PPPEXT Working Group
INTERNET-DRAFT
Category: Standards Track
<draft-josefsson-pppext-eap-tls-eap-06.txt>
22 March 2003

Ashwin Palekar
Dan Simon
Microsoft
Glen Zorn
Cisco
S. Josefsson
Extundo

Protected EAP Protocol (PEAP)

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

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

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

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

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

Copyright Notice

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

Abstract

The Extensible Authentication Protocol (EAP), defined in RFC 2284, provides for support of multiple authentication methods. While EAP was originally created for use with PPP, it has since been adopted for use with IEEE 802.1X "Network Port Authentication".

Since its deployment, a number of weaknesses in EAP or some EAP protocols have become apparent. These include no per packet confidentiality and integrity protection; which results in lack of protection to user identity, notification messages or EAP negotiation; and sequencing of EAP methods. In addition, there is no standardized mechanism for key exchange; no built-in support for fragmentation and reassembly; no support for acknowledged success/failure indications; and no support for fast reconnect.

In addition, some EAP protocols (e.g. like EAP-MD5) are susceptible to dictionary and brute force attacks; do not provide confidentiality; do not support server authentication required to prevent spoofing by rogue servers (gateways), and do not support the generation of key strength required for 802.11i.

By wrapping the EAP protocol within TLS, Protected EAP (PEAP) addresses these deficiencies in EAP or EAP protocols. EAP method(s) running within PEAP are provided with built-in support for

Privacy of user identity.
Protection to individual EAP methods. For example, protection can provide dictionary attack resistance to protocols susceptible to that attack.
Protected EAP notification.
Protected sequencing of EAP methods.
Protected negotiation.
Protected EAP header.
Protected exchange of parameters (TLVs) between client and server.
Standardized mechanism for key exchange.
Proven key derivation and management.
Session resumption.
Server authentication.
Protected Acknowledged and result exchange.
Fragmentation and reassembly.

Table of Contents

1. Introduction

The Extensible Authentication Protocol (EAP), described in [RFC2284], provides a standard mechanism for support of multiple authentication methods. Through the use of EAP, support for a number of authentication schemes may be added, including smart cards, Kerberos, Public Key, One Time Passwords, and others.

EAP was developed or use on wired networks, where physical security was presumed. EAP over PPP, defined in [RFC2284], is typically deployed with leased lines or modem connections, requiring an attacker to gain access to the telephone network in order to snoop on the conversation or inject packets. [IEEE8021X] defines EAP over IEEE 802 local area networks(EAPOL), presuming the existence of switched media; in order to snoop or inject packets, an attacker

would need to gain administrative access to
the switch. Due to the presumption of physical security, facilities for protection of the EAP conversation were not provided.

Where an attacker can easily gain access to the medium (such as on a wireless network or where EAP is run over IP), the presumption of physical security is no longer valid. Since the EAP method negotiation is unprotected, an attacker can inject packets in order to cause the negotiation of a method with lesser security. Denial of service attacks are also possible. Since the initial EAP Identity Request/Response exchange is sent in the clear, an attacker snooping on the conversation can collect user identities for use in subsequent attacks.

By initially negotiating a TLS channel, and then conducting the EAP conversation within it, PEAP provides for per-packet encryption, authentication, integrity and replay protection of the EAP conversation.

Benefits include:

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

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

Protected negotiation
Since within PEAP, the EAP conversation is authenticated, integrity and replay protected on a per-packet basis, the EAP method negotiation that occurs within PEAP is protected, as are error messages sent within the TLS channel (TLS alerts or EAP Notification packets).

Header protection
Within PEAP, the EAP conversation is conducted within a TLS channel. As a result, the EAP header is protected against modification.

Protected termination

By sending success/failure indications within the TLS channel, PEAP 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 PEAP,methods leveraging PEAP 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. PEAP supports this capability by leveraging the TLS session resumption facility, and any EAP method running under PEAP can take advantage of it.

Proven and Method independent key management
In order to provide keying material for a wide range of link layer ciphersuites, EAP methods need to provide a key hierarchy generating authentication and encryption keys, as well as

initialization vectors. Development of a secure key hierarchy is complex, and not easy to generalize for all EAP methods. By relying on the well-reviewed TLS [RFC2246] key derivation method, PEAP provides the required keying material for any EAP method running within it. This frees EAP method developments from creating keying material with key strength required for 802.11i wireless LAN. If EAP methods will also be deployed without the protection of PEAP or IPSEC, then the EAP methods should derive key material of sufficient strength to prevent a Man-in-the-middle attack described in the compound binding draft [CompoundBinding].

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 requiring the authentication.

Backend Authentication Server
: An Authentication Server is an entity that provides an Authentication Service to an NAS. This service verifies from the credentials provided by the peer, the claim of identity made by the peer.

EAP server
: The EAP server is the entity that terminates the EAP conversation with the peer. The EAP server may reside on the NAS, or alternatively within a backend authentication server.

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

Master key
: The key derived between the EAP client and EAP server during the EAP authentication process.

Master session key
: The keys derived from the master key are subsequently used in generation of the transient session keys for authentication, encryption, binding exchange, and IV-generation.

NAS
: Short for "Network Access Server".

Peer
: The other end of the point-to-point link (PPP), point-to-point LAN segment (IEEE 802.1X) or 802.11 wireless link, which is being authenticated by the NAS. In IEEE 802.1X, this end is known as the Supplicant.

TLS Ciphersuite
: The ciphersuite negotiated for protection of the PEAP Part 2

conversation.

Transient session keys
: The transient session keys are derived from the master session keys, and are of the appropriate size and type for use with the chosen link layer ciphersuite.

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.

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

As a result, where the NAS acts as a "passthrough" it does not have knowledge of the TLS master secret derived between the EAP Peer and the EAP server. In order to provide keying material for link-layer ciphersuites, the NAS obtains the master session keys, which are derived from the TLS master secret via a one-way function. This

enables the NAS and EAP peer to derive keys suitable for encrypting, authenticating and integrity protecting session data. However, the

NAS cannot decrypt the PEAP conversation or spoof session resumption, since this requires knowledge of the TLS master secret.

