                   The EAP-TLS Authentication Protocol

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   The Extensible Authentication Protocol (EAP), defined in RFC 3748,
   provides support for multiple authentication methods. Transport
   Layer Security (TLS) provides for mutual authentication, integrity-
   protected ciphersuite negotiation, and key exchange between two
   endpoints. This document defines EAP-TLS, which includes support for
   certificate-based mutual authentication and key derivation.

   This document obsoletes RFC 2716. A summary of the changes between
   this document and RFC 2716 is available in Appendix A.

Table of Contents

1. Introduction

   The Extensible Authentication Protocol (EAP), described in [RFC3748],
   provides a standard mechanism for support of multiple authentication
   methods. Through the use of EAP, support for a number of
   authentication schemes may be added, including smart cards, Kerberos,
   Public Key, One Time Passwords, and others. EAP has been defined for
   use with a variety of lower layers, including the Point-to-Point
   Protocol (PPP) [RFC1661], Layer 2 tunneling protocols such as the
   Point-to-Point Tunneling Protocol (PPTP) [RFC2637] or Layer 2
   Tunneling Protocol (L2TP) [RFC2661], IEEE 802 wired networks
   [IEEE-802.1X], and wireless technologies such as IEEE 802.11 [IEEE-
   802.11] and IEEE 802.16 [IEEE-802.16e].

   While the EAP methods defined in [RFC3748] did not support mutual
   authentication, the use of EAP with wireless technologies such as
   [IEEE-802.11] has resulted in development of a new set of

requirements. As described in "Extensible Authentication Protocol
(EAP) Method Requirements for Wireless LANs" [RFC4017], it is
desirable for EAP methods used for wireless LAN authentication to
support mutual authentication and key derivation. Other link layers
can also make use of EAP to enable mutual authentication and key
derivation.

This document defines EAP-Transport Layer Security (EAP-TLS), which
includes support for certificate-based mutual authentication and key
derivation, utilizing the protected ciphersuite negotiation, mutual
authentication and key management capabilities of the TLS protocol,
described in "The Transport Layer Security (TLS) Protocol
Version 1.1" [RFC4346]. While this document obsoletes RFC 2716
[RFC2716], it remains backward compatible with it. A summary of the
changes between this document and RFC 2716 is available in Appendix
A.

1.1. Requirements

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

1.2. Terminology

This document frequently uses the following terms:

authenticator
   The entity initiating EAP authentication.

peer
   The entity that responds to the authenticator. In [IEEE-802.1X],
   this entity is known as the Supplicant.

backend authentication server
   A backend authentication server is an entity that provides an
   authentication service to an authenticator. When used, this server
   typically executes EAP methods for the authenticator. This
   terminology is also used in [IEEE-802.1X].

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

   Master Session Key (MSK)
      Keying material that is derived between the EAP peer and server and
      exported by the EAP method.

   Extended Master Session Key (EMSK)
      Additional keying material derived between the EAP peer and server
      that is exported by the EAP method.

2. Protocol Overview

2.1. Overview of the EAP-TLS Conversation

   As described in [RFC3748], 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 user-Id.

   From this point forward, while nominally the EAP conversation occurs
   between the EAP authenticator and the peer, the authenticator MAY act
   as a pass-through 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.

2.1.1. Base Case

   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 peer's TLS version number, a
   sessionId, a random number, and a set of ciphersuites supported by
   the peer. The version offered by the peer 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 peer'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 peer, indicating a resumption of the previously established session with that sessionId. The server will also choose a ciphersuite from those offered by the peer. If the session matches the peer's, then the ciphersuite MUST match the one negotiated during the handshake protocol execution that established the session.

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 Digital Signature Standard (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 peer to authenticate itself via public key. While the EAP server SHOULD require peer authentication, this is not mandatory, since there are circumstances in which peer authentication will not be needed (e.g., emergency services, as described in [UNAUTH]), or where the peer will authenticate via some other means.

If the peer supports EAP-TLS and is configured to use it, it MUST respond to the EAP-Request with an EAP-Response packet of EAP-Type=EAP-TLS. If the preceding server_hello message sent by the EAP server in the preceding EAP-Request packet did not indicate the resumption of a previous session, the data field of this packet MUST encapsulate one or more TLS records containing a TLS client_key_exchange, change_cipher_spec, and finished messages. If the EAP server sent a certificate_request message in the preceding EAP-Request packet, then unless the peer is configured for privacy (see Section 2.1.4) the peer MUST send, in addition, certificate and certificate_verify 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 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.

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

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

2.1.2. Session Resumption

   The purpose of the sessionId within the TLS protocol is to allow for
   improved efficiency in the case where a peer 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 also
   can be used to provide "fast reconnect" functionality as defined in
   Section 7.2.1 of [RFC3748].

   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 to choose a new session.

   In the case where the EAP server and authenticator reside on the same
   device, the peer will only be able to continue sessions when
   connecting to the same authenticator. Should the authenticators be
   set up in a rotary or round-robin, then it may not be possible for
   the peer to know in advance the authenticator to which it will be
   connecting, 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 authenticators will
   utilize 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.

