Tunnel EAP Method (TEAP) Version 1
draft-ietf-emu-eap-tunnel-method-03.txt

Abstract

This document defines the Tunnel Extensible Authentication Protocol
(TEAP) version 1. TEAP is a tunnel based EAP method that enables
secure communication between a peer and a server by using the
Transport Layer Security (TLS) to establish a mutually authenticated
tunnel. Within the tunnel, Type-Length-Value (TLV) objects are used
to convey authentication related data between the EAP peer and the
EAP server.

Status of This Memo

Copyright Notice

carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

   An Extensible Authentication Protocol (EAP) tunnel method is an EAP
   method that establishes a secure tunnel and executes other EAP
   methods under the protection of that secure tunnel. An EAP tunnel
   method can be used in any lower layer protocol that supports EAP
   authentication. There are several existing EAP tunnel methods that
   use Transport Layer Security (TLS) [RFC5246] to establish the secure
   tunnel. EAP methods supporting this include Protected EAP (PEAP)
   [PEAP], Tunneled Transport Layer Security EAP (TTLS) [RFC5281] and
   EAP Flexible Authentication via Secure Tunneling (EAP-FAST)
   [RFC4851]. However, they all are either vendor specific or
   informational and industry calls for a standard-track tunnel EAP
   method. [I-D.ietf-emu-eaptunnel-req] outlines the list of
   requirements for a standard tunnel based EAP method.

   Since the introduction of EAP-FAST [RFC4851] a few years ago, it has
   been widely adopted in variety of devices and platforms due to its
   strong security, flexibility and ease of deployment. It has been
   adopted by EMU working group as the basis for the standard tunnel
   based EAP method. This document describes Tunnel Extensible
   Authentication Protocol (TEAP) version 1, based on EAP-FAST [RFC4851]
   with some minor changes, to meet the requirements outlined in
   [I-D.ietf-emu-eaptunnel-req] for a standard tunnel based EAP method.

1.1. Specification 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. Design Goals

   Network access solutions requiring user friendly and easily
   deployable secure authentication mechanisms highlight the need for
   strong mutual authentication protocols that enable the use of weaker
   user credentials. This document defines an Extensible Authentication
   Protocol (EAP) which consists of establishing a Transport Layer
   Security (TLS) tunnel using TLS version 1.2 [RFC5246] or a successor
   version supported by both parties. Once the tunnel is established,
   the protocol further exchanges data in the form of Type-Length-Value
   (TLV) objects to perform further authentication. TEAP supports the
   TLS extension defined in [RFC5077] to support fast re-establishment
   of the secure tunnel without having to maintain per-session state on
   the server.

   TEAP's design motivations included:

   o Mutual authentication: an EAP server must be able to verify the
      identity and authenticity of the peer, and the peer must be able
      to verify the authenticity of the EAP server.

   o Immunity to passive dictionary attacks: many authentication
      protocols require a password to be explicitly provided (either as
      cleartext or hashed) by the peer to the EAP server; at minimum,
      the communication of the weak credential (e.g., password) must be
      immune from eavesdropping.

   o Immunity to man-in-the-middle (MitM) attacks: in establishing a
      mutually authenticated protected tunnel, the protocol must prevent
      adversaries from successfully interjecting information into the
      conversation between the peer and the EAP server.

   o Flexibility to enable support for most password authentication
      interfaces: as many different password interfaces (e.g., Microsoft
      Challenge Handshake Authentication Protocol (MS-CHAP), Lightweight
      Directory Access Protocol (LDAP), One-Time Password (OTP), etc.)
      exist to authenticate a peer, the protocol must provide this
      support for legacy password authentication seamlessly.

   o Cryptographic algorithm agility: a cryptographic algorithm's
      strength is not perpetual, as weaknesses in an algorithm are
      discovered or increased processing power overtakes an algorithm
      over time. Hence, the protocol must not be tied to any single
      cryptographic algorithm. Instead, it MUST support run-time
      negotiation to select among an extensible set of cryptographic
      algorithms and also allow users to choose the algorithm that best
      meets their needs.

   o Sequence of chained EAP methods: Several circumstances are best
      addressed by using chained EAP methods. For example, it may be
      desirable to authenticate the user and also authenticate the
      device being used. The protocol must support chained EAP methods
      while including protection against attacks on method chaining.

   With these motivational goals defined, further secondary design
   criteria are imposed:

   o Flexibility to extend the communications inside the tunnel: with
      the growing complexity in network infrastructures, the need to
      gain authentication, authorization, and accounting is also
      evolving. For instance, there may be instances in which multiple
      existing authentication protocols are required to achieve mutual
      authentication. Similarly, different protected conversations may
      be required to achieve the proper authorization once a peer has
      successfully authenticated.

   o Minimize the authentication server's per user authentication state
      requirements: with large deployments, it is typical to have
      servers authenticating many peers. With many different
      authentication servers deployed, a peer's session state may need
      to be replicated to allow for high availability or mobility
      scenarios. To facilitate scalable authentication server
      deployments and more efficient per user state management, it is
      desirable for a peer to cache its session state that has been
      securely encapsulated by the authentication server infrastructure.

   o Efficiency: specifically when using wireless media, peers will be
      limited in computational and power resources. The protocol must
      enable the network access communication to be computationally
      lightweight.

   o Channel bindings: EAP channel bindings seek to authenticate
      previously unauthenticated information provided by the
      authenticator to the EAP peer, by allowing the peer and server to
      compare their perception of network properties in a secure
      channel. It is used to solve the lying NAS and the lying provider
      problems. The protocol should provide support for EAP channel
      bindings as defined in [I-D.ietf-emu-chbind].

1.3. Terminology

   Much of the terminology in this document comes from [RFC3748].
   Additional terms are defined below:

   Protected Access Credential (PAC)

      Credentials distributed to a peer for future optimized network
      authentication. The PAC consists of a minimum of two components:
      a shared secret and an opaque element. The shared secret
      component contains the pre-shared key between the peer and the
      authentication server. The opaque part is provided to the peer
      and is presented to the authentication server when the peer wishes
      to obtain access to network resources. The opaque element and
      shared secret are used with TLS stateless session resumption
      defined in RFC 5077 [RFC5077] to establish a protected TLS
      session. The secret key and opaque part may distributed using RFC
      5077 messages or using TLVs within the TEAP tunnel. Finally, a
      PAC may optionally include other information that may be useful to
      the peer.

   Type-Length-Value (TLV)

      The TEAP protocol utilizes objects in Type Length Value (TLV)
      format. The TLV format is defined in Section 4.2.

2. Protocol Overview

   TEAP authentication occurs in two phases. In the first phase, TEAP
   employs the TLS [RFC5246] handshake to provide an authenticated key
   exchange and to establish a protected tunnel. Once the tunnel is
   established, the second phase begins with the peer and server
   engaging in further conversations to establish the required
   authentication and authorization policies. TEAP makes use of Type-
   Length-Value objects (TLVs) to carry out the inner authentication,
   results and other information, such as channel binding information.

