Network Working Group
Requests for Commments: 2716
Category: Experimental

B. Aboba
D. Simon
Microsoft

October 1999

PPP EAP TLS Authentication Protocol

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

1. Abstract

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP also defines an extensible Link Control Protocol (LCP), which can be used to negotiate authentication methods, as well as an Encryption Control Protocol (ECP), used to negotiate data encryption over PPP links, and a Compression Control Protocol (CCP), used to negotiate compression methods. The Extensible Authentication Protocol (EAP) is a PPP extension that provides support for additional authentication methods within PPP.

Transport Level Security (TLS) provides for mutual authentication, integrity-protected ciphersuite negotiation and key exchange between two endpoints. This document describes how EAP-TLS, which includes support for fragmentation and reassembly, provides for these TLS mechanisms within EAP.

2. Introduction

The Extensible Authentication Protocol (EAP), described in [5], provides a standard mechanism for support of additional authentication methods within PPP. 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. To date however, EAP methods such as [6] have focussed on authenticating a client to a server.

However, it may be desirable to support mutual authentication, and since PPP encryption protocols such as [9] and [10] assume existence of a session key, it is useful to have a mechanism for session key establishment. Since design of secure key management protocols is non-trivial, it is desirable to avoid creating new mechanisms for this. The EAP protocol described in this document allows a PPP peer to take advantage of the protected ciphersuite negotiation, mutual authentication and key management capabilities of the TLS protocol, described in [12].

2.1. 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 [11].

3. Protocol overview

3.1. Overview of the EAP-TLS conversation

As described in [5], the EAP-TLS conversation will typically begin with the authenticator and the peer negotiating EAP. The authenticator will then 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.

From this point forward, while nominally the EAP conversation occurs between the PPP 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 RADIUS server or backend security 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, the EAP server MUST respond with an EAP-TLS/Start packet, which is an EAP-Request packet with EAP-Type=EAP-TLS, the Start (S) bit set, and no data. The EAP-TLS conversation will then begin, with the peer sending an EAP-Response packet with EAP-Type=EAP-TLS. The data field of that 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 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=EAP-TLS. 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. The server_hello handshake message contains a TLS version number, another random number, a sessionId, and a ciphersuite. The version offered by the server MUST correspond to TLS v1.0 or later.

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

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. While this model was developed for use with HTTP authentication, it may also have application to PPP authentication (e.g. multilink).

As a result, it is left up to the peer whether to attempt to continue a previous session, thus shortening the TLS 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 authenticator reside on the same device, then 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 RADIUS 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. 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 certificate_request message is included when the server desires the client to authenticate itself via public key. While the EAP server SHOULD require client authentication, this is not a requirement, since it may be possible that the server will require that the peer authenticate via some other means.

The peer MUST respond to the EAP-Request with an EAP-Response packet of EAP-Type=EAP-TLS. 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. If the EAP server sent a certificate_request message in the preceding EAP-Request packet, then the peer MUST send, in addition, certificate and certificate_verify handshake messages. The former contains a certificate for the peer's signature public key, while the latter contains the peer's signed authentication response to the EAP server. After receiving this packet, the EAP server will verify the peer's certificate and digital signature, if requested.

If the peer's authentication is unsuccessful, the EAP server SHOULD send an EAP-Request packet with EAP-Type=EAP-TLS, encapsulating a TLS record containing the appropriate TLS alert message. The EAP server SHOULD send a TLS alert message rather immediately terminating the conversation so as to allow the peer to inform the user of the cause of the failure and possibly allow for a restart of the conversation.

To ensure that the peer receives the TLS alert message, the EAP server MUST wait for the peer to reply with an EAP-Response packet. The EAP-Response packet sent by the peer MAY encapsulate a TLS client_hello handshake message, in which case the EAP server MAY allow the EAP-TLS conversation to be restarted, or it MAY contain an EAP-Response packet with EAP-Type=EAP-TLS and no data, in which case the EAP-Server MUST send an EAP-Failure packet, and terminate the conversation. It is up to the EAP server whether to allow restarts, and if so, how many times the conversation can be restarted. An EAP Server implementing restart capability SHOULD impose a limit on the number of restarts, so as to protect against denial of service attacks.

