVs. Outer-TLVs MUST NOT be included in messages after the first two PEAPv2 messages sent by peer and EAP-server respectively. That is the first EAP server to peer message and first peer to EAP server message. If the message is fragmented, the whole set of messages is counted as one message. If Outer-TLVs are included in messages after the first two PEAPv2 messages, they MUST be ignored.

4.20.2. Inner TLVs

The following table provides a guide to which Inner TLVs may be encapsulated in TLS in PEAPv2 Part 2, in which kind of packets, and in what quantity:

Request	Response	Success	Failure	Inner-TLVs
0-1	0-1	0-1	0-1	Intermediate Result
0-1	0-1	0	0	EAP Payload
0-1	0-1	1	1	Result
0-1	0-1	1	1	Crypto-Binding
0+	0+	0+	0+	Error
0+	0+	0	0	NAK
0-1	0-1	0-1	0-1	Connection-Binding
0+	0+	0+	0+	Vendor-Specific
0+	0	0+	0-1	URI
0+	0	0	0	Identity-Type
0+	0+	0+	0+	Server-Trusted-Root
0	0-1	0	0-1	Request-Action

Vendor TLVs (included in Vendor-Specific TLVs) sent in the protected success and failure packets MUST be marked as optional. If Vendor TLVs sent in protected success/failure packets are marked as Mandatory, then the peer or EAP server MUST drop the connection.

The following defines the meaning of packet type in the table above:

Packet type	Description
Request	- TLV request packet sent by the EAP server to the peer.
Response	- TLV packet sent by the peer to the EAP server.
Success	- TLV packet sent by the peer or EAP server as a protected success
Failure	- TLV packet sent by the peer or EAP server as a protected failure.

4.20.3. EAP-Payload TLV

The EAP-Payload TLV can contain other TLVs. The table below defines which TLVs can be contained inside the EAP-Payload TLV and how many such TLVs can be included.

Request	Response	TLV
0+	0+	Vendor-Specific
0+	0+	Identity-Type

Vendor TLVs inside Vendor-Specific TLV MUST be marked as optional when included in EAP-Payload TLV.

4.20.4. Connection-Binding TLV

The Connection-Binding TLV can contain other TLVs. The table below defines which TLVs can be contained inside the Connecting-Binding TLV and how many such TLVs can be included.

Request	Response	TLV
0-1	0	Calling-Station-ID
0-1	0	Called-Station-ID
0-1	0	NAS-port-type
0+	0+	Vendor-Specific

Vendor TLVs inside the Vendor-Specific TLV MUST be marked as optional when included in Connection-Binding TLV.

4.20.5. Server-Trusted-Root TLV

The Server-Trusted-Root TLV can contain other TLVs. The table below defines which TLVs can be contained inside the Server-Trusted-Root TLV and how many such TLVs can be included.

Request	Response	TLV
0-1	0	PKCS#7 TLV

5. Security Considerations

5.1. Authentication and integrity protection

PEAPv2 provides a server authenticated, encrypted and integrity protected tunnel. All data within the tunnel has these properties. Data outside the tunnel such as EAP Success and Failure, Outer-TLVs, authentication methods negotiated outside of PEAPv2 and the PEAPv2 headers themselves (including the EAP-Type in the header) are not protected by this tunnel.

In addition, the Crypto-Binding TLV can reveal man-in-the-middle attack described in section 6.8. Hence, the server should not reveal any sensitive data to the client until after the Crypto-Binding TLV has been properly verified.

In order to detect modification of Outer-TLVs, the first two Outer-TLVs messages sent by both peer and EAP-server are included in the calculation of the Crypto-Binding TLV. Outer-TLVs SHOULD NOT be included in other PEAP packets since there is no mechanism to detect modification.

In order to detect modification of EAP-Type sent in the clear (EAP-Type should be set to PEAP), the EAP-Type sent in the first two messages by both peer and EAP-server is included in the calculation of Crypto-Binding TLV. The EAP Type in the clear could be modified in other PEAP packets and will likely result in failure, hence it is not included in the Crypto-Binding calculation.

5.2. Method Negotiation

If the peer does not support PEAPv2, or does not wish to utilize PEAPv2 authentication, it MUST respond to the initial EAP-Request/PEAP-Start with a NAK, suggesting an alternate authentication method. Since the NAK is sent in cleartext with no integrity protection or authentication, it is subject to spoofing. Inauthentic NAK packets can be used to trick the peer and authenticator into "negotiating down" to a weaker form of authentication, such as EAP-MD5 (which only provides one way authentication and does not derive a key).

Since a subsequent protected EAP conversation can take place within the TLS session, selection of PEAPv2 as an authentication method does not limit the potential secondary authentication methods. As a result, the only legitimate reason for a peer to NAK PEAPv2 as an authentication method is that it does not support it. Where the additional security of PEAPv2 is required, server implementations SHOULD respond to a NAK with an EAP-Failure, terminating the authentication conversation.

Since method negotiation outside of PEAP is not protected, if the peer is configured to allow PEAP and other EAP methods at the same time, the negotiation is subject to downgrade attacks. Since method negotiation outside of PEAP is not protected, if the peer is configured to allow PEAPv2 and previous PEAP versions at the same time, the negotiation is subject to negotiation downgrade attacks. However, peers configured to allow PEAPv2 and later PEAP versions may not be subject to downgrade negotiation attack since the highest version supported by both peers is checked within the protected tunnel.

If peer implementations select incorrect methods or credentials with EAP servers, then attacks are possible on the credentials. Hence, a PEAPv2 peer implementation should preferably be configured with a set of credentials and methods that may be used with a specific PEAPv2 server. The peer implementation may be configured to use different methods and/or credentials based on the PEAPv2 server.

5.3. TLS Session Cache Handling

In cases where a TLS session has been successfully resumed, in some circumstances, it is possible for the EAP server to skip the PEAPv2 Part 2 conversation, and successfully conclude the conversation with a protected termination.