   TEAP makes use of the TLS enhancements in Ticket Extension [RFC5077]
   to enable an optimized TLS tunnel session resume while minimizing
   server state. The ticket is referred to as the Protected Access
   Credential opaque data (or PAC-Opaque). The PAC-Opaque may be
   distributed through the use of the NewSessionTicket message or
   through a mechanism that uses TLVs within phase 2 of TEAP. The
   secret key used to resume the session in TEAP is referred to as the
   Protected Access Credential key (or PAC-Key). When the
   NewSessionTicket message is being used to distribute the PAC-Opaque,
   the PAC-Key is the Master Secret for the session. If TEAP phase 2 is
   used to distribute the PAC-Opaque, then the PAC-Key is distributed
   along with the PAC-Opaque. TEAP implementations MUST support the RFC
   5077 mechanism for distributing a PAC-Opaque and it is RECOMMENDED
   that implementations support the capability to distribute the ticket
   and secret key within the TEAP tunnel.

   The TEAP conversation is used to establish or resume an existing
   session to typically establish network connectivity between a peer
   and the network. Upon successful execution of TEAP, both EAP peer
   and EAP server derive strong session key material that can then be
   communicated to the network access server (NAS) for use in
   establishing a link layer security association.

2.1. Architectural Model

   The network architectural model for TEAP usage is shown below:

```
    +----------+        +----------+        +----------+        +----------+
    |          |        |          |        |          |        | Inner    |
    |   Peer   | <----> | Authen-  | <----> |  TEAP    | <----> | Method   |
    |          |        | ticator  |        | server   |        | server   |
    |          |        |          |        |          |        |          |
    +----------+        +----------+        +----------+        +----------+
```
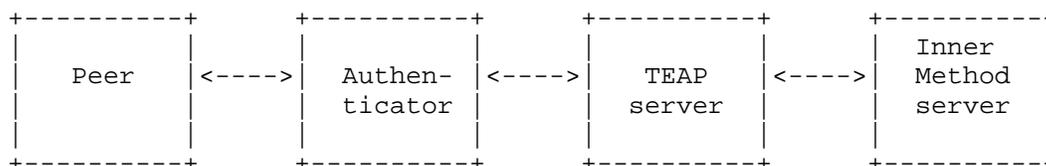
                     TEAP Architectural Model

The entities depicted above are logical entities and may or may not
correspond to separate network components. For example, the TEAP
server and inner method server might be a single entity; or the
authenticator and TEAP server might be a single entity; or the
functions of the authenticator, TEAP server, and inner method server
might be combined into a single physical device. For example,
typical IEEE 802.11 deployments place the Authenticator in an access
point (AP) while a Radius server may provide the TEAP and inner
method server components. The above diagram illustrates the division
of labor among entities in a general manner and shows how a
distributed system might be constructed; however, actual systems
might be realized more simply. The security considerations
Section 7.3 provides an additional discussion of the implications of
separating the TEAP server from the inner method server.

2.2. Protocol Layering Model

   TEAP packets are encapsulated within EAP; EAP in turn requires a
   carrier protocol for transport. TEAP packets encapsulate TLS, which
   is then used to encapsulate user authentication information. Thus,
   TEAP messaging can be described using a layered model, where each
   layer encapsulates the layer above it. The following diagram
   clarifies the relationship between protocols:

```
   +------------------------------------------------------------+
   |     Inner EAP Method     |      Other TLV information       |
   |------------------------------------------------------------|
   |            TLV Encapsulation (TLVs)                         |
   |------------------------------------------------------------|
   |      TLS          |       Optional Outer TLVs               |
   |------------------------------------------------------------|
   |                         TEAP                                |
   |------------------------------------------------------------|
   |                         EAP                                 |
   |------------------------------------------------------------|
   |    Carrier Protocol (EAP over LAN, RADIUS, Diameter, etc.)  |
   +------------------------------------------------------------+
```
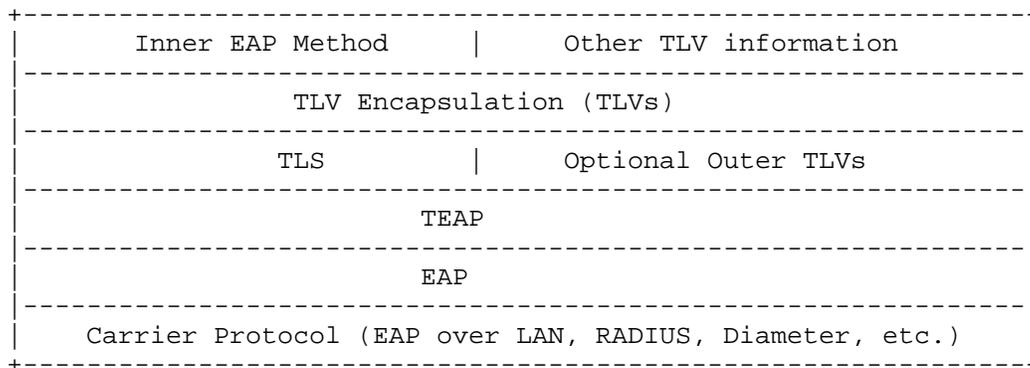
                      Protocol Layering Model

   The TLV layer is a payload with Type-Length-Value (TLV) Objects
   defined in Section 4.2. The TLV objects are used to carry arbitrary
   parameters between an EAP peer and an EAP server. All conversations
   in the TEAP protected tunnel must be encapsulated in a TLV layer.

   TEAP packets may include TLVs both inside and outside the TLS tunnel.

The term "Outer TLVs" is used to refer to optional TLVs outside the TLS tunnel, which are only allowed in the first two messages in the TEAP protocol. That is the first EAP server to peer message and first peer to EAP server message. If the message is fragmented, the whole set of messages is counted as one message. The term "Inner TLVs" is used to refer to TLVs sent within the TLS tunnel. In TEAP Phase 1, Outer TLVs are used to help establishing the TLS tunnel, but no Inner TLVs are used. In Phase 2 of the TEAP conversation, TLS records may encapsulate zero or more Inner TLVs, but no Outer TLVs.

Methods for encapsulating EAP within carrier protocols are already defined. For example, IEEE 802.1X [IEEE.802-1X.2004] may be used to transport EAP between the peer and the authenticator; RADIUS [RFC3579] or Diameter [RFC4072] may be used to transport EAP between the authenticator and the EAP server.

3. TEAP Protocol

TEAP authentication occurs in two phases. In the first phase, TEAP employs the TLS handshake to provide an authenticated key exchange and to establish a protected tunnel. Once the tunnel is established the second phase begins with the peer and server engaging in further conversations to establish the required authentication and authorization policies. The operation of the protocol, including Phase 1 and Phase 2, is the topic of this section. The format of TEAP messages is given in Section 4 and the cryptographic calculations are given in Section 5.

3.1. Version Negotiation

TEAP packets contain a 3-bit version field, following the TLS Flags field, which enables future TEAP implementations to be backward compatible with previous versions of the protocol. This specification documents the TEAP version 1 protocol; implementations of this specification MUST use a version field set to 1.

Version negotiation proceeds as follows:

In the first EAP-Request sent with EAP type=TEAP, the EAP server must set the version field to the highest supported version number.