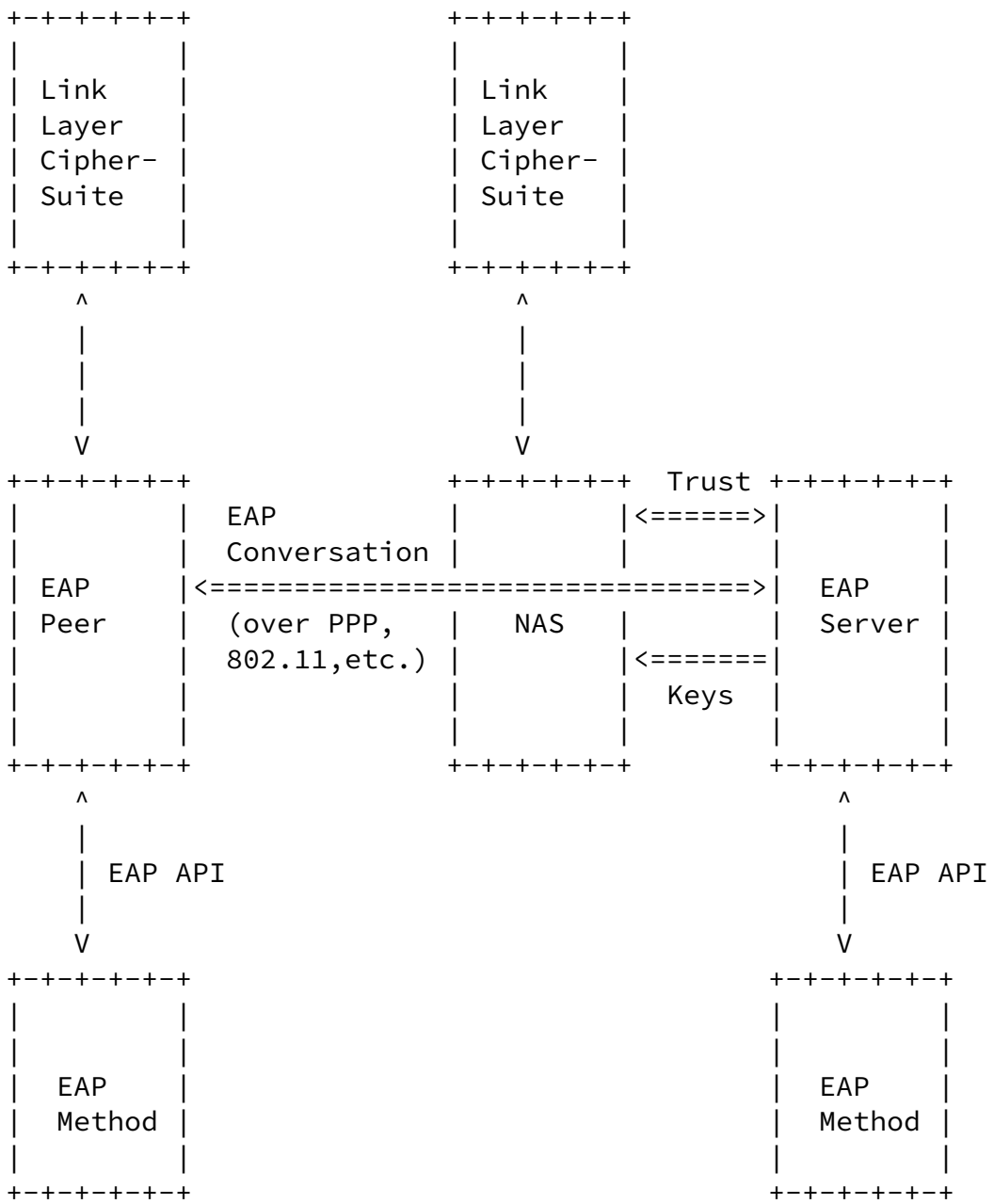


Figure 1 - Relationship between EAP client, backend authentication server and NAS.

2. Protocol overview

Protected EAP (PEAP) 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, a complete EAP conversation is carried out, unless part 1 provided client authentication.

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

2.1. PEAP Part 1

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, containing the peer's EAP-ID. Since the initial identity exchange is used primarily to route the EAP conversation to the EAP server, 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 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 and is provisioned with the appropriate certificates, 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.

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, and no data. 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 data field of the EAP-Response packet will encapsulate one or more TLS records in TLS record layer format, containing a TLS client_hello handshake message. The current cipher spec for the TLS records will be TLS_NULL_WITH_NULL_NULL and null compression. This current cipher spec remains the same until the change_cipher_spec

message signals that subsequent records will have the negotiated attributes for the remainder of the handshake.

The client_hello message contains the client's TLS version number, a sessionId, a random number, and a set of TLS ciphersuites supported by the client. The version offered by the client MUST correspond to TLS v1.0 or later.

The EAP server will then respond with an EAP-Request packet with EAP-Type=PEAP. The data field of this packet will encapsulate one or more TLS records. These will contain a TLS server_hello handshake message, possibly followed by TLS certificate, server_key_exchange, certificate_request, server_hello_done and/or finished handshake messages, and/or a TLS change_cipher_spec message.

Since after the TLS session is established, another complete EAP negotiation will occur and the peer will authenticate using a secondary mechanism, with PEAP the client need not authenticate as part of TLS session establishment. As a result, although the EAP-Request packet sent by the EAP Server MAY contain a certificate_request message, this is not required.

The certificate_request message indicates that the server desires the client to authenticate itself via public key. Typically when the EAP server sends a certificate_request message, the intent is to complete the PEAP authentication without requiring negotiation of an additional EAP method. However, it is valid for the server to request a certificate in the server_hello and for the client refuse to provide one. In this case, the EAP server MUST require that PEAP Part 2 be completed.

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. 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.

The server_hello handshake message contains a TLS version number, another random number, a sessionId, and a TLS ciphersuite. The version offered by the server MUST correspond to TLS v1.0 or later. In order to provide confidentiality, integrity and replay protection, and authentication, the negotiated TLS ciphersuite MUST provide all of these security services.

If the client's sessionId is null or unrecognized by the server, the server MUST choose the sessionId to establish a new session; otherwise, the sessionId will match that offered by the client, indicating a resumption of the previously established session with that sessionId. The server will also choose a TLS ciphersuite from

---

those offered by the client; if the session matches the client's, then the TLS ciphersuite MUST match the one negotiated during the handshake protocol execution that established the session.

