PPPEXT Working Group
INTERNET-DRAFT
Category: Standards Track
<draft-josefsson-pppext-eap-tls-eap-05.txt>
September 2002

H. Andersson
S. Josefsson
RSA Security
Glen Zorn
Cisco
Dan Simon
Ashwin Palekar
Microsoft

Protected EAP Protocol (PEAP)

Copyright Notice

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 have become apparent. These include lack of protection of the user identity or the EAP negotiation; no standardized mechanism for key exchange; no built-in support for fragmentation and reassembly; and lack of support for fast reconnect.

By wrapping the EAP protocol within TLS, Protected EAP (PEAP) addresses these deficiencies. Any EAP method running within PEAP is provided with built-in support for key exchange, session resumption and 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.

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). As a result, the NAS acts as a "passthrough", and need not understand specific EAP methods.

Figure 1 describes the relationship between the EAP peer, NAS and backend authentication server. As described in the figure, the EAP conversation "passes through" the NAS on its way between the client and the backend authentication server. While the authentication conversation is between the EAP peer and backend authentication server, the NAS and backend authentication server need to have established trust for the conversation to proceed.

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

As a result, the NAS does not have knowledge of the TLS master secret derived between the EAP Peer and the backend authentication server, and cannot decrypt the PEAP conversation. In order to providing keying material for link-layer ciphersuites however, the NAS does obtain the master session keys, which are derived from the TLS master secret via a one-way function.

1.1. EAP Issues

With the increasing adoption of EAP, a number of deficiencies have become apparent. Since the EAP method negotiation is unprotected, where an attacker can easily access the medium (such as on a wireless network or where EAP is run over IP), it is possible for an attacker to inject packets in order to cause the negotiation of a method with lesser security. Denial of service attacks are also possible. By protecting the EAP negotiation within a TLS channel, PEAP addresses this issue.

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, PEAP provies support for identity protection.

```
+-+-+-+-+-+               +-+-+-+-+-+
|         |               |         |
|         |               |         |
| Cipher- |               | Cipher- |
| Suite   |               | Suite   |
|         |               |         |
+-+-+-+-+-+               +-+-+-+-+-+
    ^                         ^
    |                         |
    |                         |
    |                         |
    V                         V
+-+-+-+-+-+               +-+-+-+-+-+  Trust +-+-+-+-+-+
|         |  EAP          |         |<======>|         |
|         |  Conversation |         |        |         |
|         |<================================>| Backend |
| Client  |  (over PPP,   |   NAS   |        |  Server |
|         |  802.11,etc.) |         |<=======|         |
|         |               |         |  Keys  |         |
|         |               |         |        |         |
+-+-+-+-+-+               +-+-+-+-+-+        +-+-+-+-+-+
    ^                                            ^
    |                                            |
    | EAP API                                    | EAP API
    |                                            |
    V                                            V
+-+-+-+-+-+                                  +-+-+-+-+-+
|         |                                  |         |
|         |                                  |         |
|  EAP    |                                  |  EAP    |
|  Method |                                  |  Method |
|         |                                  |         |
+-+-+-+-+-+                                  +-+-+-+-+-+
```

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

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.

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.

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 developers from taking on the difficult (and error prone) task of designing a key hierarchy for each method.

1.2. 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].

1.3. 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.

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.

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 that are subsequently used in generation of the transient session keys for authentication, encryption, and IV-generation. So that the master session keys are usable with any link layer ciphersuite, they are longer than is necessary, and are truncated to fit.

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.

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 the authenticator and the peer negotiating EAP. 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 userId.

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. 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, so that only an EAP-Success or EAP-Failure message is sent inside the TLS channel. 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, after the server_finished message is sent, and before PEAP part 2, the server sends 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. 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 be able to negotiate the following TLS ciphersuites:

```
    TLS_RSA_WITH_RC4_128_MD5
    TLS_RSA_WITH_RC4_128_SHA
```

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 preceeding 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.

2.1.1. Forging of Success and Failure packets

Within 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 encrypting and integrity protecting the EAP conversation within a TLS channel, PEAP provides protection against these attacks. Since the EAP Server MUST authenticate itself to the EAP Peer in PEAP Part 1, once the TLS channel has been brought up, EAP Success or Failure packets should be sent down the encrypted channel, rather than being sent in cleartext. As a result, once PEAP has been selected as the authentication method, and the PEAP conversation has begun, a peer receiving cleartext Success or Failure packets MUST silently discard them.

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 the TLS session in Part 1 is successful. 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.

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.