If the EAP peer supports this version of the protocol, it MUST respond with an EAP-Response of EAP type=TEAP, and the version number proposed by the TEAP server.

If the TEAP peer does not support this version but supports the
version that is lower than the version proposed by the TEAP
server, it responds with an EAP-Response of EAP type=TEAP and the
highest supported version number. If the TEAP peer only supports
the version that is higher than the version proposed by the TEAP
server, then use of TEAP will not be possible. In this case, the
TEAP peer should send back an EAP-Nak with other proposed EAP
method if available.

If the TEAP server does not support the version number proposed by
the TEAP peer, it MAY terminate the conversation with EAP-Failure
or negotiate for another EAP type. Otherwise the TEAP
conversation continues.

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

The TEAP version is not protected by TLS; and hence can be modified
in transit. In order to detect a modification of the TEAP version,
the peers MUST exchange the TEAP version number received during
version negotiation using the Crypto-Binding TLV described in
Section 4.2.13. The receiver of the Crypto-Binding TLV MUST verify
that the version received in the Crypto-Binding TLV matches the
version sent by the receiver in the TEAP version negotiation.

3.2. TEAP Authentication Phase 1: Tunnel Establishment

TEAP is based on the TLS handshake [RFC5246] to establish an
authenticated and protected tunnel. The TLS version offered by the
peer and server MUST be TLS version 1.2 [RFC5246] or later. This
version of the TEAP implementation MUST support the following TLS
ciphersuites:

TLS_RSA_WITH_AES_128_CBC_SHA [RFC5246]

TLS_DHE_RSA_WITH_AES_128_CBC_SHA [RFC5246]

Other ciphersuites MAY be supported. It is REQUIRED that anonymous
ciphersuites such as TLS_DH_anon_WITH_AES_128_CBC_SHA [RFC5246] only
be used in the case when the inner authentication method provides
mutual authentication, key generation, and resistance to man-in-the-
middle and dictionary attack. During the TEAP Phase 1 conversation,
the TEAP endpoints MAY negotiate TLS compression. During TLS tunnel
establishment, TLS extensions MAY be used. For instance, Certificate

Status Request extension [RFC6066] can be used to leverage a certificate-status protocol such as OCSP [RFC2560] to check the validity of server certificates. TLS renegotiation indications defined in RFC 5746 [RFC5746] MUST be supported.

The EAP server initiates the TEAP conversation with an EAP request containing a TEAP/Start packet. This packet includes a set Start (S) bit, the TEAP version as specified in Section 3.1, and an authority identity TLV. The TLS payload in the initial packet is empty. The authority identity TLV (Authority-ID TLV) is used to provide the peer a hint of the server's identity that may be useful in helping the peer select the appropriate credential to use. Assuming that the peer supports TEAP, the conversation continues with the peer sending an EAP-Response packet with EAP type of TEAP with the Start (S) bit clear and the version as specified in Section 3.1. This message encapsulates one or more TLS records containing the TLS handshake messages. If the TEAP version negotiation is successful then the TEAP conversation continues until the EAP server and EAP peer are ready to enter Phase 2. When the full TLS handshake is performed, then the first payload of TEAP Phase 2 MAY be sent along with server-finished handshake message to reduce the number of round trips.

TEAP implementations MUST support client authentication during tunnel establishment using the TLS ciphersuites specified in Section 3.2. The EAP peer does not need to authenticate as part of the TLS exchange, but can alternatively be authenticated through additional exchanges carried out in Phase 2.

The TEAP tunnel protects peer identity information exchanged during phase 2 from disclosure outside the tunnel. Implementations that wish to provide identity privacy for the peer identity must carefully consider what information is disclosed outside the tunnel prior to phase 2. TEAP implementations SHOULD support the immediate renegotiation of a TLS session to initiate a new handshake message exchange under the protection of the current cipher suite. This allows support for protection of the peer's identity when using TLS client authentication.

The following sections describe resuming a TLS session based on server-side or client-side state.

3.2.1. TLS Session Resume Using Server State

TEAP session resumption is achieved in the same manner TLS achieves session resume. To support session resumption, the server and peer must minimally cache the Session ID, master secret, and ciphersuite. The peer attempts to resume a session by including a valid Session ID from a previous handshake in its ClientHello message. If the server

finds a match for the Session ID and is willing to establish a new
connection using the specified session state, the server will respond
with the same Session ID and proceed with the TEAP Phase 1 tunnel
establishment based on a TLS abbreviated handshake. After a
successful conclusion of the TEAP Phase 1 conversation, the
conversation then continues on to Phase 2.

3.2.2. TLS Session Resume Using a PAC

   TEAP supports the resumption of sessions based on server state being
   stored on the client side using the TLS SessionTicket extension
   techniques described in [RFC5077]. This version of TEAP supports the
   provisioning of a ticket called a Protected Access Credential (PAC)
   through the use of the NewSessionTicket handshake described in
   [RFC5077], as well as provisioning of a PAC inside the protected
   tunnel. Implementations may provide additional ways to provision the
   PAC, such as manual configuration. Since the PAC mentioned here is
   used for establishing the TLS Tunnel, it is more specifically
   referred to as the Tunnel PAC. The Tunnel PAC is a security
   credential provided by the EAP server to a peer and comprised of:

   1. PAC-Key: this is the key used by the peer as the TLS master
       secret to establish the TEAP Phase 1 tunnel. The PAC-Key is a
       strong high-entropy at minimum 48-octet key and is typically the
       master secret from a previous TLS session. The PAC-Key is a
       secret and MUST be treated accordingly. In the case that a PAC-
       Key is provisioned to the client through another means it must
       have its confidentiality and integrity protected by a mechanism,
       such as the TEAP phase 2 tunnel. The PAC-Key must be stored
       securely by the peer.

   2. PAC-Opaque: this is a variable length field containing the ticket
       that is sent to the EAP server during the TEAP Phase 1 tunnel
       establishment based on RFC 5077. The PAC-Opaque can only be
       interpreted by the EAP server to recover the required information
       for the server to validate the peer's identity and
       authentication. The PAC-Opaque includes the PAC-Key and other
       TLS session parameters. It may contain the PAC's peer identity.
       The PAC-Opaque format and contents are specific to the PAC
       issuing server. The PAC-Opaque may be presented in the clear, so
       an attacker MUST NOT be able to gain useful information from the
       PAC-Opaque itself. The server issuing the PAC-Opaque must ensure
       it is protected with strong cryptographic keys and algorithms.
       The PAC-Opaque may be distributed using the NewSessionTicket
       message defined in RFC 5077 or it may be distributed through
       another mechanism such as the phase 2 TLVs defined in this
       document.

3. PAC-Info: this is an optional variable length field used to
    provide, at a minimum, the authority identity of the PAC issuer.
    Other useful but not mandatory information, such as the PAC-Key
    lifetime, may also be conveyed by the PAC issuing server to the
    peer during PAC provisioning or refreshment. PAC-Info is not
    included if the NewSessionTicket message is used to provision the
    PAC.