PEAP 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 to support some TLS ciphersuites (e.g. TLS ciphersuites utilizing the IDEA encryption algorithm). To ensure interoperability, PEAP peers and Authenticators MUST support and be able to negotiate the following TLS ciphersuites:

```
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA (FIPS compliant)
```

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

If the EAP server is not resuming a previously established session, then it MUST include a TLS server_certificate handshake message, and a server_hello_done handshake message MUST be the last handshake message encapsulated in this EAP-Request packet.

The certificate message contains a public key certificate chain for either a key exchange public key (such as an RSA or Diffie-Hellman

key exchange public key) or a signature public key (such as an RSA or DSS signature public key). In the latter case, a TLS server_key_exchange handshake message MUST also be included to allow the key exchange to take place.

The peer MUST respond to the EAP-Request with an EAP-Response packet of EAP-Type=PEAP. The data field of this packet will encapsulate one or more TLS records containing a TLS change_cipher_spec message and finished handshake message, and possibly certificate, certificate_verify and/or client_key_exchange handshake messages. 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.

If the preceding server_hello message sent by the EAP server in the preceding EAP-Request packet did not indicate the resumption of a previous session, then the peer MUST send, in addition to the change_cipher_spec and finished messages, a client_key_exchange message, which completes the exchange of a shared master secret between the peer and the EAP server.

The EAP server MUST then respond with an EAP-Request packet with EAP-Type=PEAP, which includes, in the case of a new TLS session, one or more TLS records containing TLS change_cipher_spec and finished handshake messages. The latter contains the EAP server's authentication response to the peer. The peer will then verify the hash in order to authenticate the EAP server.

If the EAP server authenticates unsuccessfully, the peer MAY send an EAP-Response packet of EAP-Type=PEAP containing a TLS Alert message identifying the reason for the failed authentication. The peer MAY send a TLS alert message rather than immediately terminating the conversation so as to allow the EAP server to log the cause of the error for examination by the system administrator.

To ensure that the EAP Server receives the TLS alert message, the peer MUST wait for the EAP-Server to reply before terminating the conversation. The EAP Server MUST reply with an EAP-Failure packet since server authentication failure is a terminal condition.

If the EAP server authenticates successfully, the peer MUST send an EAP-Response packet of EAP-Type=PEAP, and no data. The EAP-Server then continues with Part 2 of the PEAP conversation.

2.2. PEAP Part 2

The second portion of the PEAP conversation consists of another complete EAP conversation occurring within the TLS session negotiated in PEAP Part 1. It will therefore occur only if establishment of a new TLS session in Part 1 is successful or a TLS session is successfully resumed in Part 1.

It MUST NOT occur if the EAP Server authenticates unsuccessfully or if an EAP-Failure has been sent by the EAP Server to the peer, terminating the conversation. Since all packets sent within the PEAP 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 are protected using the negotiated TLS ciphersuite.

Part 2 of the PEAP conversation typically begins with the Authenticator sending an EAP-Request/Identity packet to the peer, protected by the TLS ciphersuite negotiated in PEAP Part 1. The peer responds with an EAP-Response/Identity packet to the authenticator, containing the peer's userId. Since this Identity Request/Response exchange is protected by
the ciphersuite negotiated in TLS, it is protected against 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 EAP-Type set to the initial method. As described in [RFC2284], 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.

2.3. Version negotiation

PEAP packets contain a three bit version field, which enables PEAP

implementations to be backward compatible with previous versions of the protocol. Implementations of this specification MUST use a version field set to 2. This specification documents the protocol for version 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 client 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 client does not support this version, it responds with an EAP-Response of EAP type=PEAP and the highest supported version number.

[4] If the EAP server supports the version proposed by the client, then all future EAP-Request packets of EAP type=PEAP MUST include the version field set to the agreed upon version number. Similarly, the EAP client MUST include the agreed upon version number in all EAP-Response packets of EAP type=PEAP.

[5] If the PEAP server does not support the version number proposed by the PEAP client, it terminates the conversation, as described in Section 2.4.

This version negotiation procedure guarantees that the EAP client 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. In order to protect against a downgrade version attack between PEAP versions support by the peers, the peers MUST exchange information on the highest version number supported during the binding exchange.

2.4. Termination

As described in [RFC2284], 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 packets may

be used by a rogue NAS to convince a peer to let itself access the network, even though the NAS has not authenticated itself.

By requiring mutual authentication and by supporting encrypted, authenticated and integrity protected success/failure indications, (described below as "protected" indications) PEAP provides protection against these attacks. Within PEAP, protected success/failure indications are supported by sending these indications within the TLS channel.

PEAP support for protected success/failure indications is constrained by the [RFC2284] and [IEEE8021X] specifications. In [IEEE8021X], the authenticator "manufactures" cleartext EAP Success and Failure packets based on the result indicated by the backend authentication server. As a result, were a PEAP server to send a protected EAP Success or EAP Failure packet as the final packet within the EAP exchange, authenticators compliant with [IEEE8021X] would silently discard the packet, and replace it with a cleartext EAP Success or Failure. Since the client will discard these unprotected indications, where an authenticator compliant with [IEEE8021X] is present, it is not be possible to conclude a successful authentication. As a result, this approach does not provide reliable authenticated success/failure indications on all media.

In addition, [RFC2284] 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 PEAP server SHOULD NOT send a protected EAP Success or EAP Failure packet as the final packet within a PEAP conversation. However, in the spirit of being "conservative in what you send, liberal in what you receive", a PEAP client SHOULD accept and process such a packet if it is received. This behavior makes it possible for implementations to save a round-trip (improving the performance of fast reconnect), assuming that the authentication occurs within a low packet loss environment in which "manufacture" of packets is guaranteed not to occur.

Instead, EAP servers MUST utilize the acknowledged and protected success/failure indications defined in Section 4. In this approach, the PEAP server sends the success/failure indication as an EAP-Request with type=33 (EAP TLV), protected within the TLS channel. The PEAP client then replies with a protected success/failure

indication as an EAP-Response with type=33 (EAP TLV). The conversation concludes with the PEAP server sending a cleartext success/failure indication.

Since both sides have already concluded a protected termination conversation, this final packet is ceremonial.

Use of a protected and acknowledged success/failure indication provides the PEAP protocol immunity against the "manufacture" of cleartext success/failure indications mandated by [IEEE8021X]. It also enables both sides of the conversation to communicate the outcome of PEAP mutual authentication, although the TLS alert mechanism already provides this capability to some extent. On the other hand, this approach requires an extra round-trip, which affects the performance of fast reconnect.