If the peers authenticates successfully, the EAP server MUST respond with an EAP-Request packet with EAP-Type=EAP-TLS, which includes, in the case of a new TLS session, one or more TLS records containing TLS change_cipher_spec and finished handshke 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=EAP-TLS 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=EAP-TLS, and no data. The EAP-Server then MUST respond with an EAP-Success message.

3.2. Retry behavior

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

For example, if the initial EAP-TLS 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 EAP-TLS start packet would be resent by the EAP server. Once the peer received the EAP-TLS 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 EAP-TLS 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.

3.3. 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 EAP-TLS 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 anwhere 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 [2]. While this is desirable, there may be cases in which multilink or the MRRU LCP option cannot be negotiated. As a result, an EAP-TLS implementation MUST provide its own support for fragmentation and reassembly.

Since EAP is a simple 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.

EAP-TLS fragmentation support is provided through addition of a flags octet 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 EAP-TLS 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 EAP-TLS 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 an EAP-TLS peer receives an EAP-Request packet with the M bit set, it MUST respond with an EAP-Response with EAP-Type=EAP-TLS 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-Reponse. 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=EAP-TLS 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 use 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-Reponse.

3.4. Identity verification

As part of the TLS negotiation, the server presents a certificate to the peer, and if mutual authentication is requested, the peer presents a certificate to the server.

Note that since the peer has made a claim of identity in the EAP-Response/Identity (MyID) packet, the EAP server SHOULD verify that the claimed identity corresponds to the certificate presented by the

peer. Typically this will be accomplished either by placing the userId within the peer certificate, or by providing a mapping between the peer certificate and the userId using a directory service.

Similarly, the peer MUST 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 that 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.

3.5. Key derivation

Since the normal TLS keys are used in the handshake, and therefore should not be used in a different context, new encryption keys must be derived from the TLS master secret for use with PPP encryption. For both peer and EAP server, the derivation proceeds as follows: given the master secret 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 secret, "client EAP encryption", random) is computed up to 128 bytes, and the value PRF("", "client EAP encryption", random) is computed up to 64 bytes (where "" is an empty string). The peer 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 the first PRF of these two output strings. TheEAP server encryption key (the one used for encrypting data from EAP server to peer), if different from the client encryption key, is obtained by truncating to the correct length the second 32 bytes of this same PRF output string. The client 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. The EAP server authentication key (the one used for computing MACs for messages from EAP server to peer), if used, and if different from the peer authentication key, is obtained by truncating to the correct length the fourth 32 bytes of this same PRF output string. The peer initialization vector (IV), used for messages from peer to EAP server if a block cipher has been specified, is obtained by truncating to the cipher's block size the first 32 bytes of the second PRF output string mentioned above. Finally, the server initialization vector (IV), used for messages from peer to EAP server

if a block cipher has been specified, is obtained by truncating to the cipher's block size the second 32 bytes of this second PRF output.

The use of these encryption and authentication keys is specific to the PPP encryption mechanism used, such as those defined in [9] and [10]. Additional keys or other non-secret values (such as IVs) can be obtained as needed for future PPP encryption methods by extending the outputs of the PRF beyond 128 bytes and 64 bytes, respectively.

3.6. ECP negotiation

Since TLS supports ciphersuite negotiation, peers completing the TLS negotiation will also have selected a ciphersuite, which includes key strength, encryption and hashing methods. As a result, a subsequent Encryption Control Protocol (ECP) conversation, if it occurs, has a predetermined result.

In order to ensure agreement between the EAP-TLS ciphersuite negotiation and the subsequent ECP negotiation (described in [6]), during ECP negotiation the PPP peer MUST offer only the ciphersuite negotiated inEAP-TLS. This ensures that the PPP authenticator MUST accept the EAP-TLS negotiated ciphersuite in order for the onversation to proceed. Should the authenticator not accept the EAP-TLS negotiated ciphersuite, then the peer MUST send an LCP terminate and disconnect.

Please note that it cannot be assumed that the PPP authenticator and EAP server are located on the same machine or that the authenticator understands the EAP-TLS conversation that has passed through it. Thus if the peer offers a ciphersuite other than the one negotiated in EAP-TLS there is no way for the authenticator to know how to respond correctly.

3.7. CCP negotiation

TLS as described in [12] supports compression as well as ciphersuite negotiation. However, TLS only provides support for a limited number of compression types which do not overlap with the compression types used in PPP. As a result, during the EAP-TLS conversation the EAP endpoints MUST NOT request or negotiate compression. Instead, the PPP Compression Control Protocol (CCP), described in [13] should be used to negotiate the desired compression scheme.