The use of the PAC is based on the SessionTicket extension defined in
[RFC5077]. The EAP server initiates the TEAP conversation as normal.
Upon receiving the Authority-ID TLV from the server, the peer checks
to see if it has an existing valid PAC-Key and PAC-Opaque for the
server. If it does, then it obtains the PAC-Opaque and puts it in
the SessionTicket extension in the ClientHello. It is RECOMMENDED in
TEAP that the peer include an empty Session ID in a ClientHello
containing a PAC-Opaque. This version of TEAP supports the
NewSessionTicket Handshake message as described in [RFC5077] for
distribution of a new PAC, as well as the provisioning of PAC inside
the protected tunnel. If the PAC-Opaque included in the
SessionTicket extension is valid and the EAP server permits the
abbreviated TLS handshake, it will select the cipher suite from
information within the PAC-Opaque and finish with the abbreviated TLS
handshake. If the server receives a Session ID and a PAC-Opaque in
the SessionTicket extension in a ClientHello, it should place the
same Session ID in the ServerHello if it is resuming a session based
on the PAC-Opaque. The conversation then proceeds as described in
[RFC5077] until the handshake completes or a fatal error occurs.
After the abbreviated handshake completes, the peer and the server
are ready to commence Phase 2.

3.2.3. Transition between Abbreviated and Full TLS Handshake

If session resumption based on server-side or client-side state
fails, the server can gracefully fall back to a full TLS handshake.
If the ServerHello received by the peer contains an empty Session ID
or a Session ID that is different than in the ClientHello, the server
may fall back to a full handshake. The peer can distinguish the
server's intent of negotiating full or abbreviated TLS handshake by
checking the next TLS handshake messages in the server response to
the ClientHello. If ChangeCipherSpec follows the ServerHello in
response to the ClientHello, then the server has accepted the session
resumption and intends to negotiate the abbreviated handshake.
Otherwise, the server intends to negotiate the full TLS handshake. A
peer can request for a new PAC to be provisioned after the full TLS
handshake and mutual authentication of the peer and the server. In
order to facilitate the fallback to a full handshake the peer SHOULD
include cipher suites that allow for a full handshake and possibly
PAC provisioning so the server can select one of these in case

session resumption fails. An example of the transition is shown in
Appendix C.

3.3. TEAP Authentication Phase 2: Tunneled Authentication

The second portion of the TEAP Authentication occurs immediately
after successful completion of Phase 1. Phase 2 occurs even if both
peer and authenticator are authenticated in the Phase 1 TLS
negotiation. Phase 2 MUST NOT occur if the Phase 1 TLS handshake
fails. Phase 2 consists of a series of requests and responses
encapsulated in TLV objects defined in Section 4.2. Phase 2 MUST
always end with a crypto-binding TLV exchange descried in
Section 4.2.13 and a protected termination exchange described in
Section 3.3.3. The TLV exchange may include the execution of zero or
more EAP methods within the protected tunnel as described in
Section 3.3.1. A server MAY proceed directly to the protected
termination exchange if it does not wish to request further
authentication from the peer. However, the peer and server must not
assume that either will skip inner EAP methods or other TLV
exchanges. The peer may have roamed to a network that requires
conformance with a different authentication policy, or the peer may
request the server take additional action (e.g., channel binding)
through the use of the Request-Action TLV as defined in
Section 4.2.9.

3.3.1. EAP Sequences

EAP [RFC3748] prohibits use of multiple authentication methods within
a single EAP conversation in order to limit vulnerabilities to man-
in-the-middle attacks. TEAP addresses man-in-the-middle attacks
through support for cryptographic protection of the inner EAP
exchange and cryptographic binding of the inner authentication
method(s) to the protected tunnel. EAP methods are executed serially
in a sequence. This version of TEAP does not support initiating
multiple EAP methods simultaneously in parallel. The methods need
not be distinct. For example, EAP-TLS could be run twice as an inner
method, first using machine credentials followed by a second instance
using user credentials.

EAP method messages are carried within EAP-Payload TLVs defined in
Section 4.2.10. If more than one method is going to be executed in
the tunnel, then upon method completion, the server MUST send an
Intermediate-Result TLV indicating the result. The peer MUST respond
to the Intermediate-Result TLV indicating its result. If the result
indicates success, the Intermediate-Result TLV MUST be accompanied by
a Crypto-Binding TLV. The Crypto-Binding TLV is further discussed in
Section 4.2.13 and Section 5.3. The Intermediate-Result TLVs can be
included with other TLVs such as EAP-Payload TLVs starting a new EAP

conversation or with the Result TLV used in the protected termination
exchange.

If both peer and server indicate success, then the method is
considered complete. If either indicates failure, then the method is
considered failed. The result of failure of an EAP method does not
always imply a failure of the overall authentication. If one
authentication method fails, the server may attempt to authenticate
the peer with a different method.

3.3.2. Optional Password Authentication

The use of EAP-FAST-GTC as defined in RFC 5421 [RFC5421] is not
recommended with TEAPv1. Implementations should instead make use of
the password authentication TLVs defined in this specification. The
authentication server initiates password authentication by sending a
Basic-Password-Auth-Req TLV defined in Section 4.2.14. If the peer
wishes to participate in password authentication then it responds
with a Basic-Password-Auth-Resp TLV as defined in Section 4.2.15 that
contains the username and password. If it does not wish to perform
password authentication then it responds with a NAK TLV indicating
the rejection of the Basic-Password-Auth-Req TLV. Upon receiving the
response, the server indicates the success or failure of the exchange
using an Intermediate-Result TLV. Multiple roundtrips of password
authentication requests and responses MAY be used to support some
"housecleaning" functions such as password change, change pin, etc.
before a user is authenticated.

3.3.3. Protected Termination and Acknowledged Result Indication

A successful TEAP Phase 2 conversation MUST always end in a
successful Crypto-Binding TLV and Result TLV exchange. A TEAP server
may initiate the Crypto-Binding TLV and Result TLV exchange without
initiating any EAP conversation in TEAP Phase 2. After the final
Result TLV exchange, the TLS tunnel is terminated and a clear text
EAP-Success or EAP-Failure is sent by the server. Peers implementing
TEAP MUST NOT accept a clear-text EAP success or failure packet prior
to the peer and server reaching synchronized protected result
indication.

The Crypto-Binding TLV exchange is used to prove that both the peer
and server participated in the tunnel establishment and sequence of
authentications. It also provides verification of the TEAP type,
version negotiated, outer TLVs exchanged before the TLS tunnel
establishment. The Crypto-Binding TLV MUST be exchanged and verified
before the final Result TLV exchange, regardless whether there is an
inner EAP method authentication or not. It MUST be included with the
Intermediate-Result TLV to perform Cryptographic Binding after each

successful EAP method in a sequence of EAP methods, before proceeding
with another inner EAP method. The server may send the final Result
TLV along with an Intermediate-Result TLV and a Crypto-Binding TLV to
indicate its intention to end the conversation. If the peer requires
nothing more from the server, it will respond with a Result TLV
indicating success accompanied by a Crypto-Binding TLV and
Intermediate-Result TLV if necessary. The server then tears down the
tunnel and sends a clear text EAP-Success or EAP-Failure.