Once PEAP has been selected as the authentication method, compliant PEAP implementations MUST silently discard unprotected success indications (e.g. cleartext EAP Success) unless both the PEAP peer and server have indicated a successful authentication exchange via the mechanism described in Section 4.

Similarly, once the TLS channel has been set up, compliant PEAP implementations MUST silently discard unprotected failure indications (e.g. cleartext EAP Failure) unless they are proceeded by a protected failure indication. Protected failure indications include the TLS alert mechanism, as well the indication mechanism described in Section 4. For example, if a PEAP peer has previously received a protected EAP-Request of Type=33 (EAP TLV) with Result=Failure, or if it has received a protected EAP-Request of Type=33 (EAP-TLV) with Result=Success, and responded with a protected EAP-Response of Type=33 (EAP-TLV) with Result=Failure, then it will accept and process a cleartext EAP Failure. However, if a PEAP peer has previously received a protected EAP-Request of Type=33 (EAP-TLV) with Result=Success, and has responded with a protected EAP-Request of Type=33 (EAP-TLV) with Result=Success, then an unprotected failure indication MUST be silently discarded.

Prior to establishment of the TLS channel, no keying material exists, so that protected success/failure indications are not possible. However, within PEAP a failure to establish the TLS channel (e.g. failure to verify the server certificate) is considered an unrecoverable error, so that where this failure has occurred, an unprotected failure indication can be safely accepted.

2.5. Error handling

Other than supporting TLS alert messages, PEAP does not have its own error message capabilities. This is unnecessary since errors in the PEAP Part 1 conversation are communicated via TLS alert messages, and errors in the PEAP Part 2 conversation are expected to be

handled by individual EAP methods.

If an error occurs at any point in the PEAP conversation, 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.

2.6. Retry behavior

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

For example, if the initial PEAP start packet sent by the EAP server were to be lost, then the peer would not receive this packet, and would not respond to it. As a result, the PEAP start packet would be resent by the EAP server. Once the peer received the PEAP start packet, it would send an EAP-Response encapsulating the client_hello message. If the EAP-Response were to be lost, then the EAP server would resend the initial PEAP start, and the peer would resend the EAP-Response.

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

2.7. Session resumption

The purpose of the sessionId within the TLS protocol 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.

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.

In the case where the EAP server and the authenticator reside on the

same device, then the client will only be able to continue sessions when connecting to the same NAS or channel server. Should these devices be set up in a rotary or round-robin then it may not be possible for the peer to know in advance the authenticator it will be connecting to, and therefore which sessionId to attempt to reuse. As a result, it is likely that the continuation attempt will fail.

In the case where the EAP authentication is remoted then continuation is much more likely to be successful, since multiple NAS devices and channel servers will remote their EAP authentications to the same backend authentication server.

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.

After a session is successfully resumed, the EAP-Server starts with Part 2 of the PEAP conversation. The peer may have roamed to a different network and successfully resumed with same EAP server. The peer and the EAP server MUST not assume that a session resume implies either of them will skip inner EAP methods.

2.8. 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 PEAP 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 [2], 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 PEAP 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.

PEAP fragmentation support is provided through addition of flag bits within the EAP-Response and EAP-Request packets, as well as a TLS 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 TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS message or set of messages. 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 TLS Message Length field is four octets, and provides the total length of the TLS message or set of messages that is being fragmented; this simplifies buffer allocation.

When a PEAP 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 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.

2.9. 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 for use with the selected link layer ciphersuites.

In the most general case, keying material must be provided for authentication, encryption and initialization vectors (IVs) in each direction.

Since EAP methods may not know the link layer ciphersuite that has

been negotiated, it may not be possible for them to provide link layer ciphersuite-specific keys. In addition, attempting to provide such keys is undesirable, since it would require the EAP method to be revised each time a new link layer ciphersuite is developed. As a result, PEAP derives master session keys which can subsequently be truncated for use with a particular link layer ciphersuite. Since the truncation algorithms are ciphersuite-specific, they are not discussed here; examples of such algorithms are provided in [RFC3079]. 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].

Both the peer and EAP server MUST derive master session keys as described in the compound Session Key derivation section (section 4.2) of the draft Compound Authentication Binding Problem

[compoundbinding].

Algorithms for the truncation of these encryption and authentication master session keys are specific to each link layer ciphersuite. Link layer ciphersuites in use with PPP include DESEbis [RFC2419], 3DES [RFC2420] and MPPE [RFC3078]. IEEE 802.11 ciphersuites are described in [IEEE80211]. An example of how encryption keys for use with MPPE [RFC3078] are derived from the TLS master session keys is given in [RFC3079].

2.10. 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 PEAP, the negotiated TLS ciphersuite relates only to the mechanism by which the PEAP 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, although this capability may be added in future by addition of TLVs to the EAP TLV method defined in Section 4.

3. Detailed description of the PEAP protocol

3.1. PEAP Packet Format

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |   Identifier  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   Flags | Ver |  Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Code

```
   1 - Request
   2 - Response
```

Identifier

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

Length

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

Type

```
   25 - PEAP
```

Flags

```
    0 1 2 3 4
   +-+-+-+-+-+
   |L M S R R|
   +-+-+-+-+-+

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

   The L bit (length included) is set to indicate the presence of the four octet TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS 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.

Version

```
    0 1 2
   +-+-+-+
   |R|1|0|
```

```
   +-+-+-+
```

   R = Reserved (must be zero)

Data

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

3.2. PEAP Request Packet

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |   Identifier  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |   Flags | Ver |      TLS Message Length
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     TLS Message Length        |       TLS Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Code

   1

Identifier

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

Length

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

Type

   25 - PEAP

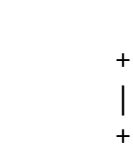
Flags