As with a normal EAP conversation described in [RFC2284], an EAP conversation encapsulated within the TLS channel as within PEAP Part 2 continues until the EAP server sends an EAP-Failure or EAP-Success. The receipt of an EAP-Failure or EAP-Success within the TLS protected channel results in a shutdown of the TLS channel by the peer and EAP server. The EAP-Failure or EAP-Success packet sent within the TLS channel is protected from snooping or packet modification, and as a result, while an EAP server MAY send an additional EAP-Failure or EAP-Success message in cleartext, this is not required, since it adds another round-trip. As described in [RFC2869], a RADIUS Access-Accept or Access-Reject packet need not contain an EAP-Message attribute, since the NAS determines the success of the conversation from the RADIUS message (Accept/Reject), not the encapsulated EAP-Message attribute.

2.3. Version negotiation

PEAP packets contain a two 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 1. 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 a lower version number, indicating the highest supported version number.

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

[5] If the EAP server does not support the version number proposed by the EAP client, it responds with an EAP-Failure sent in the clear.

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.

2.4. 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.

2.5. 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.

2.6. 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 tunnel 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 tunnel 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.

2.7. 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.

2.8. 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].

For both peer and EAP server, the derivation of master session keys proceeds as follows:

[1] Given the master key negotiated by the TLS handshake, the pseudorandom function (PRF) defined in the specification for the version of TLS in use, and the value random defined as the concatenation of the handshake message fields client_hello.random and server_hello.random (in that order), the value PRF(master key, "client PEAP encryption", random) is computed up to 128 bytes, and the value PRF("", "client PEAP encryption", random) is computed up to 64 bytes (where "" is an empty string).

[2] The peer master session encryption key (the one used for encrypting data from peer to EAP server) is obtained by truncating to the correct length the first 32 bytes of these two PRF output strings.

[3] The EAP server master session encryption key (the one used to encrypting data from EAP server to peer), if different from the client master session encryption key, is obtained by truncating to the correct length the second 32 bytes of this same PRF output string.

[4] The peer master session authentication key (the one used for computing MACs for messages from peer to EAP server), if used, is obtained by truncating to the correct length the third 32 bytes of this same PRF output string.

[5] The EAP server master session authentication key (the one used for computing MACs for messages from EAP server to peer), if used, and if different from the peer master session authentication key, is obtained by truncating to the correct length the fourth 32 bytes of this same PRF output string.

[6] The peer master session initialization vector (IV), used for messages from peer to EAP server, is obtained by truncating to the

cipher’s block size the first 32 bytes of the second PRF output string mentioned above.

[7] Finally, the EAP server master session initialization vector (IV), used for messages from peer to EAP server, is obtained by truncating to the cipher’s block size the second 32 bytes of this second PRF output.

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]. Additional keys or other non-secret values (such as IVs) can be obtained as needed by extending the outputs of the PRF beyond 128 bytes and 64 bytes, respectively.

2.9. 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, PEAP does not support secure negotiation of link layer ciphersuites. While such a negotiation is preferable from a security perspective, it is in practice difficult to integrate with existing PPP and IEEE 802.11 link layer security negotiation, as well as with backend authentication servers.

Depending on the link layer technology in use, the link layer security negotiation will occur at different stages in the connection process. In IEEE 802.11, selection of the link layer security mechanism occurs via the association/re-association messages, prior to authentication. In contrast, within PPP, link layer security negotiation occurs in ECP [RFC1968], which occurs after authentication.

As a result, within IEEE 802.11, by the time that PEAP is invoked, the link layer security technology has already been selected. Thus if PEAP were to support a protected link layer ciphersuite negotiation whose conclusion disagreed with the IEEE 802.11 negotiation, a reassociation (and additional authentication!) would be required to synchronize the results of the two negotiations. Within PPP, it is conceivable that the results of a PEAP secure link layer security negotiation could be subsequently reflected in the ECP negotiation.

There are other issues as well. While link layer ciphersuite negotiation occurs between the peer and the NAS, the EAP conversation occurs between the peer and the EAP server. Since the EAP server may not be aware of the link layer ciphersuites supported by the NAS, it is conceivable that the NAS and peer can negotiate a link layer ciphersuite that is not supported by the NAS. To address this issue, it would be necessary for the NAS to send the list of supported link layer ciphersuites to the backend authentication server, and have the backend security server respond with a list of acceptable choices. However, when used with technologies such as IEEE 802.11 where link layer security technology selection occurs prior to authentication, multiple association/reassociation exchanges might be required to synchronize the negotiations, resulting in extended connectivity loss.

The situation typically cannot be addressed merely by omitting IEEE 802.11 link layer security negotiation. Unless all users on the AP are to be authenticated with PEAP or an alternative EAP method providing secure link layer security negotiation, then omitting IEEE 802.11 security negotiation would leave some users without the ability to negotiate security mechanisms.

For these reasons, protected negotiation of link layer ciphersuites within PEAP is considered impractical and is omitted from this specification.