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
-------------------        -------------
                           <- EAP-Request/
                           Identity
EAP-Response/
Identity (MyID) ->
                           <- EAP-Request/
                           EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS Start)
EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                           <- EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                           TLS change_cipher_spec
                           TLS finished)
EAP-Response/
EAP-Type=EAP-TLS
(TLS change_cipher_spec,
 TLS finished) ->
                           <- EAP-Success
```

2.1.3. Termination

   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 immediately terminating the
   conversation so as to allow the peer to inform the user or log 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 per-peer limit

on the number of restarts, so as to protect against denial-of-service
attacks.

If the peer 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 handshake messages. The latter
contains the EAP server's authentication response to the peer. The
peer will then verify the finished message in order to authenticate
the EAP server.

If EAP server authentication is unsuccessful, the peer SHOULD delete
the session from its cache, preventing reuse of the sessionId. 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.

In the case where the server authenticates to the peer successfully,
but the peer 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-Type=EAP-TLS
                           (TLS Start)
EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                           <- EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS server_hello,
                             TLS certificate,
                   [TLS server_key_exchange,]
                TLS certificate_request,
                  TLS server_hello_done)

EAP-Response/
EAP-Type=EAP-TLS
(TLS certificate,
 TLS client_key_exchange,
 TLS certificate_verify,
 TLS change_cipher_spec,
 TLS finished) ->

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

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

   Authenticating Peer   Authenticator
   ------------------   -------------
                              <- EAP-Request/
                              Identity
   EAP-Response/
   Identity (MyID) ->
                              <- EAP-Request/
                              EAP-Type=EAP-TLS
                              (TLS Start)
   EAP-Response/
   EAP-Type=EAP-TLS
    (TLS client_hello)->
                              <- EAP-Request/
                              EAP-Type=EAP-TLS
                              (TLS server_hello,
                               TLS certificate,
                     [TLS server_key_exchange,]
                      TLS certificate_request,
                      TLS server_hello_done)
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS Alert message) ->
                              <- EAP-Failure
                              (User Disconnected)

2.1.4. Privacy

   EAP-TLS peer and server implementations MAY support privacy.
   Disclosure of the username is avoided by utilizing a privacy Network
   Access Identifier (NAI) [RFC4282] in the EAP-Response/Identity, and
   transmitting the peer certificate within a TLS session providing
   confidentiality.

   In order to avoid disclosing the peer username, an EAP-TLS peer
   configured for privacy MUST negotiate a TLS ciphersuite supporting
   confidentiality and MUST provide a client certificate list containing
   no entries in response to the initial certificate_request from the
   EAP-TLS server.

   An EAP-TLS server supporting privacy MUST NOT treat a certificate
   list containing no entries as a terminal condition; instead, it MUST
   bring up the TLS session and then send a hello_request. The
   handshake then proceeds normally; the peer sends a client_hello and
   the server replies with a server_hello, certificate,
   server_key_exchange, certificate_request, server_hello_done, etc.

   For the calculation of exported keying material (see Section 2.3),
   the master_secret derived within the second handshake is used.

   An EAP-TLS peer supporting privacy MUST provide a certificate list
   containing at least one entry in response to the subsequent
   certificate_request sent by the server. If the EAP-TLS server
   supporting privacy does not receive a client certificate in response
   to the subsequent certificate_request, then it MUST abort the
   session.

   EAP-TLS privacy support is designed to allow EAP-TLS peers that do
   not support privacy to interoperate with EAP-TLS servers supporting
   privacy. EAP-TLS servers supporting privacy MUST request a client
   certificate, and MUST be able to accept a client certificate offered
   by the EAP-TLS peer, in order to preserve interoperability with EAP-
   TLS peers that do not support privacy.

   However, an EAP-TLS peer configured for privacy typically will not be
   able to successfully authenticate with an EAP-TLS server that does
   not support privacy, since such a server will typically treat the
   refusal to provide a client certificate as a terminal error. As a
   result, unless authentication failure is considered preferable to
   disclosure of the username, EAP-TLS peers SHOULD only be configured
   for privacy on networks known to support it.

   This is most easily achieved with EAP lower layers that support
   network advertisement, so that the network and appropriate privacy
   configuration can be determined. In order to determine the privacy
   configuration on link layers (such as IEEE 802 wired networks) that
   do not support network advertisement, it may be desirable to utilize
   information provided in the server certificate (such as the subject
   and subjectAltName fields) or within identity selection hints
   [RFC4284] to determine the appropriate configuration.

   In the case where the peer and server support privacy and mutual
   authentication, the conversation will appear as follows:

   Authenticating Peer   Authenticator
   -------------------   -------------
                            <- EAP-Request/
                            Identity
   EAP-Response/
   Identity (Anonymous NAI) ->
                            <- EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS Start)

```
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello)->
                            <- EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS server_hello,
                             TLS certificate,
                        [TLS server_key_exchange,]
                         TLS certificate_request,
                            TLS server_hello_done)
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS certificate (no cert),
    TLS client_key_exchange,
    TLS change_cipher_spec,
    TLS finished) ->
                            <- EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS change_cipher_spec,
                              finished,
                              hello_request)
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello)->
                            <- EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS server_hello,
                              TLS certificate,
                        TLS server_key_exchange,
                        TLS certificate_request,
                            TLS server_hello_done)
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS certificate,
    TLS client_key_exchange,
    TLS certificate_verify,
    TLS change_cipher_spec,
    TLS finished) ->
                            <- EAP-Request/
                            EAP-Type=EAP-TLS
                            (TLS change_cipher_spec,
                             TLS finished)
   EAP-Response/
   EAP-Type=EAP-TLS ->
                            <- EAP-Success
```

2.1.5. 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 16 MB. The group of EAP-TLS messages
   sent in a single round may thus be larger than the MTU size or the
   maximum Remote Authentication Dail-In User Service (RADIUS) packet
   size of 4096 octets. As a result, an EAP-TLS implementation MUST
   provide its own support for fragmentation and reassembly. However,
   in order to ensure interoperability with existing implementations,
   TLS handshake messages SHOULD NOT be fragmented into multiple TLS
   records if they fit within a single TLS record.

   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 single certificate
   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.

   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-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=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 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.

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