```
    0 1 2 3 4
   +-+-+-+-+-+
   |L M S R R|
   +-+-+-+-+-+
```

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

The L bit (length included) is set to indicate the presence of the four octet TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS 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.

Version



R = Reserved (must be zero)

TLS Message Length

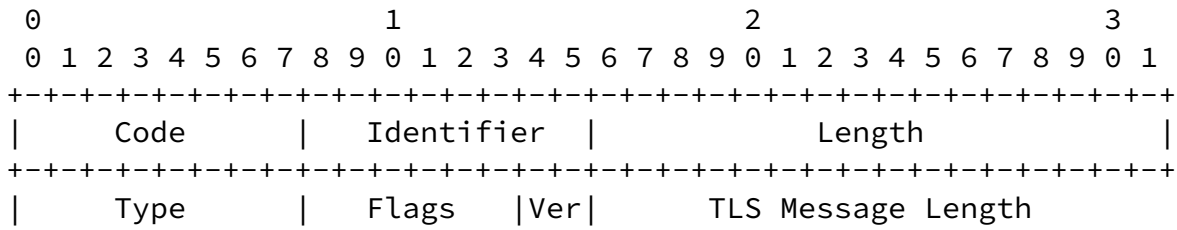
The TLS Message Length field is four octets, and is present only if the L bit is set. This field provides the total length of the TLS message or set of messages that is being fragmented.

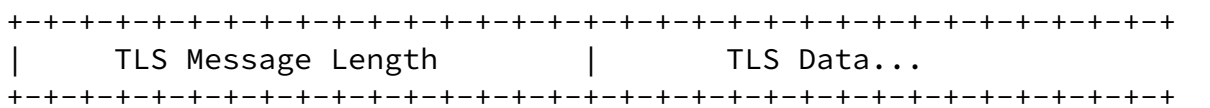
TLS data

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

3.3. PEAP Response Packet

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





Code

2

Identifier

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

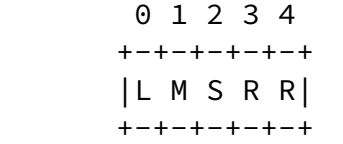
Length

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

Type

25 - PEAP

Flags



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

The L bit (length included) is set to indicate the presence of the four octet TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS 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.

Version