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 5
+-+-+-+-+-+-+
|L M S R 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
+-+-+
|R 1|
+-+-+
```

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 5
   +-+-+-+-+-+-+
   |L M S R 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
+-+-+
|R 1|
+-+-+
```

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.

```
 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

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

```
 0 1 2 3 4 5
+-+-+-+-+-+-+
|L M S R 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
+-+-+
|R 1|
+-+-+
```

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. Security Considerations

4.1. 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.

4.2. 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 entirely, and immediately send an EAP-Success message within the TLS channel established via session resumption.

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 by successful completion of the PEAP Part 2 authentication.

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 MUST use other means than successful TLS resumption as the indicator of whether the client is authenticated or not. Failing to do this would enable a peer to gain access by completing PEAP Part 1, tearing down the connection and re-connect and resume PEAP Part 1 thereby proving herself authenticated. Thus, TLS resumption MUST only be used as an indicator of whether the client is authenticated or not 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 SHOULD skip the PEAP Part 2 conversation entirely after a successful session resumption, immediately sending an EAP-Success message within the TLS channel.

4.3. 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 tunnel using PPTP or L2TP, the peer will typically already have Internet connectivity established at the time of tunnel 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.

4.4. 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 tunnel 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 tunnel server that needs to use the key. The specification of this transit mechanism is outside the scope of this document.

4.5. 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].

4.6. 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 tunnel 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. 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-02.txt, December 2001.

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

6. 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).

Appendix A - Examples

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

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

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

TLS channel established
(messages sent within the TLS channel)

EAP-Response/
EAP-Type=PEAP ->

                       <- 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-Success

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

Where all peers are known to support PEAP, 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
                        (PEAP Start, S bit set)

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

TLS channel established
```

```
(messages sent within the TLS channel)

EAP-Response/
EAP-Type=PEAP ->

                       <- 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-Success

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

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
                        (PEAP Start, S bit set)

EAP-Response/
EAP-Type=PEAP
(TLS client_hello)->
                        <- EAP-Request/
                        EAP-Type=PEAP
                        (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 ->
                        <- EAP-Request/
                           EAP-Type=PEAP
                        (Fragment 2: M bit set)
EAP-Response/
EAP-Type=PEAP ->
                        <- EAP-Request/
                        EAP-Type=PEAP
                        (Fragment 3)
EAP-Response/
EAP-Type=PEAP
([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
EAP-Response/
EAP-Type=PEAP
(Fragment 2)->
                       <- EAP-Request/
                        EAP-Type=PEAP
                        (TLS change_cipher_spec,
                         TLS finished)

TLS channel established
(messages sent within the TLS channel)

EAP-Response/
EAP-Type=PEAP ->

                       <- 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-Success

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

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

TLS channel established
(messages sent within the TLS channel)

EAP-Response/
EAP-Type=PEAP ->

                       <- 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-Failure
                        (TLS session cache entry flushed)

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

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

                        <- EAP-Request/
                        EAP-Type=PEAP
PPP EAP-Response/
EAP-Type=PEAP
(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) ->
                        <- PPP EAP-Request/
                        EAP-Request/
                        EAP-Type=PEAP
                        (PEAP Start)
EAP-Response/
EAP-Type=PEAP
(TLS client_hello)->
                        <- EAP-Request/
                        EAP-Type=PEAP
                        (TLS server_hello,
                        TLS change_cipher_spec
                        TLS finished)
EAP-Response/
EAP-Type=PEAP
(TLS change_cipher_spec,
 TLS finished) ->
                        <- 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
                        (TLS Start)
EAP-Response/
EAP-Type=PEAP
(TLS client_hello) ->
                        <- EAP-Request/
                        EAP-Type=PEAP
                        (TLS server_hello,
                         TLS change_cipher_spec,
                         TLS finished)
EAP-Response/
EAP-Type=PEAP
(TLS change_cipher_spec,
 TLS finished) ->
                        <- EAP-Request
                        EAP-Type=PEAP
                        (TLS Alert message)
EAP-Response
EAP-Type=PEAP ->
                         <- 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
                        (TLS Start)
```

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

Acknowledgments

Thanks to Jan-Ove Larsson and Magnus Nystrom of RSA Security, and Narendra Gidwani and Bernard Aboba of Microsoft for useful discussions of this problem space.

Author Addresses

Hakan Andersson
RSA Security
Box 107 04
SE-121 29 Stockholm
Sweden

Phone: +46 8 725 9758
Fax: +46 8 649 4970
EMail: handersson@rsasecurity.com

Simon Josefsson
RSA Security
Box 107 04
SE-121 29 Stockholm
Sweden

Phone: +46 8 725 0914
Fax: +46 8 649 4970
EMail: sjosefsson@rsasecurity.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

Dan Simon
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

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

Ashwin Palekar

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

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

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

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

Full Copyright Statement

Expiration Date

This memo is filed as <draft-josefsson-pppext-eap-tls-eap-05.txt>, and expires March 19, 2003.