PEAPv2 "fast reconnect" is desirable in applications such as wireless roaming, since it minimizes interruptions in connectivity. It is

also desirable when the "inner" EAP mechanism used is such that it requires user interaction. The user should not be required to re-authenticate herself, using biometrics, token cards or similar, every time the radio connectivity is handed over between access points in wireless environments.

However, there are issues that need to be understood in order to avoid introducing security vulnerabilities.

Since PEAPv2 Part 1 may not provide client authentication, establishment of a TLS session (and an entry in the TLS session cache) does not by itself provide an indication of the peer's authenticity.

Some PEAPv2 implementations may not be capable of removing TLS session cache entries established in PEAPv2 Part 1 after an unsuccessful PEAPv2 Part 2 authentication. In such implementations, the existence of a TLS session cache entry provides no indication that the peer has previously been authenticated. As a result, implementations that do not remove TLS session cache entries after a failed PEAPv2 Part 2 authentication or failed protected termination MUST use other means than successful TLS resumption as the indicator of whether the client is authenticated or not. The implementation MUST determine that the client is authenticated only after the completion of protected termination. Failing to do this would enable a peer to gain access by completing PEAPv2 Part 1, tearing down the connection, re-connecting and resuming PEAPv2 Part 1, thereby proving herself authenticated. Thus, TLS resumption MUST only be enabled if the implementation supports TLS session cache removal. If an EAP server implementing PEAPv2 removes TLS session cache entries of peers failing PEAPv2 Part 2 authentication, then it MAY skip the PEAPv2 Part 2 conversation entirely after a successful session resumption, successfully terminating the PEAPv2 conversation as described in Section 2.4.

5.4. Certificate Revocation

Since the EAP server usually has network connectivity during the EAP conversation, the server is capable of following a certificate chain or verifying whether the peer's certificate has been revoked. In contrast, the peer may or may not have network connectivity, and thus while it can validate the EAP server's certificate based on a pre-configured set of CAs, it may not be able to follow a certificate chain or verify whether the EAP server's certificate has been revoked.

In the case where the peer is initiating a voluntary Layer 2 channel using PPTP or L2TP, the peer will typically already have network

connectivity established at the time of channel initiation. As a result, during the EAP conversation it is capable of checking for certificate revocation.

As part of the TLS negotiation, the server presents a certificate to the peer. The peer SHOULD verify the validity of the EAP server certificate, and SHOULD also examine the EAP server name presented in the certificate, in order to determine whether the EAP server can be trusted. Please note that in the case where the EAP authentication is remoted, the EAP server will not reside on the same machine as the authenticator, and therefore the name in the EAP server's certificate cannot be expected to match that of the intended destination. In this case, a more appropriate test might be whether the EAP server's certificate is signed by a CA controlling the intended destination and whether the EAP server exists within a target sub-domain.

In the case where the peer is attempting to obtain network access, it will not have network connectivity. The TLS Extensions [RFC3546] support piggybacking of an Online Certificate Status Protocol (OCSP) response within TLS, therefore can be utilized by the peer in order to verify the validity of server certificate. However, since not all TLS implementations implement the TLS extensions, it may be necessary for the peer to wait to check for certificate revocation until after network access has been obtained. In this case, the peer SHOULD conduct the certificate status check immediately upon going online and SHOULD NOT send data until it has received a positive response to the status request. If the server certificate is found to be invalid as per client policy, then the peer SHOULD disconnect.

If the client has a policy to require checking certificate revocation and it cannot obtain revocation information then it may need to disallow the use of all or some of the inner methods since some methods may reveal some sensitive information.

5.5. Separation of the EAP Server and the Authenticator

As a result of a complete PEAPv2 Part 1 and Part 2 conversation, the EAP endpoints will mutually authenticate, and derive a session key for subsequent use in link layer security. Since the peer and EAP client reside on the same machine, it is necessary for the EAP client module to pass the session key to the link layer encryption module.

The situation may be more complex on the Authenticator, which may or may not reside on the same machine as the EAP server. In the case where the EAP server and the Authenticator reside on different machines, there are several implications for security. Firstly, the mutual authentication defined in PEAPv2 will occur between the peer and the EAP server, not between the peer and the authenticator. This

means that as a result of the PEAP conversation, it is not possible for the peer to validate the identity of the NAS or channel server that it is speaking to.

The second issue is that the session key negotiated between the peer and EAP server will need to be transmitted to the authenticator. Therefore a secure mechanism needs to be provided to transmit the session key from the EAP server to the authenticator or channel server that needs to use the key. The specification of this transit mechanism is outside the scope of this document.

5.6. Separation of PEAPv2 Part 1 and 2 Servers

The EAP server involved in PEAPv2 Part 2 need not necessarily be the same as the EAP server involved in PEAPv2 Part 1. For example, a local authentication server or proxy might serve as the endpoint for the Part 1 conversation, establishing the TLS channel. Subsequently, once the EAP-Response/Identity has been received within the TLS channel, it can be decrypted and forwarded in cleartext to the destination realm EAP server. The rest of the conversation will therefore occur between the destination realm EAP server and the peer, with the local authentication server or proxy acting as an encrypting/decrypting gateway. This permits a non-TLS capable EAP server to participate in the PEAPv2 conversation.

Note however that such an approach introduces security vulnerabilities. Since the EAP Response/Identity is sent in the clear between the proxy and the EAP server, this enables an attacker to snoop the user's identity. It also enables a remote environments, which may be public hot spots or Internet coffee shops, to gain knowledge of the identity of their users. Since one of the potential benefits of PEAPv2 is identity protection, this is undesirable.

If the EAP method negotiated during PEAPv2 Part 2 does not support mutual authentication, then if the Part 2 conversation is proxied to another destination, the PEAPv2 peer will not have the opportunity to verify the secondary EAP server's identity. Only the initial EAP server's identity will have been verified as part of TLS session establishment.