If the peer receives a Result TLV indicating success from the server,
but its authentication policies are not satisfied (for example it
requires a particular authentication mechanism be run or it wants to
request a PAC), it may request further action from the server using
the Request-Action TLV. The Request-Action TLV is sent with a Status
field indicating what EAP Success/Failure result the peer would
expect if the requested action is not granted. The value of the
Action field indicates what the peer would like to do next. The
format and values for the Request-Action TLV are defined in
Section 4.2.9.

Upon receiving the Request-Action TLV the server may process the
request or ignore it, based on its policy. If the server ignores the
request, it proceeds with termination of the tunnel and send the
clear text EAP Success or Failure message based on the value of the
peer's result TLV. If the server honors and processes the request,
it continues with the requested action. The conversation completes
with a Result TLV exchange. The Result TLV may be included with the
TLV that completes the requested action.

Error handling for Phase 2 is discussed in Section 3.6.2.

3.4. Determining Peer-Id and Server-Id

The Peer-Id and Server-Id [RFC5247] may be determined based on the
types of credentials used during either the TEAP tunnel creation or
authentication. In the case of multiple peer authentications, all
authenticated peer identities need to be exported.

When X.509 certificates are used for peer authentication, the Peer-Id
is determined by the subject or subjectAltName fields in the peer
certificate. As noted in [RFC5280]:

   The subject field identifies the entity associated with the public
   key stored in the subject public key field. The subject name MAY
   be carried in the subject field and/or the subjectAltName
   extension.... If subject naming information is present only in
   the subjectAltName extension (e.g., a key bound only to an email
   address or URI), then the subject name MUST be an empty sequence

and the subjectAltName extension MUST be critical.

Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN).

If an inner EAP method is run, then the Peer-Id is obtained from the inner method.

When the server uses an X.509 certificate to establish the TLS tunnel, the Server-Id is determined in a similar fashion as stated above for the Peer-Id; e.g., the subject or subjectAltName field in the server certificate defines the Server-Id.

3.5. TEAP Session Identifier

The EAP session identifier [RFC5247] is constructed using the tls_unique from the TLS tunnel establishment as defined by [RFC5929]. The Session-Id is defined as follows:

Session-Id = teap_type || tls_unique

where teap_type is the EAP method type assigned to TEAP.

tls_unique = tls_unique for the phase 1 outer tunnel as defined by [RFC5929].

3.6. Error Handling

TEAP uses the following error handling rules summarized below:

1. Errors in the TLS layer are communicated via TLS alert messages in all phases of TEAP.

2. The Intermediate-Result TLVs carry success or failure indications of the individual EAP methods in TEAP Phase 2. Errors within the EAP conversation in Phase 2 are expected to be handled by individual EAP methods.

3. Violations of the TLV rules are handled using Result TLVs together with Error TLVs.

4. Tunnel compromised errors (errors caused by Crypto-Binding failed or missing) are handled using Result TLVs and Error TLVs.

3.6.1. TLS Layer Errors

   If the TEAP server detects an error at any point in the TLS Handshake
   or the TLS layer, the server SHOULD send a TEAP request encapsulating
   a TLS record containing the appropriate TLS alert message rather than
   immediately terminating the conversation so as to allow the peer to
   inform the user of the cause of the failure and possibly allow for a
   restart of the conversation. The peer MUST send a TEAP response to
   an alert message. The EAP-Response packet sent by the peer may
   encapsulate a TLS ClientHello handshake message, in which case the
   TEAP server MAY allow the TEAP conversation to be restarted, or it
   MAY contain a TEAP response with a zero-length message, in which case
   the server MUST terminate the conversation with an EAP-Failure
   packet. It is up to the TEAP server whether to allow restarts, and
   if so, how many times the conversation can be restarted. A TEAP
   server implementing restart capability SHOULD impose a limit on the
   number of restarts, so as to protect against denial-of-service
   attacks.

   If the TEAP peer detects an error at any point in the TLS layer, the
   TEAP peer should send a TEAP response encapsulating a TLS record
   containing the appropriate TLS alert message. The server may restart
   the conversation by sending an TEAP request packet encapsulating the
   TLS HelloRequest handshake message. The peer may allow the TEAP
   conversation to be restarted or it may terminate the conversation by
   sending an TEAP response with an zero-length message.

3.6.2. Phase 2 Errors

   Any time the peer or the server finds a fatal error outside of the
   TLS layer during Phase 2 TLV processing, it MUST send a Result TLV of
   failure and an Error TLV with the appropriate error code. For errors
   involving the processing of the sequence of exchanges, such as a
   violation of TLV rules (e.g., multiple EAP-Payload TLVs), the error
   code is Unexpected_TLVs_Exchanged. For errors involving a tunnel
   compromise, the error-code is Tunnel_Compromise_Error. Upon sending
   a Result TLV with a fatal Error TLV the sender terminates the TLS
   tunnel. Note that a server will still wait for a message from the
   peer after it sends a failure, however the server does not need to
   process the contents of the response message.

   If a server receives a Result TLV of failure with a fatal Error TLV,
   it SHOULD send a clear text EAP-Failure. If a peer receives a Result
   TLV of failure, it MUST respond with a Result TLV indicating failure.
   If the server has sent a Result TLV of failure, it ignores the peer
   response, and it SHOULD send a clear text EAP-Failure.

3.7. Fragmentation

   A single TLS record may be up to 16384 octets in length, but a TLS
   message may span multiple TLS records, and a TLS certificate message
   may in principle be as long as 16 MB. This is larger than the
   maximum size for a message on most media types, therefore it is
   desirable to support fragmentation. Note that in order to protect
   against reassembly lockup and denial-of-service attacks, it may be
   desirable for an implementation to set a maximum size for one such
   group of TLS messages. Since a typical certificate chain is rarely
   longer than a few thousand octets, and no other field is likely to be
   anywhere near as long, a reasonable choice of maximum acceptable
   message length might be 64 KB. This is still a fairly large message
   packet size so an TEAP implementation MUST provide its own support
   for fragmentation and reassembly.

   Since EAP is a lock-step 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.

   TEAP fragmentation support is provided through the addition of flag
   bits within the EAP-Response and EAP-Request packets, as well as a
   TLS Message Length field of four octets. Flags include the Length
   included (L), More fragments (M), and TEAP 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 TEAP start message
   sent from the EAP server to the peer. The TLS Message Length field
   is four octets, and provides the total length of the TLS message or
   set of messages that is being fragmented; this simplifies buffer
   allocation.

   When a TEAP peer receives an EAP-Request packet with the M bit set,
   it MUST respond with an EAP-Response with EAP-Type of TEAP 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 TEAP server receives an EAP-Response with the M
   bit set, it must respond with an EAP-Request with EAP-Type of TEAP
   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.

3.8. PAC Provisioning

To request provisioning of a PAC, a peer sends a PAC TLV as defined
in Section 4.2.12 containing a PAC Attribute as defined in
Section 4.2.12.1 of PAC Type set to the appropriate value. The
request MAY be issued after the peer has determined that it has
successfully authenticated the EAP server and validated the Crypto-
Binding TLV as defined in Section 4.2.13 to ensure that the TLS
tunnel's integrity is intact. The peer MUST send separate PAC TLVs
for each type of PAC it wants to be provisioned. Multiple PAC TLVs
can be sent in the same packet or different packets. The EAP server
will send the PACs after its internal policy has been satisfied, or
it MAY ignore the request or request additional authentications if
its policy dictates. If a peer receives a PAC with an unknown type,
it MUST ignore it.