```
Authenticating Peer        Authenticator
-------------------        -------------
                           <- EAP-Request/
                           Identity
EAP-Response/
Identity (MyID) ->
                           <- EAP-Request/
                           EAP-Type=EAP-TLS
                           (TLS Start, S bit set)
EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                           <- 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)
EAP-Response/
EAP-Type=EAP-TLS ->
                           <- EAP-Request/
                             EAP-Type=EAP-TLS
                           (Fragment 2: M bit set)
EAP-Response/
EAP-Type=EAP-TLS ->
                           <- EAP-Request/
                           EAP-Type=EAP-TLS
                           (Fragment 3)
```

```
     EAP-Response/
     EAP-Type=EAP-TLS
     (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=EAP-TLS
     EAP-Response/
     EAP-Type=EAP-TLS
     (Fragment 2)->
                               <- EAP-Request/
                                EAP-Type=EAP-TLS
                                (TLS change_cipher_spec,
                                 TLS finished)
     EAP-Response/
     EAP-Type=EAP-TLS ->
                               <- EAP-Success
```

2.2. Identity Verification

   As noted in Section 5.1 of [RFC3748]:

      It is RECOMMENDED that the Identity Response be used primarily for
      routing purposes and selecting which EAP method to use. EAP
      Methods SHOULD include a method-specific mechanism for obtaining
      the identity, so that they do not have to rely on the Identity
      Response.

   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. EAP-TLS therefore provides a
   mechanism for determining both the peer identity (Peer-Id in
   [KEYFRAME]) and server identity (Server-Id in [KEYFRAME]). For
   details, see Section 5.2.

   Since the identity presented in the EAP-Response/Identity need not be
   related to the identity presented in the peer certificate, EAP-TLS
   implementations SHOULD NOT require that they be identical. However,
   if they are not identical, the identity presented in the EAP-
   Response/Identity is unauthenticated information, and SHOULD NOT be
   used for access control or accounting purposes.

2.3. Key Hierarchy

   Figure 1 illustrates the TLS Key Hierarchy, described in [RFC4346]
   Section 6.3. The derivation proceeds as follows:

   master_secret = TLS-PRF-48(pre_master_secret, "master secret",
                    client.random || server.random) key_block   =
   TLS-PRF-X(master_secret, "key expansion",
                    server.random || client.random)

   Where:

   TLS-PRF-X =   TLS pseudo-random function defined in [RFC4346],
                   computed to X octets.

   In EAP-TLS, the MSK, EMSK, and Initialization Vector (IV) are derived
   from the TLS master secret via a one-way function. This ensures that
   the TLS master secret cannot be derived from the MSK, EMSK, or IV
   unless the one-way function (TLS PRF) is broken. Since the MSK and
   EMSK are derived from the TLS master secret, if the TLS master secret
   is compromised then the MSK and EMSK are also compromised.

   The MSK is divided into two halves, corresponding to the "Peer to
   Authenticator Encryption Key" (Enc-RECV-Key, 32 octets) and
   "Authenticator to Peer Encryption Key" (Enc-SEND-Key, 32 octets).

   The IV is a 64-octet quantity that is a known value; octets 0-31 are
   known as the "Peer to Authenticator IV" or RECV-IV, and octets 32-63
   are known as the "Authenticator to Peer IV", or SEND-IV.

```
              |                    | pre_master_secret       |
       server |                    |                         | client
       Random |                    V                         | Random
              |   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           |
              |   |                                |          |
       +----->|   |          master_secret         |<----+
              |   |             (TMS)               |          |
              |   |                                |          |
              |   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           |
              |                    |                         |
              V                    V                         V
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                         |
     |                      key_block                          |
     |                label == "key expansion"                 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |       |       |       |       |       |
       | client| server| client| server| client| server
       | MAC   | MAC   | write | write | IV    | IV
       |       |       |       |       |       |
       V       V       V       V       V       V
```
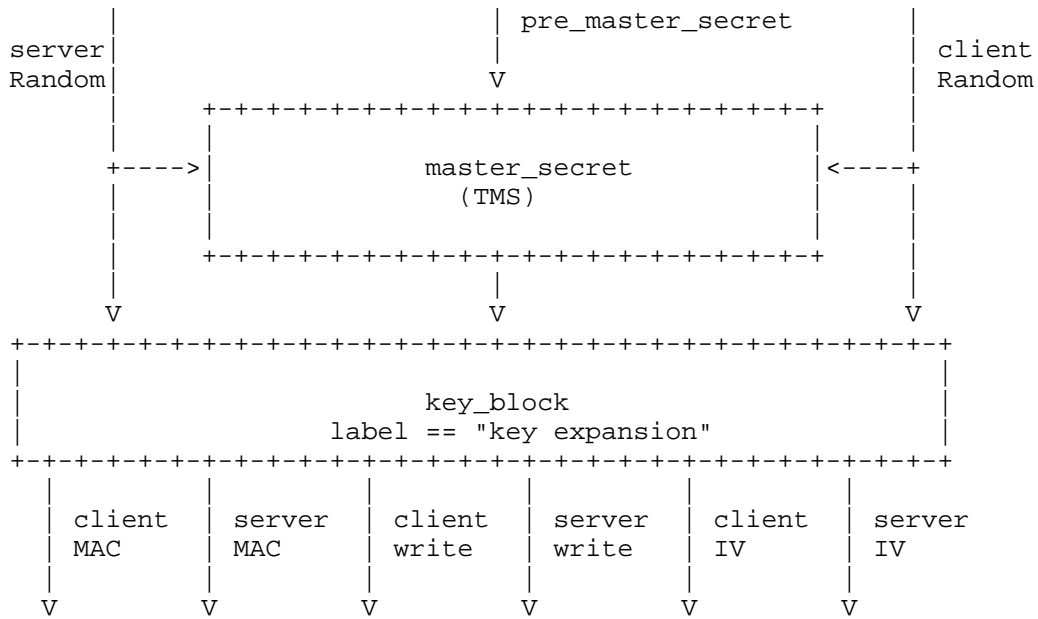
                Figure 1 - TLS [RFC4346] Key Hierarchy

   EAP-TLS derives exported keying material and parameters as follows:

```
   Key_Material = TLS-PRF-128(master_secret, "client EAP encryption",
                    client.random || server.random)
   MSK          = Key_Material(0,63)
   EMSK         = Key_Material(64,127)
   IV           = TLS-PRF-64("", "client EAP encryption",
                    client.random || server.random)


   Enc-RECV-Key = MSK(0,31) = Peer to Authenticator Encryption Key
                  (MS-MPPE-Recv-Key in [RFC2548]).  Also known as the
                  PMK in [IEEE-802.11].
   Enc-SEND-Key = MSK(32,63) = Authenticator to Peer Encryption Key
                  (MS-MPPE-Send-Key in [RFC2548])
   RECV-IV      = IV(0,31) = Peer to Authenticator Initialization Vector
   SEND-IV      = IV(32,63) = Authenticator to Peer Initialization
                              Vector
   Session-Id   = 0x0D || client.random || server.random
```

   Where:

   Key_Material(W,Z) = Octets W through Z inclusive of the key material.
   IV(W,Z)   = Octets W through Z inclusive of the IV.
   MSK(W,Z)   = Octets W through Z inclusive of the MSK.
   EMSK(W,Z)   = Octets W through Z inclusive of the EMSK.
   TLS-PRF-X   = TLS PRF function computed to X octets.
   client.random   = Nonce generated by the TLS client.
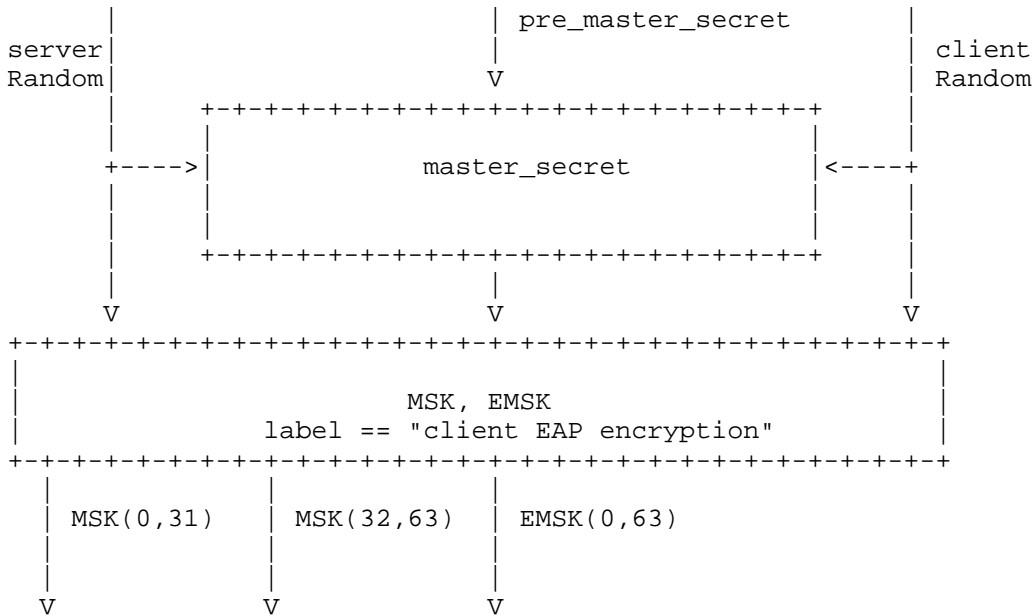   server.random   = Nonce generated by the TLS server.