Similarly, if the EAP method negotiated during PEAPv2 Part 2 is vulnerable to dictionary attack, then an attacker capturing the cleartext exchange will be able to mount an offline dictionary attack on the password.

Finally, when a Part 2 conversation is terminated at a different location than the Part 1 conversation, the Part 2 destination is unaware that the EAP client has negotiated PEAPv2. As a result, it is

unable to enforce policies requiring PEAP. Since some EAP methods require PEAPv2 in order to generate keys or lessen security vulnerabilities, where such methods are in use, such a configuration may be unacceptable.

In summary, PEAPv2 encrypting/decrypting gateway configurations are vulnerable to attack and SHOULD NOT be used. Instead, the entire PEAPv2 connection SHOULD be proxied to the final destination, and the subsequently derived master session keys need to be transmitted back. T his provides end to end protection of PEAPv2. The specification of this transit mechanism is outside the scope of this document, but mechanisms similar to [RFC2548] can be used. These steps protect the client from revealing her identity to the remote environment.

In order to find the proper PEAP destination, the EAP client SHOULD place a Network Access Identifier (NAI) conforming to [RFC2486] in the Identity Response.

There may be cases where a natural trust relationship exists between the (foreign) authentication server and final EAP server, such as on a campus or between two offices within the same company, where there is no danger in revealing the identity of the station to the authentication server. In these cases, a proxy solution without end to end protection of PEAPv2 MAY be used. If RADIUS is used to communicate between gateway and EAP server, then the PEAPv2 encrypting/decrypting gateway SHOULD provide support for IPsec protection of RADIUS in order to provide confidentiality for the portion of the conversation between the gateway and the EAP server, as described in [RFC3579].

5.7. Identity Verification

Since the TLS session has not yet been negotiated, the initial Identity request/response occurs in the clear without integrity protection or authentication. It is therefore subject to snooping and packet modification.

In configurations where all users are required to authenticate with PEAPv2 and the first portion of the PEAPv2 conversation is terminated at a local backend authentication server, without routing by proxies, the initial cleartext Identity Request/Response exchange is not needed in order to determine the required authentication method(s) or route the authentication conversation to its destination. As a result, the initial Identity and Request/Response exchange MAY NOT be present, and a subsequent Identity Request/Response exchange MAY occur after the TLS session is established.

If the initial cleartext Identity Request/Response has been tampered

with, after the TLS session is established, it is conceivable that the EAP Server will discover that it cannot verify the peer's claim of identity. For example, the peer's userID may not be valid or may not be within a realm handled by the EAP server. Rather than attempting to proxy the authentication to the server within the correct realm, the EAP server SHOULD terminate the conversation.

The PEAPv2 peer can present the server with multiple identities. This includes the claim of identity within the initial EAP-Response/Identity (MyID) packet, which is typically used to route the EAP conversation to the appropriate home backend authentication server. There may also be subsequent EAP-Response/Identity packets sent by the peer once the TLS channel has been established.

Note that since the PEAPv2 peer may not present a certificate, it is not always possible to check the initial EAP-Response/Identity against the identity presented in the certificate, as is done in [RFC2716].

Moreover, it cannot be assumed that the peer identities presented within multiple EAP-Response/Identity packets will be the same. For example, the initial EAP-Response/Identity might correspond to a machine identity, while subsequent identities might be those of the user. Thus, PEAPv2 implementations SHOULD NOT abort the authentication just because the identities do not match. However, since the initial EAP-Response/Identity will determine the EAP server handling the authentication, if this or any other identity is inappropriate for use with the destination EAP server, there is no alternative but to terminate the PEAPv2 conversation.

The protected identity or identities presented by the peer within PEAPv2 Part 2 may not be identical to the cleartext identity presented in PEAPv2 Part 1, for legitimate reasons. In order to shield the userID from snooping, the cleartext Identity may only provide enough information to enable routing of the authentication request to the correct realm. For example, the peer may initially claim the identity of "nouser@bigco.com" in order to route the authentication request to the bigco.com EAP server. Subsequently, once the TLS session has been negotiated, in PEAPv2 Part 2, the peer may claim the identity of "fred@bigco.com". Thus, PEAPv2 can provide protection for the user's identity, though not necessarily the destination realm, unless the PEAPv2 Part 1 conversation terminates at the local authentication server.

As a result, PEAPv2 implementations SHOULD NOT attempt to compare the Identities claimed with Parts 1 and 2 of the PEAPv2 conversation. Similarly, if multiple Identities are claimed within PEAPv2 Part 2, these SHOULD NOT be compared. An EAP conversation may involve more

than one EAP authentication method, and the identities claimed for each of these authentications could be different (e.g. a machine authentication, followed by a user authentication).

5.8. Man-in-the-Middle Attack Protection

TLS protection can address a number of weaknesses in the EAP method; as well as EAP protocol weaknesses listed in the abstract and introduction sections in this document.

Hence, the recommended solution is to always deploy authentication methods with protection of PEAPv2.

if a deployment chooses to allow a EAP method protected by PEAPv2 without protection of PEAPv2 or IPsec at the same time, then this opens up a possibility of a man-in-the-middle attack.

A man-in-the-middle can spoof the client to authenticate to it instead of the real EAP server; and forward the authentication to the real server over a protected tunnel. Since the attacker has access to the keys derived from the tunnel, it can gain access to the network.

PEAP version 2 prevents this attack by using the keys generated by the inner EAP method in the crypto-binding exchange described in protected termination section. This attack is not prevented if the inner EAP method does not generate keys or if the keys generated by the inner EAP method can be compromised. Hence, in cases where the inner EAP method does not generate keys, the recommended solution is to always deploy authentication methods protected by PEAPv2.

Alternatively, the attack can also be thwarted if the inner EAP method can signal to the peer that the packets are being sent within the tunnel. In most cases this may require modification to the inner EAP method. In order to allow for these implementations, PEAPv2 implementations should inform inner EAP methods that the EAP method is being protected by a PEAPv2 tunnel.