```
 0 1 2
+-+-+-+
|R|1|0|
+-+-+-+
```

R = Reserved (must be zero)

TLS Message Length

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

TLS data

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

4. EAP TLV method

The EAP-TLV method is a payload with 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; cryptographic binding; IPv6 Binding Update.

The EAP peer may not necessarily implement all the TLVs supported by the EAP server; and hence to allow for interoperability, the TLV method allows a EAP server to discover if a TLV is supported by the EAP peer, using the NAK TLV.

The mandatory bit in a TLV indicates that the peer MUST understand the TLV. A peer can determine that a TLV is unknown when it does not support the TLV; or when the TLV is corrupted. The mandatory bit does not indicate that the peer successfully applied the value of the TLV. The specification of a TLV could define additional conditions under which the TLV can be determined to be unknown.

If an EAP peer finds an unknown TLV which is marked as mandatory; it MUST indicate a failure to the EAP server using the NAK TLV; and all the other TLVs in the message MUST be ignored.

If an EAP peer finds an unknown TLV which is marked as optional; then it MUST ignore the TLV. The EAP peer is not required to inform

the EAP server of unknown TLVs which are marked as optional. If the EAP peer finds that the packet has no TLVs, then it MUST send a response with EAP-TLV Response Packet. The Response packet may contain no TLVs.

If an EAP server finds an unknown TLV which is marked as mandatory; the other TLVs in the message MUST be ignored. The EAP server can drop the connection or send a EAP-TLV request packet with NAK-TLV to the EAP client.

Compliant PEAP implementations MUST support the EAP TLV method, processing of mandatory/optional settings on the TLV, the NAK TLV.

TLVs can be contained/nested in other TLVs. A EAP-TLV Request packet is a EAP method; and it can be sequenced before or after any other EAP method. The packet does not have to contain any TLVs or does not have to contain any mandatory TLVs.

4.1. Protected success/failure

Compliant PEAP implementations MUST support acknowledged protected success/failure together with the Binding exchange.

The Result TLV is used to indicate success or failure of the PEAP

tunnel. The PEAP tunnel success/failure packet MUST contain a Result TLV along with the Cryto-Binding TLV. Crypto-Binding TLV may be used in other EAP-TLV packets. Result TLV MUST NOT be sent in packets other than the protected success/failure indication.

If a CRYPTO BINDING TLV does not exist in a packet that contains Result TLV, then the EAP peer must disconnect the connection.

If a CRYPTO BINDING TLV fails validation, then peer must disconnect the connection. Implementations that can delete the TLS handle MUST delete the TLS handle. Implementations that keep track of session state MUST ensure that the session handle cannot be used to skip stage2 authentication.

When using the Result-TLV, the only outcome which should be considered as successful authentication is when an EAP Request of Type=EAP-TLVs with Result TLV of Status=Success is answered by an EAP Response of Type=EAP-TLVs with Result TLV of Status=Success.

If the EAP server has set Result-TLV with Status=Success; and the response from the EAP peer is Status=Failure, then the server MUST either continue EAP conversation or return Result=TLV with Status=Failure. This allows EAP peer to indicate that it refuses to accept the authentication without negotiating certain auth methods as per its policy.

All other combinations (EAP-TLVs Failure, EAP-TLVs Success), (EAP-TLVs Failure, EAP-TLVs Failure), (no EAP-TLVs exchange or no protected EAP Success or Failure, no Crypto-Binding TLVs, crypto-binding TLV validation is not successful) should be considered failed authentications, both by the PEAP peer and authenticator. Once the PEAP peer and authenticator considers them as failed authentications, they are the last packets inside the protected tunnel. These are considered failed authentications regardless of whether a cleartext EAP Success or EAP Failure packet is subsequently sent. Because the EAP-TLVs method is protected within the TLS channel, these packets cannot be spoofed, whereas cleartext EAP Success and EAP Failure messages can be sent by an attacker.

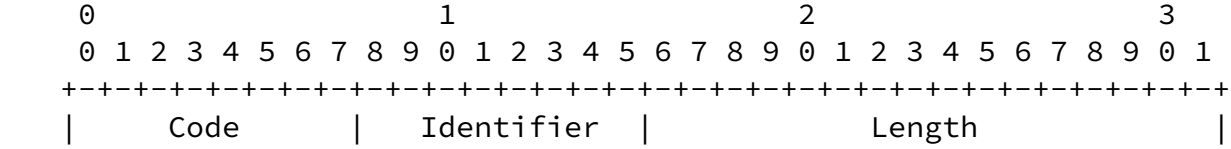
In order for the validation of crypto-binding TLV to be successful, the EAP server and EAP peer should be in-sync on which EAP methods inside the tunnel have been successful. If any or all EAP methods inside the tunnels have failed as per EAP server or EAP peer, then that does not mean the Result will always be set to failure.

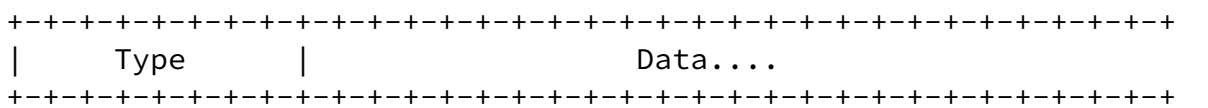
In a successful authentication for a tunnel, the last packet exchange (both request and response) inside the tunnel MUST always contain a valid Crypto-Binding TLV and Result-TLV=Success.

Compliant PEAP implementations MUST support the EAP TLV method, processing of mandatory/optional settings on the TLV, the NAK TLV, Result-TLV, Method-Identity-TLV, Crypto-Binding-TLV.

4.2. EAP-TLV Request Packet

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





Code

1

Identifier

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

Length

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

Type

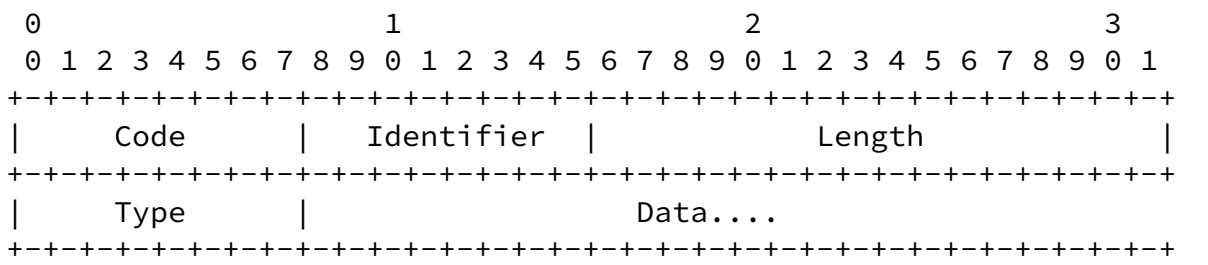
33 - EAP-TLV

Data

The Data field is of variable length, and contains EAP-TLV TLVs.

4.3. EAP-TLV Response Packet

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



Code

2

Identifier

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

Length

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

Type

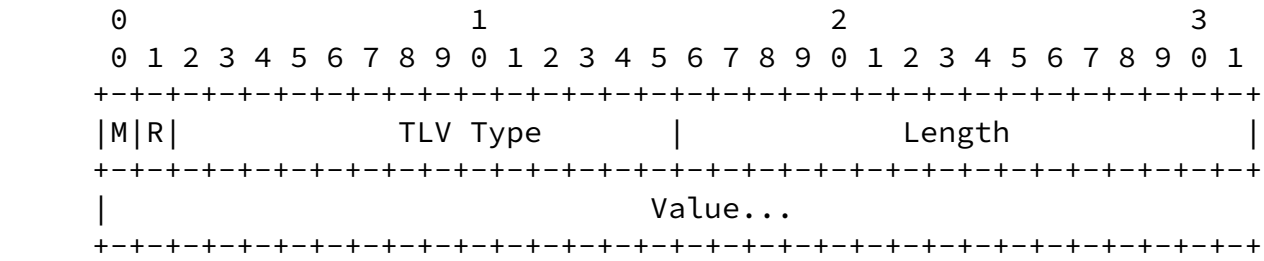
33 - EAP EAP-TLV

Data

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

4.4. EAP-TLV TLV format

EAP-TLV TLVs are defined as follows:



M

0 - Non-mandatory TLV
1 - Mandatory TLV

R

Reserved, set to 0.

TLV Type

A 14-bit field, denoting the attribute type. Allocated TLV Types include:
0 - Reserved
1 - Reserved
2 - Reserved

3 - - RESULT_TLV - Acknowledged Result
4 - NAK_TLV
5 - CRYPTO_BINDING TLV
6 - METHOD_IDENTITY TLV

Length

The length of the Value field in octets.

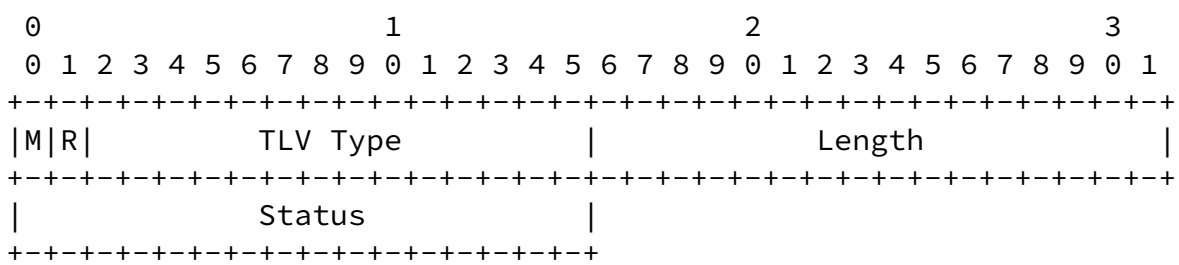
Value

The value of the attribute.

CRYPTO_BINDING_TLV and METHOD_IDENTITY_TLV are defined in the draft Compound Authentication Binding Problem[CompoundBinding].

4.5. Result TLV

The Result TLV provides support for acknowledged Success and Failure messages within PEAP. It is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

3 - Success/Failure

Length

2

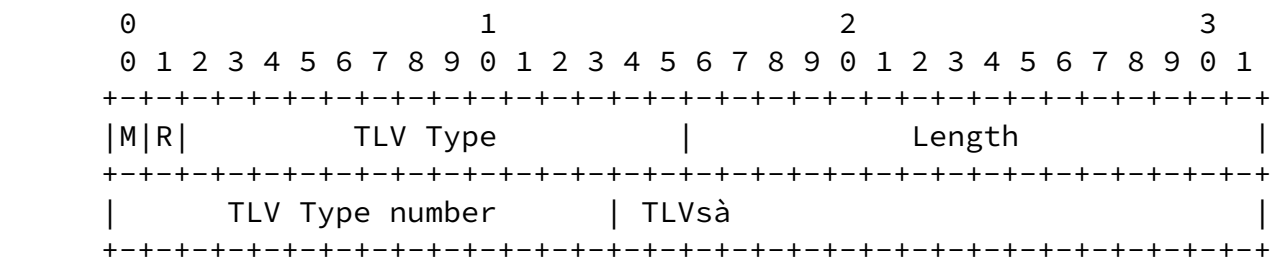
Status

The status field is two octets. Values include:

1 - Success
2 - - Failure

4.6. NAK TLV

The NAK TLV allows a peer to detect when TLVs that are not supported by the other peer. It is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

4 -

Length

<tbd>

TLV Type number.

The field contains TLV type that is not supported.

TLVs..

The field contains a list of optional TLVs. These could be used in future to send information on why the field was determined to

be unknown.

5. Security Considerations

5.1. Authentication and integrity protection

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

Note however, that [IEEE8021X] manufactures cleartext EAP Success

and EAP Failure messages, so that even though the Result TLV will be protected, this will be followed by a cleartext EAP Success or EAP Failure packet.

5.2. Method negotiation

If the peer does not support PEAP, or does not wish to utilize PEAP 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. Unauthentic 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 PEAP as an authentication method does not limit the potential secondary authentication methods. As a result, the only legitimate reason for a peer to NAK PEAP as an authentication method is that it does not support it. Where the additional security of PEAP is required, server implementations SHOULD respond to a NAK with an EAP-Failure, terminating the authentication conversation.

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 PEAP Part 2 conversation, and successfully conclude the conversation as described in Section 2.4.