A PAC-TLV containing PAC-Acknowledge attribute MUST be sent by the
peer to acknowledge the receipt of the Tunnel PAC. A PAC-TLV
containing PAC-Acknowledge attribute MUST NOT be used by the peer to
acknowledge the receipt of other types of PACs.

3.9. Certificate Provisioning Within the Tunnel

Provisioning of a peer's certificate is supported in TEAP by
performing the Simple PKI Request/Response from [RFC5272] using
PKCS#10 and PKCS#7 TLVs, respectively. A peer sends the Simple PKI
Request using a PKCS#10 CertificateRequest [RFC2986] encoded into the
body of a PKCS#10 TLV (see section Section 4.2.17). The TEAP Server
issues a Simple PKI Response using a PKCS#7 [RFC2315] degenerate
"certs-only" message encoded into the body of a PKCS#7 TLV (see
section Section 4.2.16).

In order to provide linking identity and proof-of-possession by
including information specific to the current authenticated TLS
session within the signed certification request, the client
generating the request SHOULD obtain the tls-unique value as defined
in Channel Bindings for TLS [RFC5929] from the TLS subsystem, encode
it using base64 encoding, and place the resulting string in the
certification request challenge password field. The tls-unique value
used MUST be from the first TLS handshake. TEAP client and server
must use their tls-unique implementation specific synchronization

methods to obtain this first tls-unique value. The server SHOULD
verify the tls-unique information. This ensures that the
authenticated TEAP client is in possession of the private key used to
sign the certification request.

The Simple PKI Request/Response generation and processing rules of
[RFC5272] SHALL apply to TEAP, with the exception of error
conditions. In the event of an error, the TEAP Server SHOULD respond
with an Error TLV using the most descriptive error code possible; it
MAY ignore the PKCS#10 request which generated the error.

3.10. Server Unauthenticated Provisioning Mode

In Server Unauthenticated Provisioning Mode, an unauthenticated
tunnel is established in phase 1 and the peer and server negotiate an
EAP method in phase 2 that supports mutual authentication and key
derivation that is resistant to attacks such as Man-in-the-middle and
dictionary attacks. This provisioning mode enables the bootstrapping
of peers when the peer lacks a strong credential usable for mutual
authentication with the server during phase 1.

Upon successful completion of the EAP method in phase 2, the peer and
server exchange a Crypto-Binding TLV to bind the inner method with
the outer tunnel and ensure that a man-in-the-middle attack has not
been attempted.

Support for the Server Unauthenticated Provisioning Mode is optional.
The cipher suite TLS_DH_anon_WITH_AES_128_CBC_SHA is RECOMMENDED when
using server unauthenticated mode, but other anonymous ciphersuites
MAY be supported as long as the TLS pre-master secret is generated
from contribution from both peers. Phase 2 EAP methods used in
Server Unauthenticated Provisioning Mode MUST provide mutual
authentication, key generation, and be resistant to dictionary
attack. Example inner methods include EAP-pwd [RFC5931] and EAP-EKE
[RFC6124].

4. Message Formats

The following sections describe the message formats used in TEAP.
The fields are transmitted from left to right in network byte order.

4.1. TEAP Message Format

A summary of the TEAP Request/Response packet format is shown below.

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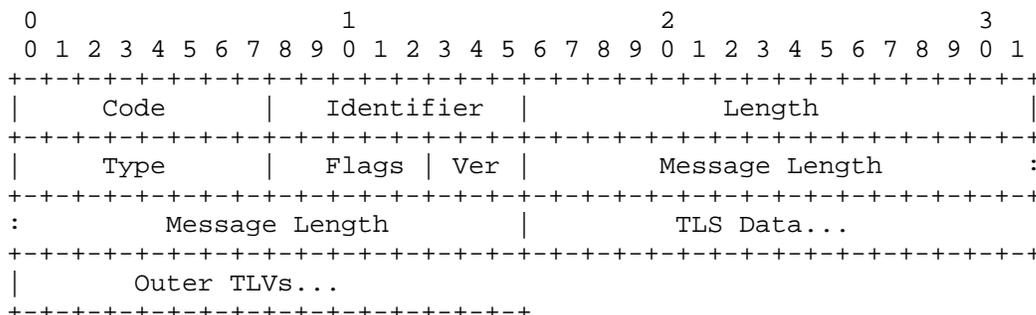
   Code

      The code field is one octet in length defined as follows:

      1 Request

      2 Response

   Identifier

      The Identifier field is one octet and aids in matching
      responses with requests. The Identifier field MUST be changed
      on each Request packet. The Identifier field in the Response
      packet MUST match the Identifier field from the corresponding
      request.

   Length

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

   Type

      TBD for TEAP

Flags

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

L   Length included; set to indicate the presence of the four
    octet Message Length field

M   More fragments; set on all but the last fragment

S   TEAP start; set in a TEAP Start message

R   Reserved (must be zero)

Ver

   This field contains the version of the protocol. This document
   describes version 1 (001 in binary) of TEAP.

Message Length

   The Message Length field is four octets, and is present only if
   the L bit is set. This field provides the total length of the
   message that may be fragmented over the data fields of multiple
   packets.

TLS Data

   When the Data field is present, it consists of an encapsulated
   TLS packet in TLS record format. A TEAP packet with Flags and
   Version fields, but with zero length TLS data field, is used to
   indicate TEAP acknowledgement for either a fragmented message,
   a TLS Alert message or a TLS Finished message.

Outer TLVs

   The Outer-TLVs consist of the optional data used to help
   establishing the TLS tunnel in TLV format. They are only
   allowed in the first two messages in the TEAP protocol. That
   is the first EAP server to peer message and first peer to EAP
   server message. The start of the Outer-TLV can be derived from
   the EAP Length field and Message Length field.

4.2. TEAP TLV Format and Support

   The TLVs defined here are standard Type-Length-Value (TLV) objects.
   The TLV objects could be used to carry arbitrary parameters between
   EAP peer and EAP server within the protected TLS tunnel.

   The EAP peer may not necessarily implement all the TLVs supported by
   the EAP server. To allow for interoperability, TLVs are designed to
   allow an EAP server to discover if a TLV is supported by the EAP
   peer, using the NAK TLV. The mandatory bit in a TLV indicates
   whether support of the TLV is required. If the peer or server does
   not support a TLV marked mandatory, then it MUST send a NAK TLV in
   the response, and all the other TLVs in the message MUST be ignored.
   If an EAP peer or server finds an unsupported TLV that is marked as
   optional, it can ignore the unsupported TLV. It MUST NOT send an NAK
   TLV for a TLV that is not marked mandatory. If all TLVs in a message
   are marked optional and none are understood by the peer, then an
   EMPTY TEAP Phase 2 message must still be sent to the other side in
   order to continue the conversation.