Since all sequence negotiations and exchanges are protected by TLS channel, they are immune to snooping and MITM attacks with the use of Crypto-Binding TLV. To make sure the same parties are involved tunnel establishment and previous inner method, before engaging the next method to sent more sensitive information, both peer and server MUST use the Crypto-Binding TLV between methods to check the tunnel integrity. If the Crypto-Binding TLV failed validation, they SHOULD stop the sequence and terminate the tunnel connection, to prevent more sensitive information being sent in subsequent methods.

5.9. Cleartext Forgeries

As described in [RFC3748], EAP Success and Failure packets are not authenticated, so that they may be forged by an attacker without fear of detection. Forged EAP Failure packets can be used to convince an EAP peer to disconnect. Forged EAP Success and Failure packets may be used to convince a peer to disconnect; or convince a peer to access the network even before authentication is complete, resulting in denial of service for the peer.

By supporting encrypted, authenticated and integrity protected success/failure indications, PEAPv2 provides protection against these attacks.

Once the peer responds with the first PEAP packet; and the EAP server receives the first PEAPv2 packet from the peer, both MUST silently discard all clear text EAP messages unless both the PEAPv2 peer and server have indicated success or failure or error using a protected error or protected termination mechanism. The success/failure decisions sent by a protected mechanism indicate the final decision of the EAP authentication conversation. After success/failure has been indicated by a protected mechanism, the PEAPv2 client can process unprotected EAP success and EAP failure message; however MUST ignore any unprotected EAP success or failure messages where the decision does not match the decision of the protected mechanism.

After a Fatal alert is received or after protected termination is complete, the peer or EAP server should accept clear text EAP messages. If the PEAPv2 tunnel is nested inside another tunnel, then the clear text EAP messages should only be accepted after protected termination of outer tunnels.

[RFC3748] states that an EAP Success or EAP Failure packet terminates the EAP conversation, so that no response is possible. Since EAP Success and EAP Failure packets are not retransmitted, if the final packet is lost, then authentication will fail. As a result, where packet loss is expected to be non-negligible, unacknowledged success/failure indications lack robustness.

As a result, a EAP server SHOULD send a clear text EAP success or EAP-failure packet after the protected success or failure packet or TLS alert. The peer MUST NOT require the clear text EAP Success or EAP Failure if it has received the protected success or failure or TLS alert. For more details, refer to [RFC3748] Section 4.2.

5.10. TLS Ciphersuites

Anonymous ciphersuites are vulnerable to man-in-the-middle attacks, and SHOULD NOT be used with PEAPv2, unless the EAP methods inside PEAPv2 can address the man-in-the-middle attack or unless the man-in-the-middle attack can be addressed by mechanisms external to PEAPv2.

5.11. Denial of Service Attacks

Denial of service attacks are possible if the attacker can insert or modify packets in the authentication channel. The attacker can modify unprotected fields in the PEAP packet such as the EAP protocol or PEAP version number. This can result in a denial of service attack. It is also possible for the attacker to modify protected fields in a packet to cause decode errors resulting in a denial of service. In these ways the attacker can prevent access for peers connecting to the network.

Denial of service attacks with multiplier impacts are more interesting than the ones above. It is possible to multiply the impact by creating a large number of TLS sessions with the EAP server.

5.12. Server Unauthenticated Tunnel Provisioning Mode

This section describes the rationale and security risks behind server unauthenticated tunnel provisioning mode. Server unauthenticated tunnel provisioning mode results in potential security vulnerabilities. Hence, this mode is optional in PEAPv2 implementations.

In order to achieve strong mutual authentication, it is best to use an out of band mechanism to pre-provision the device with strong symmetric or asymmetric keys. In addition, if the device is not physically secure (mobile or devices at public places), then it is important to ensure that the device has secure storage.

Server unauthenticated tunnel provisioning mode is not recommended for use in devices which already support secure provisioning and secure credential storage capabilities, since the security vulnerabilities will outweight the benefits.

If the provisioned credential is a shared key or asymmetric key issued to the peer, then the credential should only be issued to devices that can protect the provisioned credentials using secure storage, or use physical security.

If the credentials are not protected, the attacker can compromise the

provisioned credentials, and use it to get access to the network. Mobile light weight devices are typically not physically secure. Another concern is that credentials provisioned to a light weight mobile device that does not use secure storage could be transferred to a general operating system and used to get access to the network.

If the provisioned credential is a certificate trusted root of the EAP server, this is public information and hence not susceptible to the same attacks as a shared key or asymmetric key.

In server unauthenticated tunnel provisioning mode, an attacker may terminate the tunnel instead of the real server. The attacker can be detected after the Crypto-Binding TLV is exchanged and validated. However, the EAP packets exchanged inside the tunnel until Crypto-Binding TLV is validated are available in unencrypted form to the attacker. It is difficult to completely negate the security risk unless the EAP methods inside the tunnel are secure; or unless physical wire security is assumed.

The standard credential request/response capability is designed to be independent of the server unauthenticated tunnel provisioning mode, and can be used in regular authentication mode to provision other credentials to the peer that can be used for authentication to the network, or for potentially authentications to other services.

The security risks vary depending on the type of credential exchanged, the scope of use of the credential, and the implementation of the device.

These are a few guidelines to reduce the security risk:

[1] Minimize the use of this mode only during initial authentication to the network to reduce the risk of attack.

[2] If the password based EAP method used in provisioned mode is susceptible to dictionary attacks, then the implementation should support deployment of sound password password policies e.g. capability to enforce strong password policies and support rotation of passwords.

[3] Disable this mode by default and require users to initiate provisioning mode explicitly rather than being prompted during initiation of regular authentication process.

[4] Provide appropriate policy capabilities to allow administrators to lockdown the device and prevent regular users from enabling the mode.

[5] Ensure that the EAP methods used support mutual authentication.

[6] Ensure that the EAP methods used generate keys of sufficient strength to prevent compounding binding from being compromised.

[7] Minimize the information disclosed to the EAP server.