3.8. Examples

In the case where the EAP-TLS mutual authentication is successful, the conversation will appear as follows:

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- PPP LCP Request-EAP
                        auth
PPP LCP ACK-EAP
auth ->
                        <- PPP EAP-Request/
                        Identity
PPP EAP-Response/
Identity (MyID) ->
                        <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
                        (TLS Start)
PPP EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                        <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
                        (TLS server_hello,
                         TLS certificate,
                  [TLS server_key_exchange,]
                  [TLS certificate_request,]
                      TLS server_hello_done)
PPP EAP-Response/
EAP-Type=EAP-TLS
(TLS certificate,
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->
                        <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
                        (TLS change_cipher_spec,
                         TLS finished)
PPP EAP-Response/
EAP-Type=EAP-TLS ->
                        <- PPP EAP-Success
PPP Authentication
Phase complete,
NCP Phase starts

ECP negotiation
CCP negotiation
```

In the case where the EAP-TLS mutual authentication is successful, and fragmentation is required, the conversation will appear as follows:

```
Authenticating Peer     Authenticator
-------------------     -------------
                        <- PPP LCP Request-EAP
                        auth
PPP LCP ACK-EAP
auth ->
                        <- PPP EAP-Request/
                        Identity
PPP EAP-Response/
Identity (MyID) ->
                        <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
                        (TLS Start, S bit set)
PPP EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                        <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                          (TLS server_hello,
                             TLS certificate,
                    [TLS server_key_exchange,]
                    [TLS certificate_request,]
                        TLS server_hello_done)
                   (Fragment 1: L, M bits set)
PPP EAP-Response/
EAP-Type=EAP-TLS ->
                        <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                        (Fragment 2: M bit set)
PPP EAP-Response/
EAP-Type=EAP-TLS ->
                        <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
                        (Fragment 3)
PPP EAP-Response/
EAP-Type=EAP-TLS
(TLS certificate,
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS inished)(Fragment 1:
 L, M bits set)->
                         <- PPP EAP-Request/
                        EAP-Type=EAP-TLS
```

```
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (Fragment 2)->
                           <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS change_cipher_spec,
                             TLS finished)
   PPP EAP-Response/
   EAP-Type=EAP-TLS ->
                            <- PPP EAP-Success
   PPP Authentication
   Phase complete,
   NCP Phase starts

   ECP negotiation
   CCP negotiation
```

In the case where 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
   -------------------     -------------
                           <- PPP LCP Request-EAP
                           auth
   PPP LCP ACK-EAP
   auth ->
                           <- PPP EAP-Request/
                           Identity
   PPP EAP-Response/
   Identity (MyID) ->
                           <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS Start)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello)->
                           <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                            TLS certificate,
                     [TLS server_key_exchange,]
                           TLS certificate_request,
                           TLS server_hello_done)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS certificate,
    TLS client_key_exchange,
```

```
    TLS certificate_verify,
    TLS change_cipher_spec,
    TLS finished) ->
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS change_cipher_spec,
                            TLS finished)
   PPP EAP-Response/
   EAP-Type=EAP-TLS ->
                            <- PPP EAP-Request
                            EAP-Type=EAP-TLS
                            (TLS Alert message)
   PPP EAP-Response/
   EAP-Type=EAP-TLS ->
                            <- PPP EAP-Failure
                            (User Disconnected)
```

In the case where server authentication is unsuccessful, the conversation will appear as follows:

```
   Authenticating Peer      Authenticator
   -------------------      -------------
                            <- PPP LCP Request-EAP
                            auth
   PPP LCP ACK-EAP
   auth ->
                            <- PPP EAP-Request/
                            Identity
   PPP EAP-Response/
   Identity (MyID) ->
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS Start)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
    (TLS client_hello)->
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS server_hello,
                             TLS certificate,
                         [TLS server_key_exchange,]
                         [TLS certificate_request,]
                          TLS server_hello_done)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
    (TLS certificate,
    TLS client_key_exchange,
   [TLS certificate_verify,]
```

```
    TLS change_cipher_spec,
    TLS finished) ->
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS change_cipher_spec,
                             TLS finished)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS change_cipher_spec,
   TLS finished)
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS Alert message) ->
                            <- PPP EAP-Failure
                            (User Disconnected)
```