PEAP "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 PEAP 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. The peer's authenticity is only proven after successful completion of the protected acknowledge exchange in PEAP part 2.

Some PEAP implementations may not be capable of removing TLS session cache entries established in PEAP Part 1 after an unsuccessful PEAP 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 PEAP Part 2 authentication or failed protected ack 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 protected acknowledge has been successfully exchanged. Failing to do this would enable a peer to gain access by completing PEAP Part 1, tearing down the connection, re-connecting and resuming PEAP 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 PEAP removes TLS session cache entries of peers failing PEAP Part 2 authentication, then it MAY skip the PEAP Part 2 conversation entirely after a successful session resumption, successfully terminating the PEAP conversation as described in Section 2.4.

5.4. Certificate revocation

Since the EAP server is on the Internet 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 Internet 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 Internet 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 Internet connectivity. The TLS Extensions [TLSEXT] 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 all TLS

---

implementations do not implement the TLS extensions, it may be necessary for the peer to wait to check for certificate revocation until after Internet 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, then the peer SHOULD disconnect.

5.5. Separation of the EAP server and the authenticator

As a result of a complete PEAP 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 PEAP 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 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 PEAP Part 1 and Part 2 Servers

The EAP server involved in PEAP Part 2 need not necessarily be the same as the EAP server involved in PEAP 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 PEAP 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 PEAP is identity protection, this is undesirable.

If the EAP method negotiated during PEAP Part 2 does not support mutual authentication, then if the Part 2 conversation is proxied to

another destination, the PEAP 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 PEAP 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 PEAP. As a result, it is unable to enforce policies requiring PEAP. Since some EAP methods require PEAP in order to generate keys or lessen security vulnerabilities, where such methods are in use, such a configuration may be unacceptable.

In summary, PEAP encrypting/decrypting gateway configurations are vulnerable to attack and SHOULD NOT be used. Instead, the entire PEAP connection SHOULD be proxied to the final destination, and the subsequently derived master session keys need to be transmitted back. This provides end to end protection of PEAP. The specification of this transit mechanism is outside the scope of this document, but mechanisms similar to [RFC2548] can be used. These steps protects 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, using a proxy solution without end to end protection of PEAP MAY be used. The PEAP 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 [RFC3162].

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 PEAP and the first portion of the PEAP 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 PEAP 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 PEAP 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, PEAP 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 PEAP conversation.

The protected identity or identities presented by the peer within PEAP Part 2 may not be identical to the cleartext identity presented in PEAP 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 PEAP Part 2, the peer may claim the identity of "fred@bigco.com". Thus, PEAP can provide protection for the user's identity, though not necessarily the destination realm, unless the PEAP Part 1 conversation terminates at the local authentication server.

As a result, PEAP implementations SHOULD NOT attempt to compare the Identities claimed with Parts 1 and 2 of the PEAP conversation. Similarly, if multiple Identities are claimed within PEAP 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 protection

If an EAP method protected by PEAP is also deployed without protection (from PEAP or IPSEC), and if the same credential is allowed in both cases, then a man-in-the-middle attack is possible. 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.

The compound binding draft [CompoundBinding] identifies a number of solutions to this attack.

The preferred solution is to deploy the authentication method with protection from PEAP or IPSEC. Protection can address the man-in-the-middle attack; and in addition can address EAP method and EAP protocol weaknesses listed in the abstract and introduction sections in this document.

Another solution is to use knowledge known only to the real peers to verify that there is no man-in-the-middle. A number of protocols derive keys for encryption, and these keys are not known to the man-in-the-middle. These keys can be used in the binding phase exchange described in compound binding [compoundbinding] draft to detect man-

in-the-middle. PEAP implementations MUST support the binding phase exchange using compound MACs as described in the section 4.2 of the compound binding draft[CompoundBinding].

Another solution is for EAP methods to securely signal to peers that they are inside the protected channel. This may require changes to the EAP protocol. In order to allow EAP methods to implement secure signaling, PEAP implementations SHOULD inform the EAP methods that they are being protected by PEAP.

6. IANA Considerations

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

There is one name space in EAP-TLV that require registration: TLV-Types.

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 registration requests where a Designated Expert should be consulted, the responsible IESG area director should appoint the Designated Expert. 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.

For registration requests requiring Expert Review, the EAP mailing list should be consulted. If the EAP mailing list is no longer operational, an alternative mailing list may be designated by the responsible IESG Area Director.

EAP-TLVs have a 14-bit field, of which 1-6 have been allocated.

7. Normative references

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

[RFC1570] Simpson, W., Editor, "PPP LCP Extensions", RFC 1570, January 1994.

[RFC1661] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[RFC1962] D. Rand. "The PPP Compression Control Protocol", RFC 1962, Novell, June 1996.

[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.

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

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

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

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

[TLSEXT] Blake-Wilson, S., et al. "TLS Extensions", Internet draft (work in progress), draft-ietf-tls-extensions-06.txt, Feb 2003.

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

[CompoundBinding] Puthenkulam, J., Lortz, V., Palekar, A., Simon, D., "The Compound Authentication Binding Problem", March 2003; draft-puthenkulam-eap-binding-02.txt.

8. Informative references

[RFC2419] Sklower, K., Meyer, G., "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", RFC2548, March 1999.

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

[RFC3078] Pall, G., Zorn, G., "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.

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

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

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

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

9. Appendix A - Examples

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

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- EAP-Request/
```