Notes:

[a] The attacker may try to crack the password in the time required for authentication to complete.

[b] The attacker may start the conversation, capture the hash, and launch a offline dictionary attack.

5.13. Security Claims

Intended use:	Wireless or Wired networks, and over the Internet, where physical security cannot be assumed.
Auth. mechanism:	Use arbitrary EAP and TLS authentication mechanisms for authentication of the client and server.
Ciphersuite negotiation:	Yes.
Mutual authentication:	Yes. Depends on the type of EAP method used within the tunnel and the type of authentication used within TLS.
Integrity protection:	Yes
Replay protection:	Yes
Confidentiality:	Yes
Key derivation:	Yes
Key strength:	Variable
Dictionary attack prot:	Not susceptible.
Fast reconnect:	Yes
Crypt. binding:	Yes.
Acknowledged S/F:	Yes
Session independence:	Yes.
Fragmentation:	Yes
State Synchronization:	Yes [80211Req]

The PEAPv2 protocol is unconditionally compliant with the requirements for WLAN authentication mechanisms, as specified in [80211Req].

PEAPv2 derives keys by combining keys from TLS and the inner EAP methods. It should be noted that the use of TLS ciphersuites with a particular key lengths does not guarantee that the key strength of the keys will be equivalent to the length. The key exchange

mechanisms (eg. RSA or Diffie-Hellman) used must provide sufficient security or they will be the weakest link. For example RSA key sizes with a modulus of 1024 bits provides less than 128 bits of security, this may provide sufficient key strength for some applications and not for others. See [PKLENGTH] for a detailed analysis of determining the public key strengths used to exchange symmetric keys.

6. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the PEAPv2 protocol, in accordance with BCP 26, [RFC2434].

The following name spaces in PEAPv2 require registration: TLV-Types, the Identity-Type field in the Identity-Type TLV, the Credential-Type field of the Server-Trusted-Root TLV, and the Action field of the Request-Action TLV. TLV types within Vendor-Specific TLVs do not require registration.

6.1. Definition of Terms

The following terms are used here with the meanings defined in BCP 26: "name space", "assigned value", "registration".

The following policies are used here with the meanings defined in BCP 26: "Private Use", "First Come First Served", "Expert Review", "Specification Required", "IETF Consensus", "Standards Action".

6.2. Recommended Registration Policies

For "Designated Expert with Specification Required", the request is posted to the EAP WG mailing list (or, if it has been disbanded, a successor designated by the Area Director) for comment and review, and MUST include a pointer to a public specification. Before a period of 30 days has passed, the Designated Expert will either approve or deny the registration request and publish a notice of the decision to the EAP WG mailing list or its successor. A denial notice must be justified by an explanation and, in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable.

TLV Types may assume a value between 0 and 16383 of which 0-20 have been allocated. Additional TLV type codes may be allocated following Designated Expert with Specification Required [RFC2434].

The Identity-Type field may assume a value between 0 and 65535, of which 0-2 have been allocated. Additional Identity-Type values may be allocated following Designated Expert with Specification Required

[RFC2434].

The Credential-Type field may assume a value between 0 and 65535, of which 0-1 have already been allocated. Additional Credential-Type values may be allocated following Designated Expert with Specification Required [RFC2434].

The Action field may assume a value between 0 and 65535, of which 0-2 have already been allocated. Additional Action values may be allocated following Designated Expert with Specification Required [RFC2434].

7. References

7.1. Normative references

[RFC1321] Rivest, R. and S. Dusse, "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, November 1998.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2486] Aboba, B. and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[80211Req] Stanley, D., Walker, J. and B. Aboba, "EAP Method Requirements for Wireless LANs", draft-walker-ieee802-req-04.txt, Internet draft (work in progress), August 2004.

7.2. Informative references

[RFC1968] Meyer, G., "The PPP Encryption Protocol (ECP)", RFC 1968, June 1996.

[RFC1990] Sklower, K., Lloyd, B., McGregor, G., Carr, D. and T. Coradetti, "The PPP Multilink Protocol (MP)", RFC 1990, August 1996.

[RFC2419] Sklower, K. and G. Meyer, "The PPP DES Encryption Protocol, Version 2 (DESE-bis)", RFC 2419, September 1998.

[RFC2420] Hummert, K., "The PPP Triple-DES Encryption Protocol (3DESE)", RFC 2420, September 1998.

[RFC2548] Zorn, G., "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, March 1999.

[RFC2865] Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC2716] Aboba, B. and D. Simon, "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999.

[RFC3078] Pall, G. and G. Zorn, "Microsoft Point-to-Point Encryption (MPPE) Protocol", RFC 3078, March 2001.

[RFC3079] Zorn, G., "Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE)", RFC 3079, March 2001.

[RFC3546] Blake-Wilson, S., et al. "TLS Extensions", RFC 3546, June 2003.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS Support for EAP", RFC 3579, September 2003.

[RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G. and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, September 2003.

[RFC3766] H. Orman and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", RFC 3766, April 2004.

[FIPSDES] National Bureau of Standards, "Data Encryption Standard", FIPS PUB 46 (January 1977).

[MODES] National Bureau of Standards, "DES Modes of Operation", FIPS PUB 81 (December 1980).

[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. 802.11-2003, 2003.

[IEEE802.11i] Institute of Electrical and Electronics Engineers, " Supplement to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security", IEEE 802.11i, November 2004.

[IEEE8021X] IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std 802.1X-2004, November 2004.

[PEAPv0] Kamath, V., Palekar, A. and M. Wodrich, "Microsoft's PEAP version 0 (Implementation in Windows XP SP1)", draft-kamath-pppext-peapv0-00.txt, Internet draft (work in progress), July 2002.

[CompoundBinding] Puthenkulam, J., Lortz, V., Palekar, A. and D. Simon, "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04.txt, Internet Draft (work in progress), October 2003.