```
       |                             |  pre_master_secret     |
 server|                             |                        |  client
 Random|                             V                        |  Random
       |       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+         |
       |       |                                     |        |
       +---->  |             master_secret           | <----+
       |       |                                     |        |
       |       |                                     |        |
       |       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+         |
       |                             |                        |
       V                             V                        V
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                           |
 |                       MSK, EMSK                           |
 |               label == "client EAP encryption"            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |            |              |
      |  MSK(0,31) |  MSK(32,63)  |  EMSK(0,63)
      |            |              |
      |            |              |
      V            V              V
```

                  Figure 2 - EAP-TLS Key Hierarchy

   The use of these keys is specific to the lower layer, as described in
   Section 2.1 of [KEYFRAME].

2.4. Ciphersuite and Compression Negotiation

   EAP-TLS implementations MUST support TLS v1.0.

   EAP-TLS implementations need not necessarily support all TLS
   ciphersuites listed in [RFC4346]. Not all TLS ciphersuites are
   supported by available TLS tool kits, and licenses may be required in
   some cases.

To ensure interoperability, EAP-TLS peers and servers MUST support
the TLS [RFC4346] mandatory-to-implement ciphersuite:

    TLS_RSA_WITH_3DES_EDE_CBC_SHA

EAP-TLS peers and servers SHOULD also support and be able to
negotiate the following TLS ciphersuites:

    TLS_RSA_WITH_RC4_128_SHA [RFC4346]
    TLS_RSA_WITH_AES_128_CBC_SHA [RFC3268]

In addition, EAP-TLS servers SHOULD support and be able to negotiate
the following TLS ciphersuite:

    TLS_RSA_WITH_RC4_128_MD5 [RFC4346]

Since TLS supports ciphersuite negotiation, peers completing the TLS
negotiation will also have selected a ciphersuite, which includes
encryption and hashing methods. Since the ciphersuite negotiated
within EAP-TLS applies only to the EAP conversation, TLS ciphersuite
negotiation MUST NOT be used to negotiate the ciphersuites used to
secure data.

TLS also supports compression as well as ciphersuite negotiation.
However, during the EAP-TLS conversation the EAP peer and server MUST
NOT request or negotiate compression.

3. Detailed Description of the EAP-TLS Protocol

3.1. EAP-TLS Request Packet

   A summary of the EAP-TLS 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     |      TLS Message Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     TLS Message Length        |       TLS Data...             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
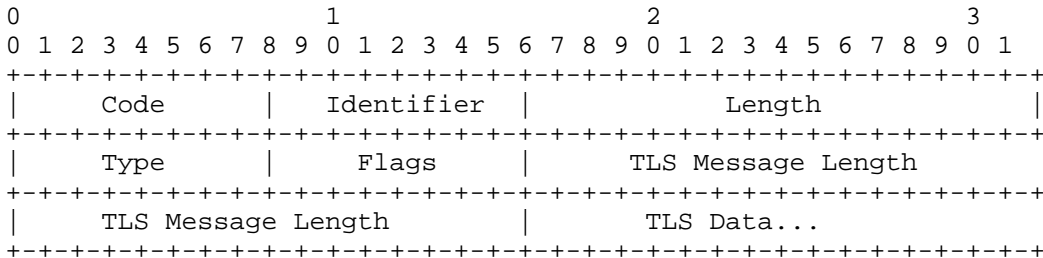
   Code

      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 Data
   fields. Octets outside the range of the Length field should be
   treated as Data Link Layer padding and MUST be ignored on
   reception.

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
   acknowledgment. Implementations of this specification MUST set
   the reserved bits to zero, and MUST ignore them on reception.

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.

3.2. EAP-TLS Response Packet

       A summary of the EAP-TLS 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     |      TLS Message Length        
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     TLS Message Length        |       TLS Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
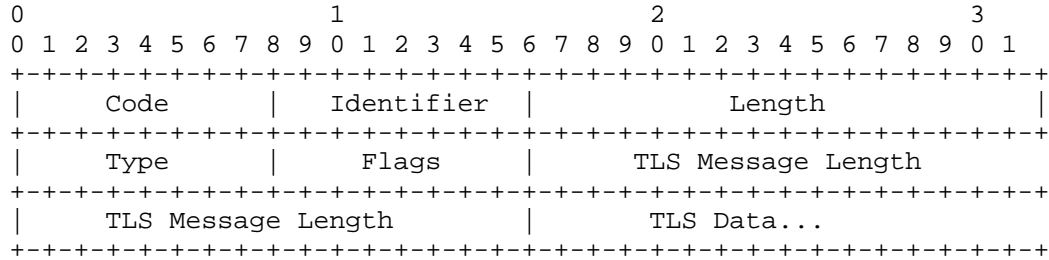
   Code

       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 Data
       fields. Octets outside the range of the Length field should be
       treated as Data Link Layer padding and MUST be ignored on
       reception.

   Type

       13 -- EAP-TLS

   Flags

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

      L = Length included
      M = More fragments
      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.
      Implementations of this specification MUST set the reserved bits
      to zero, and MUST ignore them on reception.

   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. IANA Considerations

   IANA has allocated EAP Type 13 for EAP-TLS. The allocation has been
   updated to reference this document.

5. Security Considerations

5.1. Security Claims

   EAP security claims are defined in Section 7.2.1 of [RFC3748]. The
   security claims for EAP-TLS are as follows:

   Auth. mechanism: Certificates
   Ciphersuite negotiation: Yes [4]
   Mutual authentication: Yes [1]
   Integrity protection: Yes [1]
   Replay protection: Yes [1]
   Confidentiality: Yes [2]
   Key derivation: Yes
   Key strength: [3]
   Dictionary attack prot.: Yes
   Fast reconnect: Yes
   Crypt. binding: N/A
   Session independence: Yes [1]
   Fragmentation: Yes
   Channel binding: No

   Notes
   -----

   [1] A formal proof of the security of EAP-TLS when used with
   [IEEE-802.11] is provided in [He]. This proof relies on the
   assumption that the private key pairs used by the EAP peer and server
   are not shared with other parties or applications. For example, a
   backend authentication server supporting EAP-TLS SHOULD NOT utilize
   the same certificate with https.

   [2] Privacy is an optional feature described in Section 2.1.4.

   [3] Section 5 of BCP 86 [RFC3766] offers advice on the required RSA
   or Diffie-Hellman (DH) module and Digital Signature Algorithm (DSA)