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

EAP-Response/
EAP-Type=PEAP, V=1
(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-Response/
EAP-Type=PEAP ->

TLS channel established
(messages sent within the TLS channel)

                         <- EAP-Request/
                          Identity
EAP-Response/
Identity (MyID2) ->
                         <- EAP-Request/
                          EAP-Type=X

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

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

Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
```


methods=1, Result TLV (Success), Method_Identity_TLV (EAP-Type=X, EAP-Type-Version=0, keylengthusedforderivation,

```
ClientIdentityLength= sizeof(MyID2), MyID2, ServerIdentityLength=0,
Media-type=19), Method_Identity_TLV (EAP-Type=PEAP, EAP-Type-
Version=2, keylengthusedforderivation, ClientIdentityLength=
sizeof(MyID1), MyID1, ServerIdentityLength=0, Media-type=19),
CompoundMAC (over entire EAP TLV inside the tunnel including EAP-
header))
EAP-Response/
EAP-Type=EAP-TLV
Result=Success
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=1, Result TLV (Success), Method_Identity_TLV (EAP-Type=X,
EAP-Type-Version=0, keylengthusedforderivation,
ClientIdentityLength= sizeof(MyID2), MyID2, ServerIdentityLength=0,
Media-type=19), Method_Identity_TLV (EAP-Type=PEAP, EAP-Type-
Version=2, keylengthusedforderivation, ClientIdentityLength=
sizeof(MyID1), MyID1, ServerIdentityLength=0, Media-type=19),
CompoundMAC (over entire EAP TLV inside the tunnel including EAP-
header))

->

TLS channel torn down
(messages sent in cleartext)

                        <- EAP-Success
```

Where all peers are known to support PEAP, a non-certificate authentication is desired for the client and the PEAP 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=1
                        (PEAP Start, S bit set)

EAP-Response/
EAP-Type=PEAP, V=1
(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-Response/
EAP-Type=PEAP, V=2 ->

TLS channel established
(messages sent within the TLS channel)

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

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

Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))


EAP-Response/
EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
```

EAP-header))

TLS channel torn down
(messages sent in cleartext)

<- EAP-Success

Where all peers are known to support PEAP, where client certificate

---

authentication is desired and the PEAP 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)
EAP-Response/
EAP-Type=PEAP, V=2 ->

TLS channel established
(messages sent within the TLS channel)

                        <- EAP-Request/
```

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


EAP-Type=PEAP, V=2 ->
                          <- EAP-Request/
                             EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))
EAP-Response/
EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each successful EAP-Type inside PEAP), Method_Identity_TLV (for
PEAP), CompoundMAC (over entire EAP TLV packet inside the tunnel
including EAP-header))

TLS channel torn down
```

```
(messages sent in cleartext)

                        <- EAP-Success
```

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

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

TLS channel established
(messages sent within the TLS channel)

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

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



```
                        <- EAP-Request/
                           EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))
EAP-Response/
EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Success), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
```

```
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))

TLS channel torn down
(messages sent in cleartext)

                        <- EAP-Success
```

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

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- EAP-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-Response/
```

```
EAP-Type=PEAP, V=2 ->

TLS channel established
(messages sent within the TLS channel)

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

EAP-Type=X or NAK ->

                        <- EAP-Request/
                         EAP-Type=X
EAP-Response/
EAP-Type=X  ->
                         <- EAP-Request/
                            EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Failure), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))
// Compound MAC calculated using TLS key material only.
EAP-Response/
EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=<number>, Result TLV (Failure), Method_Identity_TLV (for
each EAP-Type inside PEAP), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))

Result=Failure  ->

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

                         <- EAP-Failure
```

In the case where server authentication is unsuccessful in PEAP 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-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) ->
                          <- EAP-Failure
                          (TLS session cache entry flushed)
```

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

```
Authenticating Peer     Authenticator
-------------------     -------------
                          <- EAP-Request/
                          Identity
```

```
EAP-Response/
Identity (MyID) ->
                        <- EAP-Request/
                        EAP-Type=PEAP,V=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
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=0, Result TLV (Success), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header))
// Compound MAC calculated using TLS keys since there were no inner
EAP methods.

EAP-Response/
EAP-Type=EAP-TLV
Crypto-Binding-TLV=(Version=0, Nounce, Number of inner EAP
methods=0, Result TLV (Success), Method_Identity_TLV (for PEAP),
CompoundMAC (over entire EAP TLV packet inside the tunnel including
EAP-header)) .
->
TLS channel torn down
(messages sent in cleartext)

                        <- EAP-Success
```

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

```
Authenticating Peer     Authenticator
```

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

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
                       (TLS 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
```


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 session is being resumed, but the authentication is unsuccessful, 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
                        (TLS Start)
EAP-Response/
EAP-Type=PEAP, V=0
(TLS client_hello)->
                        <- EAP-Request/
                        EAP-Type=PEAP, V=0
                        (TLS server_hello,
                         TLS change_cipher_spec,
```

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

                              <- EAP-Failure
```

10. Acknowledgments and Contributions

Thanks to Jan-Ove Larsson, Magnus Nystrom of RSA Security; Bernard Aboba, Vivek Kamath, Stephen Bensley, Narendra Gidwani of Microsoft; Joe Salowey, Hao Zhou, Ilan Frenkel, Nancy Cam-Winget of Cisco; Hakan Andersson of RSA; 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 he was 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
USA

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

Simon Josefsson
Drottningholmsv□□gen 70
112 42 Stockholm
Sweden

Phone: +46 8 619 04 22
EMail: jas@extundo.com

11. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of

claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary

rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

12. Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved. This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Expiration Date

This memo is filed as <draft-josefsson-pppext-eap-tls-eap-06.txt>, and expires after six months.