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

   EAP implementations compliant with this specification MUST support
   TLV exchanges, as well as the processing of mandatory/optional
   settings on the TLV. Implementations conforming to this
   specification MUST support the following TLVs:

      Authority-ID TLV

      Identity-Type TLV

      Result TLV

      NAK TLV

      Error TLV

      Request-Action TLV

      EAP-Payload TLV

      Intermediate-Result TLV

      Crypto-Binding TLV

      Basic-Password-Auth-Req TLV

      Basic-Password-Auth-Resp TLV

4.2.1. General TLV Format

   TLVs are defined as described 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|            TLV Type         |            Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Value...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
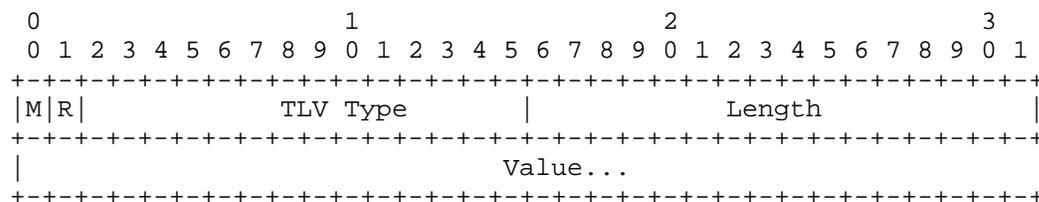
      M

         0 Optional TLV

         1 Mandatory TLV

      R

         Reserved, set to zero (0)

      TLV Type

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

         0 Unassigned

         1 Authority-ID TLV (Section 4.2.2)

2 Identity-Type TLV (Section 4.2.3)

3 Result TLV (Section 4.2.4)

4 NAK TLV (Section 4.2.5)

5 Error TLV (Section 4.2.6)

6 Channel-Binding TLV (Section 4.2.7)

7 Vendor-Specific TLV (Section 4.2.8)

8 Request-Action TLV (Section 4.2.9)

9 EAP-Payload TLV (Section 4.2.10)

10 Intermediate-Result TLV (Section 4.2.11)

11 PAC TLV (Section 4.2.12)

12 Crypto-Binding TLV (Section 4.2.13)

13 Basic-Password-Auth-Req TLV (Section 4.2.14)

14 Basic-Password-Auth-Resp TLV (Section 4.2.15)

15 PKCS#7 TLV (Section 4.2.16)

16 PKCS#10 TLV (Section 4.2.17)

17 Server-Trusted-Root TLV (Section 4.2.18)

Length

   The length of the Value field in octets.

Value

   The value of the TLV.

4.2.2. Authority-ID TLV

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|        TLV Type         |              Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ID...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
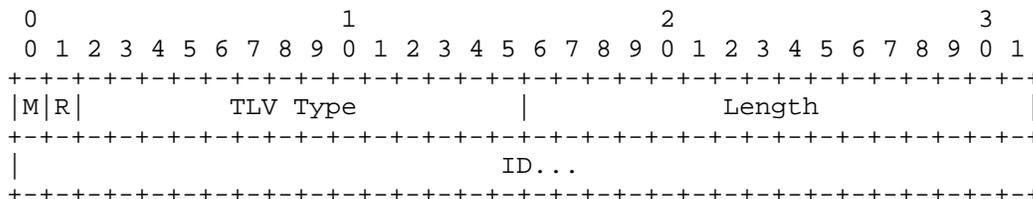
        M

           Mandatory, set to (0)

        R

           Reserved, set to zero (0)

        Type

           The Type field is two octets. It is set to 1 for Authority ID

        Length

           The Length filed is two octets, which contains the length of
           the ID field in octets.

        ID

           Hint of the identity of the server, to help the peer to match
           the credentials available for the server. It should be unique
           across the deployment.

4.2.3. Identity-Type TLV

   The Identity-Type TLV allows an EAP server to send a hint to help the
   EAP peer select the right type of identity; for example; user or
   machine. TEAPv1 implementations MUST support this TLV. If the
   Identity-Type field does not contain one of the known values or if
   the EAP peer does not have an identity corresponding to the identity-
   type, then the peer SHOULD respond with an Identity-Type TLV with the
   one of available identity types. If the server receives an identity

type in the response that does not match the requested type, then the
peer does not possess the requested credential type and the server
SHOULD proceed with authentication for the credential type proposed
by the peer or proceed with requesting another credential type, or
simply apply the network policy based on the configured policy, e.g.,
sending Result TLV with Failure.

The Identity-Type TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|         TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identity-Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   M

      0 (Optional)

   R
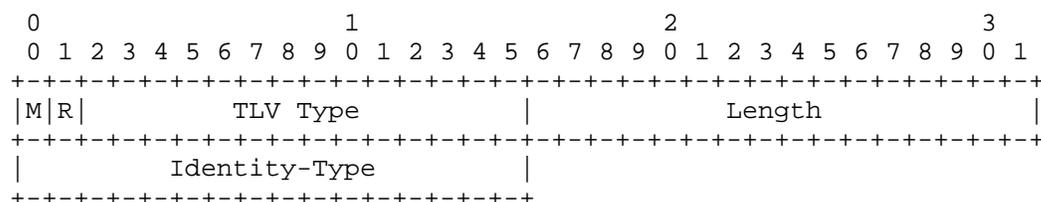
      Reserved, set to zero (0)

   TLV Type

      2 for Identity-Type TLV

   Length

      2

   Identity-Type

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

      1 User

         2 Machine

4.2.4. Result TLV

   The Result TLV provides support for acknowledged success and failure
   messages for protected termination within TEAP. If the Status field
   does not contain one of the known values, then the peer or EAP server
   MUST treat this as a fatal error of Unexpected_TLVs_Exchanged. The
   behavior of the Result TLV is further discussed in Section 3.3.3 and
   Section 3.6.2. A Result TLV indicating failure MUST NOT be
   accompanied by the following TLVs: NAK, EAP-Payload TLV, or Crypto-
   Binding TLV. The Result TLV is defined as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |M|R|        TLV Type           |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            Status             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


     M

        Mandatory, set to one (1)

     R
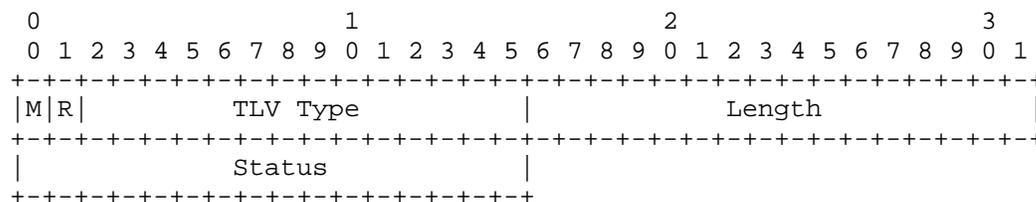
        Reserved, set to zero (0)

     TLV Type

        3 for Result TLV

     Length

        2

     Status

        The Status field is two octets. Values include:

          1 Success

          2 Failure

4.2.5. NAK TLV

   The NAK TLV allows a peer to detect TLVs that are not supported by
   the other peer. A TEAP packet can contain 0 or more NAK TLVs. A NAK
   TLV should not be accompanied by other TLVs. A NAK TLV MUST NOT be
   sent in response to a message containing a Result TLV, instead a
   Result TLV of failure should be sent indicating failure and an Error
   TLV of Unexpected_TLVs_Exchanged. The NAK TLV is defined as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |M|R|         TLV Type          |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                            Vendor-Id                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          NAK-Type          |             TLVs...
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