Appendix A - Examples

A.1 Cleartext Identity Exchange

In the case where an identity exchange occurs within PEAPv2 Part 1, the conversation will appear as follows:

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- EAP-Request/
                        Identity
EAP-Response/
Identity (MyID1) ->

// Identity sent in the clear. May be a hint to help route the
authentication request to EAP server, instead of the full user
identity.

                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (PEAP Start, S bit set)
EAP-Response/
EAP-Type=PEAP, V=2
(TLS client_hello)->
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (TLS server_hello,
                         TLS certificate,
                  [TLS server_key_exchange,]
                  [TLS certificate_request,]
                      TLS server_hello_done)
EAP-Response/
EAP-Type=PEAP, V=2
([TLS certificate,]
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (TLS change_cipher_spec,
                         TLS finished,
                         EAP-Request/EAP-Type=EAP-TLV
                        [EAP-Payload-TLV[EAP-Request/
                         Identity]])


// identity protected by TLS. EAP-TLV packet does not include an EAP-
```

header.

TLS channel established (EAP messages sent within TLS channel encapsulated in EAP-TLV packets without EAP header)

```
EAP-TLV [EAP-Payload-TLV
[EAP-Response/Identity (MyID2)]]]->

                          <- EAP-TLV [EAP-Payload-TLV
                          [EAP-Request/EAP-Type=X]]

EAP-TLV [EAP-Payload-TLV
[EAP-Response/EAP-Type=X]] ->

// Protected termination

                          <- EAP-TLV [Result TLV (Success),
                          Crypto-Binding TLV (Version=2,
                          received-version=2, Nonce, B1_MAC),
                          Intermediate-Result TLV (Success)]

EAP-TLV [Result TLV (Success),
Intermediate-Result TLV (Success),
Crypto-Binding TLV (Version=2,
received-version=2,Nonce, B2_MAC)]->

TLS channel torn down
(messages sent in cleartext)

                         <- EAP-Success
```

A.2 No cleartext Identity Exchange

Where all peers are known to support PEAPv2, a non-certificate authentication is desired for the client and the PEAPv2 Part 1 conversation is carried out between the peer and a local EAP server, the cleartext identity exchange may be omitted and the conversation appears as follows:

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (PEAP Start, S bit set)

EAP-Response/
EAP-Type=PEAP, V=2
(TLS client_hello)->
```

```
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS server_hello,
                           TLS certificate,
                    [TLS server_key_exchange,]
                    [TLS certificate_request,]
                        TLS server_hello_done)
   EAP-Response/
   EAP-Type=PEAP, V=2
   ([TLS certificate,]
    TLS client_key_exchange,
   [TLS certificate_verify,]
    TLS change_cipher_spec,
    TLS finished) ->
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (TLS change_cipher_spec,
                            TLS finished,
                           EAP-TLV [EAP-Payload-TLV
                           (EAP-Request/Identity)])

   TLS channel established
   (messages sent within the TLS channel)

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/Identity (MyID)]] ->

                          <- EAP-TLV [EAP-Payload-TLV
                          [EAP-Type=EAP-Request/
                          EAP-Type=X]]

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/EAP-Type=X
   or NAK] ->
                          <- EAP-TLV [EAP-Payload-TLV
                          [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload-TLV [EAP-Response/
   EAP-Type=X]] ->

   // Protected success
                           <- EAP-TLV [Crypto-Binding TLV=
                           (Version=2, Received-version=2,
                           Nonce, B1_MAC),
                           Intermediate-Result TLV(Success),
                           Result TLV (Success)]

   EAP-TLV [Crypto-Binding TLV=
```

```
(Version=2,Received-version=2,
Nonce, B2_MAC),
Intermediate-Result TLV (Success),
Result TLV (Success)]->

TLS channel torn down
(messages sent in cleartext)

                        <- EAP-Success
```

A.3 Client certificate authentication with identity privacy

Where all peers are known to support PEAPv2, where client certificate authentication is desired and the PEAPv2 Part 1 conversation is carried out between the peer and a local EAP server, the cleartext identity exchange may be omitted and the conversation appears as follows:

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (PEAP Start, S bit set)

EAP-Response/
EAP-Type=PEAP, V=2
(TLS client_hello)->
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (TLS server_hello,
                         TLS certificate,
                         [TLS server_key_exchange,]
                         TLS server_hello_done)
EAP-Response/
EAP-Type=PEAP, V=2
(TLS client_key_exchange,
 TLS change_cipher_spec,
 TLS finished) ->
                        <- EAP-Request/
                        EAP-Type=PEAP, V=2
                        (TLS change_cipher_spec,
                         TLS finished,TLS Hello-Request)
EAP-Response/
EAP-Type=PEAP, V=2
(TLS client_hello)->

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

```
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS server_hello,
                           TLS certificate,
                    [TLS server_key_exchange,]
                    [TLS certificate_request,]
                        TLS server_hello_done)
   EAP-Response/
   EAP-Type=PEAP, V=2
   ([TLS certificate,]
    TLS client_key_exchange,
   [TLS certificate_verify,]
    TLS change_cipher_spec,
    TLS finished) ->

                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS change_cipher_spec,
                           TLS finished, EAP-TLV
                          [Crypto-Binding TLV (version=2,
                          Received-version=2, Nonce,
                          B1_MAC),
                          Result TLV (Success)])

   // packet format within the TLS channel

   EAP-TLV [
   Crypto-Binding TLV=(Version=2,
   Received-version=2,
   Nonce, B2_MAC),
   Result TLV (Success)]

   TLS channel torn down
   (messages sent in cleartext)

                          <- EAP-Success
```

A.4 Fragmentation and Reassembly

In the case where PEAPv2 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=2
                          (PEAP Start, S bit set)

   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (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=2 ->