   subgroup size in bits, for a given level of attack resistance in
   bits. For example, a 2048-bit RSA key is recommended to provide
   128-bit equivalent key strength. The National Institute of Standards
   and Technology (NIST) also offers advice on appropriate key sizes in
   [SP800-57].

   [4] EAP-TLS inherits the secure ciphersuite negotiation features of
   TLS, including key derivation function negotiation when utilized with
   TLS v1.2 [RFC4346bis].

5.2. Peer and Server Identities

   The EAP-TLS peer name (Peer-Id) represents the identity to be used
   for access control and accounting purposes. The Server-Id represents
   the identity of the EAP server. Together the Peer-Id and Server-Id
   name the entities involved in deriving the MSK/EMSK.

   In EAP-TLS, the Peer-Id and Server-Id are determined from the subject
   or subjectAltName fields in the peer and server certificates. For
   details, see Section 4.1.2.6 of [RFC3280]. Where the subjectAltName
   field is present in the peer or server certificate, the Peer-Id or
   Server-Id MUST be set to the contents of the subjectAltName. If
   subject naming information is present only in the subjectAltName
   extension of a peer or server certificate, then the subject field
   MUST be an empty sequence and the subjectAltName extension MUST be
   critical.

   Where the peer identity represents a host, a subjectAltName of type
   dnsName SHOULD be present in the peer certificate. Where the peer
   identity represents a user and not a resource, a subjectAltName of
   type rfc822Name SHOULD be used, conforming to the grammar for the
   Network Access Identifier (NAI) defined in Section 2.1 of [RFC4282].
   If a dnsName or rfc822Name are not available, other field types (for
   example, a subjectAltName of type ipAddress or
   uniformResourceIdentifier) MAY be used.

   A server identity will typically represent a host, not a user or a
   resource. As a result, a subjectAltName of type dnsName SHOULD be
   present in the server certificate. If a dnsName is not available
   other field types (for example, a subjectAltName of type ipAddress or
   uniformResourceIdentifier) MAY be used.

   Conforming implementations generating new certificates with Network
   Access Identifiers (NAIs) MUST use the rfc822Name in the subject
   alternative name field to describe such identities. The use of the
   subject name field to contain an emailAddress Relative Distinguished
   Name (RDN) is deprecated, and MUST NOT be used. The subject name
   field MAY contain other RDNs for representing the subject's identity.

   Where it is non-empty, the subject name field MUST contain an X.500
   distinguished name (DN). If subject naming information is present
   only in the subject name field of a peer certificate and the peer
   identity represents a host or device, the subject name field SHOULD
   contain a CommonName (CN) RDN or serialNumber RDN. If subject naming
   information is present only in the subject name field of a server
   certificate, then the subject name field SHOULD contain a CN RDN or
   serialNumber RDN.

It is possible for more than one subjectAltName field to be present
in a peer or server certificate in addition to an empty or non-empty
subject distinguished name. EAP-TLS implementations supporting
export of the Peer-Id and Server-Id SHOULD export all the
subjectAltName fields within Peer-Ids or Server-Ids, and SHOULD also
export a non-empty subject distinguished name field within the Peer-
Ids or Server-Ids. All of the exported Peer-Ids and Server-Ids are
considered valid.

EAP-TLS implementations supporting export of the Peer-Id and Server-
Id SHOULD export Peer-Ids and Server-Ids in the same order in which
they appear within the certificate. Such canonical ordering would
aid in comparison operations and would enable using those identifiers
for key derivation if that is deemed useful. However, the ordering
of fields within the certificate SHOULD NOT be used for access
control.

5.3. Certificate Validation

Since the EAP-TLS server is typically connected to the Internet, it
SHOULD support validating the peer certificate using RFC 3280
[RFC3280] compliant path validation, including the ability to
retrieve intermediate certificates that may be necessary to validate
the peer certificate. For details, see Section 4.2.2.1 of [RFC3280].

Where the EAP-TLS server is unable to retrieve intermediate
certificates, either it will need to be pre-configured with the
necessary intermediate certificates to complete path validation or it
will rely on the EAP-TLS peer to provide this information as part of
the TLS handshake (see Section 7.4.6 of [RFC4346]).

In contrast to the EAP-TLS server, the EAP-TLS peer may not have
Internet connectivity. Therefore, the EAP-TLS server SHOULD provide
its entire certificate chain minus the root to facilitate certificate
validation by the peer. The EAP-TLS peer SHOULD support validating
the server certificate using RFC 3280 [RFC3280] compliant path
validation.

Once a TLS session is established, EAP-TLS peer and server
implementations MUST validate that the identities represented in the
certificate are appropriate and authorized for use with EAP-TLS. The
authorization process makes use of the contents of the certificates
as well as other contextual information. While authorization
requirements will vary from deployment to deployment, it is
RECOMMENDED that implementations be able to authorize based on the
EAP-TLS Peer-Id and Server-Id determined as described in Section 5.2.

In the case of the EAP-TLS peer, this involves ensuring that the
certificate presented by the EAP-TLS server was intended to be used
as a server certificate. Implementations SHOULD use the Extended Key