In the case where a previously established session is being resumed, and both sides authenticate successfully, the conversation will appear as follows:

```
   Authenticating Peer      Authenticator
   -------------------      -------------
                            <- PPP LCP Request-EAP
                            auth
   PPP LCP ACK-EAP
   auth ->
                            <- PPP EAP-Request/
                            Identity
   PPP EAP-Response/
   Identity (MyID) ->
                            <- PPP EAP-Request/
                            EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS Start)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello)->
                            <- PPP EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS server_hello,
                            TLS change_cipher_spec
                            TLS finished)
```

```
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS change_cipher_spec,
    TLS finished) ->
                            <- PPP EAP-Success
   PPP Authentication
   Phase complete,
   NCP Phase starts

   ECP negotiation

   CCP negotiation
```

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
   -------------------     -------------
                           <- PPP LCP Request-EAP
                           auth
   PPP LCP ACK-EAP
   auth ->
                           <- PPP EAP-Request/
                           Identity
   PPP EAP-Response/
   Identity (MyID) ->
                           <- PPP EAP-Request/
                           EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS Start)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello) ->
                           <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                            TLS change_cipher_spec,
                            TLS finished)
   PPP EA-Response/
   EAP-Type=EAP-TLS
   (TLS change_cipher_spec,
    TLS finished) ->
                           <- PPP EAP-Request
                           EAP-Type=EAP-TLS
                           (TLS Alert message)
```

```
   PPP EAP-Response
   EAP-Type=EAP-TLS ->
                            <- PPP EAP-Failure
                            (User Disconnected)
```

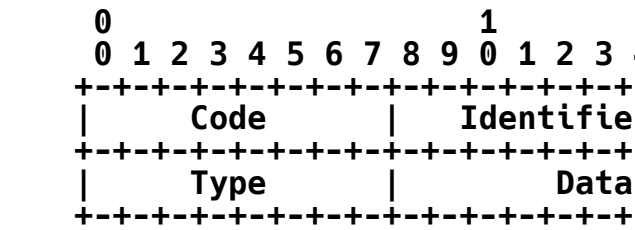
In the case where a previously established session is being resumed, and the server authentication is unsuccessful, the conversation will appear as follows:

```
   Authenticating Peer     Authenticator
   -------------------     -------------
                           <- PPP LCP Request-EAP
                           auth
   PPP LCP ACK-EAP
   auth ->
                           <- PPP EAP-Request/
                           Identity
   PPP EAP-Response/
   Identity (MyID) ->
                           <- PPP EAP-Request/
                           EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS Start)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello)->
                           <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                            TLS change_cipher_spec,
                            TLS finished)
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS change_cipher_spec,
   TLS finished)
                           <- PPP EAP-Request/
                           EAP-Type=EAP-TLS
   PPP EAP-Response/
   EAP-Type=EAP-TLS
   (TLS Alert message) ->
                           <- PPP EAP-Failure
                           (User Disconnected)
```

4. Detailed description of the EAP-TLS protocol

4.1. PPP EAP TLS Packet Format

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



Code

1 - Request
2 - Response

Identifier

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

Length

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

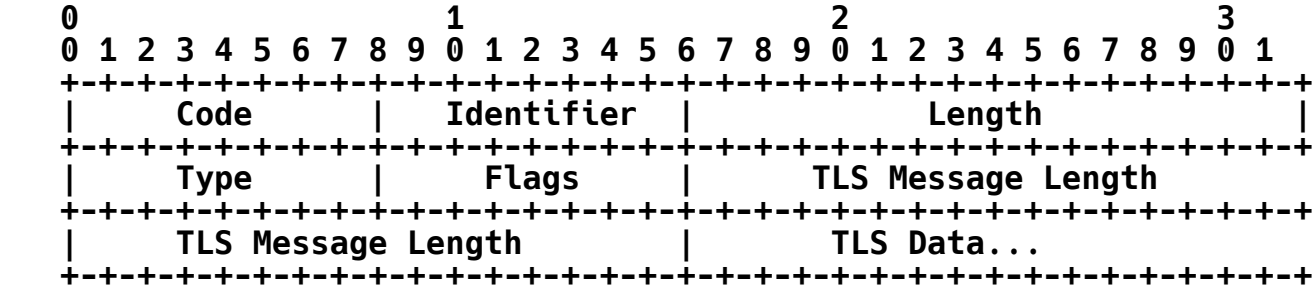
13 - EAP TLS

Data

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

4.2. PPP EAP TLS Request Packet

A summary of the PPP EAP TLS 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.

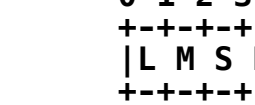
Length

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

Type

13 - EAP TLS

Flags



L = Length included
M = More fragments
S = EAP-TLS start
R = Reserved

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 (EAP-TLS start) is set in an EAP-TLS Start message. This differentiates the EAP-TLS Start message from a fragment acknowledgement.

TLS Message Length

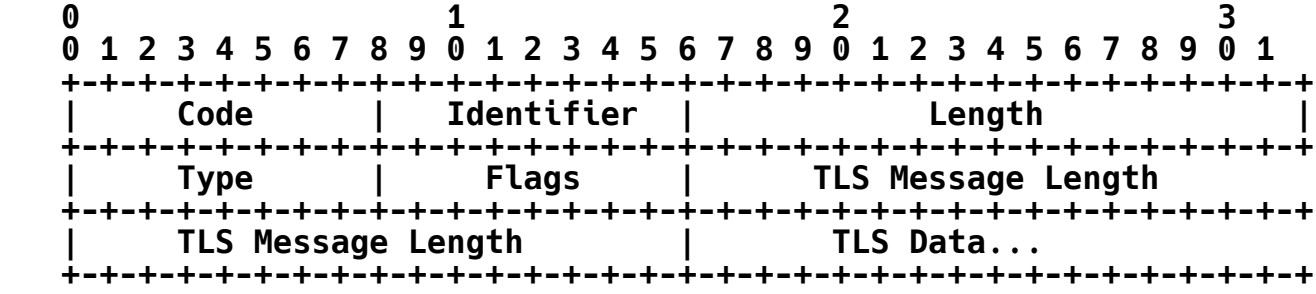
The TLS Message Length field is four octets, and is present only if the L bit is set. This field provides the total length of the TLS message or set of messages that is being fragmented.

TLS data

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

4.3. PPP EAP TLS Response Packet

A summary of the PPP EAP TLS 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, Identifir, Length, Type, and TLS data fields.

Type

13 - EAP TLS

Flags