                          <- EAP-Request/
                             EAP-Type=PEAP, V=2
                          (Fragment 2: M bit set)
   EAP-Response/
   EAP-Type=PEAP, V=2 ->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (Fragment 3)
   EAP-Response/
   EAP-Type=PEAP, V=2
   ([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=2
   EAP-Response/
   EAP-Type=PEAP, V=2
   (Fragment 2)->
                         <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS change_cipher_spec,
                           TLS finished, EAP-TLV
                          [EAP-Payload-TLV[
                          EAP-Request/Identity]])
```

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

   EAP-TLV
   [EAP-Payload-TLV
   [EAP-Response/Identity(myID)]] ->

                          <- EAP-TLV [ EAP-Payload-TLV
                         [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/EAP-Type=X or NAK]]->

                          <- EAP-TLV [ EAP-Payload-TLV
                         [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/EAP-Type=X] ->

                          <- EAP-TLV [Crypto-Binding TLV
                          =(Version=0, Received-Version=2,
                          Nonce, B1_MAC),
                          Intermediate-Result TLV(Success),
                          Result TLV (Success)]

   EAP-TLV [
   Crypto-Binding TLV=(Version=2,
   Received-Version=2,Nonce, B2_MAC),
   Result TLV (Success),
   Intermediate-Result TLV (Success)] ->

   TLS channel torn down
   (messages sent in cleartext)

                           <- EAP-Success
```

A.5 Server authentication fails in Part 2

In the case where the server authenticates to the client successfully in PEAPv2 Part 1, but the client fails to authenticate to the server in PEAPv2 Part 2, the conversation will appear as follows:

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

```
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (PEAP Start, S bit set)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS server_hello,
                           TLS certificate,
                    [TLS server_key_exchange,]
                    [TLS certificate_request,]
                        TLS server_hello_done)
   EAP-Response/
   EAP-Type=PEAP, V=2
   ([TLS certificate,]
    TLS client_key_exchange,
   [TLS certificate_verify,]
    TLS change_cipher_spec,
    TLS finished) ->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS change_cipher_spec,
                           TLS finished, EAP-TLV
                          [EAP-Payload-TLV
                          [EAP-Request/Identity]])


   TLS channel established
   (messages sent within the TLS channel)

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/Identity (MyID)]] ->

                           <- EAP-TLV [EAP-Payload-TLV
                          [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload
   [EAP-Response/EAP-Type=X
   or NAK]] ->
                         <- EAP-TLV [EAP-Payload
                          [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload
   [EAP-Response/
   EAP-Type=X]] ->
                           <- EAP-TLV [Crypto-Binding TLV
                           (Version=2, Received-Version=2,
```

```
                          Nonce, B1_MAC),
                          Intermediate-Result TLV (Failure),
                          Result TLV (Failure)]

   EAP-TLV [Crypto-Binding TLV
   (Version=0, Received-version=2,
   Nonce, B2_MAC),
   Result TLV (Failure),
   Intermediate-Result TLV (Failure)]

   (TLS session cache entry flushed)
   TLS channel torn down
   (messages sent in cleartext)

                          <- EAP-Failure
```

A.6 Server authentication fails in Part 1

In the case where server authentication is unsuccessful in PEAPv2 Part 1, the conversation will appear as follows:

```
   Authenticating Peer     Authenticator
   -------------------     -------------
                          <- EAP-Request/
                          Identity
   EAP-Response/
   Identity (MyID) ->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (PEAP Start)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS server_hello,
                           TLS certificate,
                     [TLS server_key_exchange,]
                         TLS server_hello_done)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_key_exchange,
   [TLS certificate_verify,]
    TLS change_cipher_spec,
    TLS finished) ->
                          <- EAP-Request/
                          EAP-Type=PEAP, V=2
                          (TLS change_cipher_spec,
```

```
                          TLS finished, EAP-TLV
                         [EAP-Payload-TLV [
                         EAP-Request/Identity]])
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS Alert message) ->
                         <- EAP-Failure
                         (TLS session cache entry flushed)
```

A.7 Session resume success

In the case where a previously established session is being resumed, the EAP server supports TLS session cache flushing for unsuccessful PEAPv2 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=2
                         (PEAP Start)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                         <- EAP-Request/
                         EAP-Type=PEAP, V=2
                         (TLS server_hello,
                         TLS change_cipher_spec
                         TLS finished)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS change_cipher_spec,
    TLS finished) ->
                         <- EAP-Request/
                            EAP-Type=EAP-TLV
                            Result TLV (Success)

   // Compound MAC calculated using TLS keys since there were no inner
   EAP methods.

   EAP-Response/
   EAP-Type=EAP-TLV
   Crypto-Binding TLV=(Version=2, Nonce, B2_MAC),
   Result TLV (Success)->
```

```
   TLS channel torn down
   (messages sent in cleartext)

                           <- EAP-Success
```

A.8 Session resume failure

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=2
                           (PEAP Start)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello) ->
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (TLS server_hello,
                            TLS change_cipher_spec,
                            TLS finished)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS change_cipher_spec,
    TLS finished) ->
                           <- EAP-Request
                           EAP-Type=PEAP, V=2
                           (TLS Alert message)
   EAP-Response
   EAP-Type=PEAP, V=2 ->
                            <- EAP-Failure
                            (TLS session cache entry flushed)
```

A.9 Session resume failure

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-Request/
                           Identity
   EAP-Response/
   Identity (MyID) ->
                           <- EAP-Request/
                           EAP-Request/
                           EAP-Type=PEAP, V=2
                           (PEAP Start)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (TLS server_hello,
                            TLS change_cipher_spec,
                            TLS finished)
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS change_cipher_spec,
   TLS finished)
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS Alert message) ->

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

A.10 PEAP version negotiation

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 an implementation compliant with this specification), the conversation will occur as follows:

```
   Authenticating Peer     Authenticator
   -------------------     -------------
                          <- EAP-Request/
                           Identity
   EAP-Response/
   Identity (MyID) ->
                           <- EAP-Request/
                           EAP-Request/
                           EAP-Type=PEAP, V=2
                           (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)


   //conversation continued using pre-standard PEAP version 0
```