Usage (see Section 4.2.1.13 of [RFC3280]) extension and ensure that
at least one of the following is true:

1) The certificate issuer included no Extended Key Usage identifiers
   in the certificate.
2) The issuer included the anyExtendedKeyUsage identifier in the
   certificate (see Section 4.2.1.13 of [RFC3280]).
3) The issuer included the id-kp-serverAuth identifier in the
   certificate (see Section 4.2.1.13 [RFC3280]).

When performing this comparison, implementations MUST follow the
validation rules specified in Section 3.1 of [RFC2818]. In the case
of the server, this involves ensuring the certificate presented by
the EAP-TLS peer was intended to be used as a client certificate.
Implementations SHOULD use the Extended Key Usage (see Section
4.2.1.13 [RFC3280]) extension and ensure that at least one of the
following is true:

1) The certificate issuer included no Extended Key Usage identifiers
   in the certificate.
2) The issuer included the anyExtendedKeyUsage identifier in the
   certificate (see Section 4.2.1.13 of [RFC3280]).
3) The issuer included the id-kp-clientAuth identifier in the
   certificate (see Section 4.2.1.13 of [RFC3280]).

5.4. Certificate Revocation

   Certificates are long-lived assertions of identity. Therefore, it is
   important for EAP-TLS implementations to be capable of checking
   whether these assertions have been revoked.

   EAP-TLS peer and server implementations MUST support the use of
   Certificate Revocation Lists (CRLs); for details, see Section 3.3 of
   [RFC3280]. EAP-TLS peer and server implementations SHOULD also
   support the Online Certificate Status Protocol (OCSP), described in
   "X.509 Internet Public Key Infrastructure Online Certificate Status
   Protocol - OCSP" [RFC2560]. OCSP messages are typically much smaller
   than CRLs, which can shorten connection times especially in
   bandwidth-constrained environments. While EAP-TLS servers are
   typically connected to the Internet during the EAP conversation, an
   EAP-TLS peer may not have Internet connectivity until authentication
   completes.

In the case where the peer is initiating a voluntary Layer 2 tunnel
using PPTP [RFC2637] or L2TP [RFC2661], the peer will typically
already have a PPP interface and Internet connectivity established at
the time of tunnel initiation.

However, in the case where the EAP-TLS peer is attempting to obtain
network access, it will not have network connectivity and is
therefore not capable of checking for certificate revocation until
after authentication completes and network connectivity is available.
For this reason, EAP-TLS peers and servers SHOULD implement
Certificate Status Request messages, as described in "Transport Layer
Security (TLS) Extensions", Section 3.6 of [RFC4366]. To enable
revocation checking in situations where servers do not support
Certificate Status Request messages and network connectivity is not
available prior to authentication completion, peer implementations
MUST also support checking for certificate revocation after
authentication completes and network connectivity is available, and
they SHOULD utilize this capability by default.

5.5. Packet Modification Attacks

The integrity protection of EAP-TLS packets does not extend to the
EAP header fields (Code, Identifier, Length) or the Type or Flags
fields. As a result, these fields can be modified by an attacker.

In most cases, modification of the Code or Identifier fields will
only result in a denial-of-service attack. However, an attacker can
add additional data to an EAP-TLS packet so as to cause it to be
longer than implied by the Length field. EAP peers, authenticators,
or servers that do not check for this could be vulnerable to a buffer
overrun.

It is also possible for an attacker to modify the Type or Flags
fields. By modifying the Type field, an attacker could cause one
TLS-based EAP method to be negotiated instead of another. For
example, the EAP-TLS Type field (13) could be changed to indicate
another TLS-based EAP method. Unless the alternative TLS-based EAP
method utilizes a different key derivation formula, it is possible
that an EAP method conversation altered by a man-in-the-middle could
run all the way to completion without detection. Unless the
ciphersuite selection policies are identical for all TLS-based EAP
methods utilizing the same key derivation formula, it may be possible
for an attacker to mount a successful downgrade attack, causing the
peer to utilize an inferior ciphersuite or TLS-based EAP method.

6. References

6.1. Normative References

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

[RFC2560]   Myers, M., Ankney, R., Malpani, A., Galperin, S., and
                C. Adams, "X.509 Internet Public Key Infrastructure
                Online Certificate Status Protocol - OCSP", RFC 2560,
                June 1999.

[RFC2818]   Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3268]   Chown, P., "Advanced Encryption Standard (AES)
                Ciphersuites for Transport Layer Security (TLS)", RFC
                3268, June 2002.

[RFC3280]   Housley, R., Polk, W., Ford, W., and D. Solo,
                "Internet X.509 Public Key Infrastructure Certificate
                and Certificate Revocation List (CRL) Profile", RFC
                3280, April 2002.

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

[RFC4282]   Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The
                Network Access Identifier", RFC 4282, December 2005.

[RFC4346]   Dierks, T. and E. Rescorla, "The Transport Layer
                Security (TLS) Protocol Version 1.1", RFC 4346, April
                2006.

[RFC4366]   Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen,
                J., and T. Wright, "Transport Layer Security (TLS)