```
0 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
|L M S R R R R R|
+-+-+-+-+-+-+-+-+
```

L = Length included
M = More fragments
S = EAP-TLS start
R = Reserved

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 (EAP-TLS start) is set in an EAP-TLS Start message. This differentiates the EAP-TLS Start message from a fragment acknowledgement.

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.

5. References

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

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

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

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

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

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

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

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

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

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

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

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

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

6. Security Considerations

6.1. 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 a PPP interface and Internet connectivity established at the time of tunnel initiation. As a result, during the EAP conversation it is capable of checking for certificate revocation.

However, in the case where the peer is initiating an intial PPP conversation, it will not have Internet connectivity and is therefore not capable of checking for certificate revocation until after NCP negotiation completes and the peer has access to the Internet. In this case, the peer SHOULD check for certificate revocation after connecting to the Internet.

6.2. Separation of the EAP server and PPP authenticator

As a result of the EAP-TLS conversation, the EAP endpoints will mutually authenticate, negotiate a ciphersuite, and derive a session key for subsequent use in PPP encryption. 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 PPP encryption module.

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

6.3. Relationship of PPP encryption to other security mechanisms

It is envisaged that EAP-TLS will be used primarily with dialup PPP connections. However, there are also circumstances in which PPP encryption may be used along with Layer 2 tunneling protocols such as PPTP and L2TP.

In compulsory layer 2 tunneling, a PPP peer makes a connection to a NAS or router which tunnels the PPP packets to a tunnel server. Since with compulsory tunneling a PPP peer cannot tell whether its packets are being tunneled, let alone whether the network device is securing the tunnel, if security is required then the client must make its own arrangements. In the case where all endpoints cannot be relied upon to implement IPSEC, TLS, or another suitable security protocol, PPP encryption provides a convenient means to ensure the privacy of packets transiting between the client and the tunnel server.

7. Acknowledgments

Thanks to Terence Spies, Glen Zorn and Narendra Gidwani of Microsoft for useful discussions of this problem space.

8. Authors' Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Phone: 425-936-6605
EMail: bernarda@microsoft.com

Dan Simon
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Phone: 425-936-6711
EMail: dansimon@microsoft.com

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.