A.11 Sequences of EAP methods

Where PEAPv2 is negotiated, with a sequence of EAP method X followed by method Y, the conversation will occur as follows:

```
   Authenticating Peer     Authenticator
   -------------------     -------------
                           <- EAP-Request/
                           Identity
   EAP-Response/
   Identity (MyID1) ->
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (PEAP Start, S bit set)

   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->
                           <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (TLS server_hello,
                            TLS certificate,
                     [TLS server_key_exchange,]
                     [TLS certificate_request,]
                         TLS server_hello_done)
   EAP-Response/
   EAP-Type=PEAP, V=2
   ([TLS certificate,]
    TLS client_key_exchange,
   [TLS certificate_verify,]
    TLS change_cipher_spec,
    TLS finished) ->
                          <- EAP-Request/
                           EAP-Type=PEAP, V=2
                           (TLS change_cipher_spec,
                            TLS finished, EAP-TLV
```

```
                          [EAP-Payload TLV[
                          EAP-Request/Identity]])

   TLS channel established
   (messages sent within the TLS channel)

   EAP-TLV [EAP-Payload TLV
   [EAP-Response/Identity]] ->

                        <- EAP-TLV [EAP-Payload TLV
                        [EAP-Request/EAP-Type=X]]]

   EAP-TLV [EAP-Payload-TLV
   [EAP-Response/EAP-Type=X]] ->

                         <- EAP-TLV [ EAP-Payload TLV
                        [EAP-Request/EAP-Type=X]]

   EAP-TLV [EAP-Payload TLV
   [EAP-Response/EAP-Type=X]]->

                          <- EAP-TLV [EAP-Payload TLV [EAP-Type=Y],
                          (Intermediate Result TLV (Success),
                          Crypto-Binding TLV
                          (Version=2, Received-version=2,
                          Nonce, B1_MAC))]

   // Next EAP conversation started after successful completion of
   previous method X. The Intermediate-Status and Crypto-Binding TLVs
   are sent in next packet to minimize round-trips.  In this example,
   identity request is not sent before negotiating EAP-Type=Y.

   EAP-TLV [EAP-Payload-TLV [EAP-Type=Y],
   (Intermediate Result TLV (Success),
   Crypto-Binding TLV (Version=2,
   Received-version=2, Nonce, B2_MAC))]->

   // Compound MAC calculated using Keys generated from
   EAP methods X and the TLS tunnel.

                          <- EAP-TLV [EAP Payload TLV [
                          EAP-Type=Y]]

   EAP-TLV[EAP Payload TLV
   [EAP-Type=Y]] ->
                          <- EAP-TLV [Result TLV (Success),
                          Intermediate Result TLV (Success),
                          Crypto-Binding TLV
```

```
                          (Version=2, Received-version=2,
                          Nonce, B1_MAC))]

   EAP-TLV [(Result TLV (Success),
   Intermediate Result TLV (Success),
   Crypto-Binding TLV (Version=2,
   Received-version=2, Nonce, B2_MAC))]->

   // Compound MAC calculated using Keys generated from EAP methods X
   and Y and the TLS tunnel.  // Compound Keys generated using Keys
   generated from EAP methods X and Y; and the TLS tunnel.

   TLS channel torn down (messages sent in cleartext)

                          <- EAP-Success
```

A.12 Successful Session Resume with Server-Identifier TLV

In the case where an server-identifier TLV is used by server to indicate its identity, the conversation will appear as follows:

```
   Authenticating Peer     Authenticator
   -------------------     -------------
                          <- EAP-Request/
                             Identity
   EAP-Response/
   Identity (MyID1) ->

   // The T bit is set to indicate the TLS message length. The bytes
   after TLS Data (which is in this case is zero length) are the Outer-
   TLVs. The Server-Identifier-TLV is sent in the clear.

                          <- EAP-Request/
                             EAP-Type=PEAP, V=2
                            (PEAP Start, S bit set, T bit set),
                            TLS message length=0,
                            Server-Identitifer TLV
   // The Server-Identifier TLV identifies the server. The peer selects
   a session handle that corresponds to that specific server to that the
   TLS session could be resumed (instead of going through the full
   handshake)

   EAP-Response/
   EAP-Type=PEAP, V=2
   (TLS client_hello)->

   // subsequent packets are same as example "session resume success"
```

Acknowledgments

Thanks to Hakan Andersson, Jan-Ove Larsson and Magnus Nystrom of RSA Security; Bernard Aboba, Vivek Kamath, Stephen Bensley and Narendra Gidwani of Microsoft; Ilan Frenkel and Nancy Cam-Winget of Cisco; Jose Puthenkulam of Intel for their contributions and critiques.

The compound binding exchange to address man-in-the-middle attack is based on the draft "The Compound Authentication Binding Problem"[CompoundBinding].

The vast majority of the work by Simon Josefsson and Hakan Andersson was done while they were employed at RSA Laboratories.

Author Addresses

Ashwin Palekar
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Phone: +1 425 882 8080
EMail: ashwinp@microsoft.com

Dan Simon
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Phone: +1 425 706 6711
EMail: dansimon@microsoft.com

Glen Zorn
Cisco Systems
500 108th Avenue N.E.
Suite 500
Bellevue, Washington 98004

Phone: + 1 425 438 8210
Fax: + 1 425 438 1848
EMail: gwz@cisco.com

Simon Josefsson
Drottningholmsvagen 70
112 42 Stockholm
Sweden

Phone: +46 8 619 04 22

EMail: jas@extundo.com

Hao Zhou
Cisco Systems, Inc.
4125 Highlander Parkway
Richfield, OH 44286

Phone: +1 330 523 2132
Fax: +1 330 523 2239
EMail: hzhou@cisco.com

Joseph Salowey
Cisco Systems
2901 3rd Ave
Seattle, WA 98121

Phone: +1 206 256 3380
EMail: jsalowey@cisco.com

Intellectual Property Statement

Disclaimer of Validity

INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Open Issues

Open issues relating to this specification are tracked on the following web site:

http://www.drizzle.com/~aboba/PEAP/