                Extensions", RFC 4366, April 2006.

6.2. Informative References

[IEEE-802.1X] Institute of Electrical and Electronics Engineers,
                "Local and Metropolitan Area Networks: Port-Based
                Network Access Control", IEEE Standard 802.1X-2004,
                December 2004.

   [IEEE-802.11]   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.
                   802.11-2007, 2007.

   [IEEE-802.16e] Institute of Electrical and Electronics Engineers,
                   "IEEE Standard for Local and Metropolitan Area
                   Networks: Part 16: Air Interface for Fixed and Mobile
                   Broadband Wireless Access Systems: Amendment for
                   Physical and Medium Access Control Layers for Combined
                   Fixed and Mobile Operations in Licensed Bands", IEEE
                   802.16e, August 2005.

   [He]   He, C., Sundararajan, M., Datta, A., Derek, A. and J.
                   Mitchell, "A Modular Correctness Proof of IEEE 802.11i
                   and TLS", CCS '05, November 7-11, 2005, Alexandria,
                   Virginia, USA

   [KEYFRAME]   Aboba, B., Simon, D. and P. Eronen, "Extensible
                   Authentication Protocol (EAP) Key Management
                   Framework", Work in Progress, November 2007.

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

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

   [RFC2637]   Hamzeh, K., Pall, G., Verthein, W., Taarud, J.,
                   Little, W., and G. Zorn, "Point-to-Point Tunneling
                   Protocol (PPTP)", RFC 2637, July 1999.

   [RFC2661]   Townsley, W., Valencia, A., Rubens, A., Pall, G.,
                   Zorn, G., and B. Palter, "Layer Two Tunneling Protocol
                   "L2TP"", RFC 2661, August 1999.

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

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

   [RFC4017]   Stanley, D., Walker, J., and B. Aboba, "Extensible
                   Authentication Protocol (EAP) Method Requirements for
                   Wireless LANs", RFC 4017, March 2005.

   [RFC4284]   Adrangi, F., Lortz, V., Bari, F., and P. Eronen,
                    "Identity Selection Hints for the Extensible
                    Authentication Protocol (EAP)", RFC 4284, January
                    2006.

   [SP800-57]   National Institute of Standards and Technology,
                    "Recommendation for Key Management", Special
                    Publication 800-57, May 2006.

   [RFC4346bis]   Dierks, T. and E. Rescorla, "The TLS Protocol Version
                    1.2", Work in Progress, February 2008.

   [UNAUTH]   Schulzrinne. H., McCann, S., Bajko, G. and H.
                    Tschofenig, "Extensions to the Emergency Services
                    Architecture for dealing with Unauthenticated and
                    Unauthorized Devices", Work in Progress, November
                    2007.

Acknowledgments

Appendix A -- Changes from RFC 2716

   This appendix lists the major changes between [RFC2716] and this
   document. Minor changes, including style, grammar, spelling, and
   editorial changes, are not mentioned here.

   o As EAP is now in use with a variety of lower layers, not just PPP
      for which it was first designed, mention of PPP is restricted to
      situations relating to PPP-specific behavior and reference is made
      to other lower layers such as IEEE 802.11, IEEE 802.16, etc.

   o The document now cites TLS v1.1 as a normative reference (Sections
      1 and 6.1).

   o The terminology section has been updated to reflect definitions
      from [RFC3748] (Section 1.2), and the EAP Key Management Framework
      [KEYFRAME] (Section 1.2).

   o Use for peer unauthenticated access is clarified (Section 2.1.1).

   o Privacy is supported as an optional feature (Section 2.1.4).

   o It is no longer recommended that the identity presented in the
      EAP-Response/Identity be compared to the identity provided in the
      peer certificate (Section 2.2).

   o The EAP-TLS key hierarchy is defined, using terminology from
      [RFC3748]. This includes formulas for the computation of TEKs as
      well as the MSK, EMSK, IV, and Session-Id (Section 2.3).

   o Mandatory and recommended TLS ciphersuites are provided. The use
      of TLS ciphersuite negotiation for determining the lower layer
      ciphersuite is prohibited (Section 2.4).

   o The Start bit is not set within an EAP-Response packet (Section
      3.2).

   o A section on security claims has been added and advice on key
      strength is provided (Section 5.1).

   o The Peer-Id and Server-Id are defined (Section 5.2), and
      requirements for certificate validation (Section 5.3) and
      revocation (Section 5.4) are provided.

   o Packet modification attacks are described (Section 5.5).

   o The examples have been updated to reflect typical messages sent in
      the described scenarios. For example, where mutual authentication
      is performed, the EAP-TLS server is shown to request a client
      certificate and the peer is shown to provide a certificate_verify
      message. A privacy example is provided, and two faulty examples
      of session resume failure were removed.

Authors' Addresses

   Dan Simon
   Microsoft Corporation
   One Microsoft Way
   Redmond, WA 98052-6399

   Phone: +1 425 882 8080
   Fax: +1 425 936 7329
   EMail: dansimon@microsoft.com

   Bernard Aboba
   Microsoft Corporation
   One Microsoft Way
   Redmond, WA 98052-6399

   Phone: +1 425 706 6605
   Fax: +1 425 936 7329
   EMail: bernarda@microsoft.com

   Ryan Hurst
   Microsoft Corporation
   One Microsoft Way
   Redmond, WA 98052-6399

   Phone: +1 425 882 8080
   Fax: +1 425 936 7329
   EMail: rmh@microsoft.com