   M

      Mandatory, set to one (1)

   R
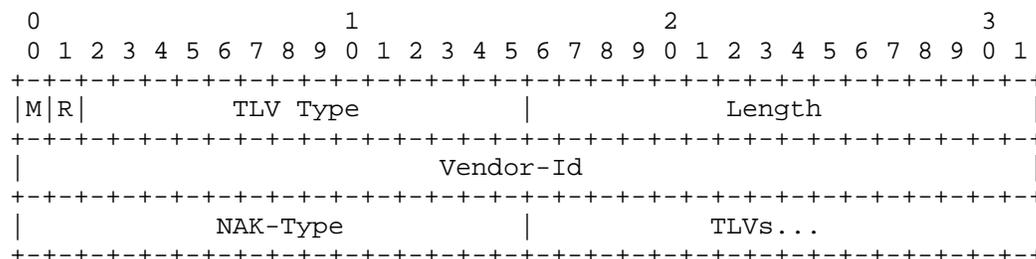
      Reserved, set to zero (0)

   TLV Type

      4 for NAK TLV

   Length

      >=6

Vendor-Id

The Vendor-Id field is four octets, and contains the Vendor-Id
of the TLV that was not supported. The high-order octet is 0
and the low-order three octets are the Structure of Management
Information (SMI) Network Management Private Enterprise Code of
the Vendor in network byte order. The Vendor-Id field MUST be
zero for TLVs that are not Vendor-Specific TLVs.

NAK-Type

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

TLVs

This field contains a list of zero or more TLVs, each of which
MUST NOT have the mandatory bit set. These optional TLVs are
for future extensibility to communicate why the offending TLV
was determined to be unsupported.

4.2.6. Error TLV

The Error TLV allows an EAP peer or server to indicate errors to the
other party. A TEAP packet can contain 0 or more Error TLVs. The
Error-Code field describes the type of error. Error Codes 1-999
represent successful outcomes (informative messages), 1000-1999
represent warnings, and codes 2000-2999 represent fatal errors. A
fatal Error TLV MUST be accompanied by a Result TLV indicating
failure and the conversation must be terminated as described in
Section 3.6.2. The Error TLV is defined as follows:


```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|          TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Error-Code                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

M

   Mandatory, set to one (1)

R
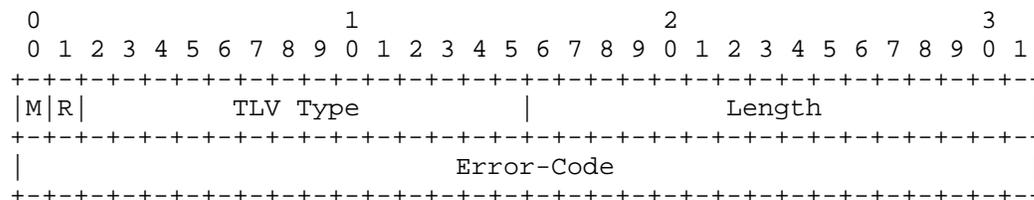
   Reserved, set to zero (0)

TLV Type

   5 for Error TLV

Length

   4

Error-Code

   The Error-Code field is four octets. Currently defined values
   for Error-Code include:

      2001 Tunnel_Compromise_Error

      2002 Unexpected_TLVs_Exchanged

      2003 Unsupported_Algorithm_In_CertificateSigning_Request

      2004 Unsupported_Extension_In_CertificateSigning_Request

      2005 Bad_Identity_In_CertificateSigning_Request

      2006 Bad_CertificateSigning_Request

      2007 Internal_CA_Error

      2008 General_PKI_Error

4.2.7. Channel-Binding TLV

   The Channel-Binding TLV provides a mechanism for carrying channel
   binding data from the peer to the EAP server and a channel binding
   response from the EAP server to the peer as described in
   [I-D.ietf-emu-chbind]. TEAPv1 implementations MAY support this TLV,
   which cannot be responded to with a NAK TLV. If the Channel-Binding
   data field does not contain one of the known values or if the EAP
   server does not support this TLV, then the server MUST ignore the
   value. The Channel-Binding TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|        TLV Type           |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Data ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

      M

         0 (Optional)

      R
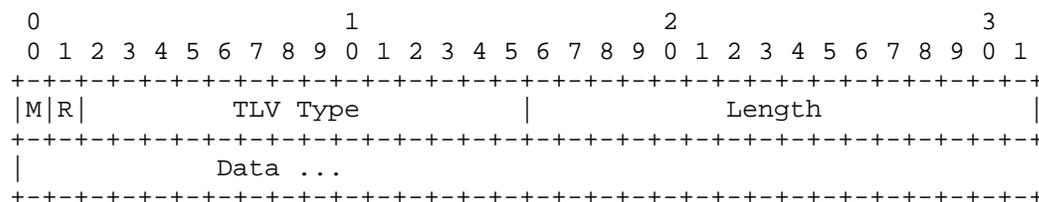
         Reserved, set to zero (0)

      TLV Type

         6 for Channel-Binding TLV

      Length

         variable

      Data

         The data field contains channel binding data defined in
         [I-D.ietf-emu-chbind].

4.2.8. Vendor-Specific TLV

   The Vendor-Specific TLV is available to allow vendors to support
   their own extended attributes not suitable for general usage. A
   Vendor-Specific TLV attribute can contain one or more TLVs, referred
   to as Vendor TLVs. The TLV-type of a Vendor-TLV is defined by the
   vendor. All the Vendor TLVs inside a single Vendor-Specific TLV
   belong to the same vendor. There can be multiple Vendor-Specific
   TLVs from different vendors in the same message.

   Vendor TLVs may be optional or mandatory. Vendor TLVs sent with
   Result TLVs MUST be marked as optional.

   The Vendor-Specific TLV is defined as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |M|R|         TLV Type          |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Vendor-Id                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Vendor TLVs....                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
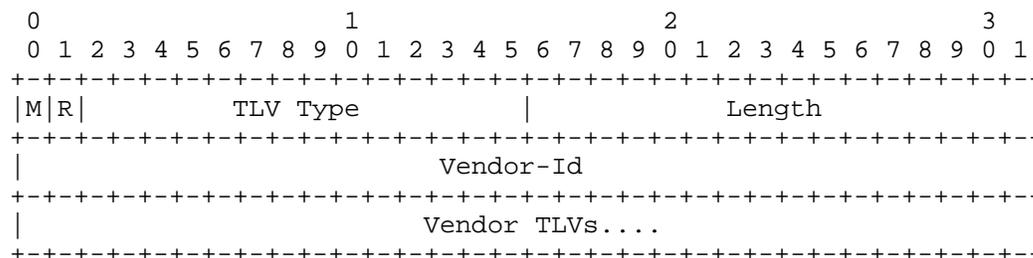
        M

           0 or 1

        R

           Reserved, set to zero (0)

        TLV Type

           7 for Vendor Specific TLV

        Length

           4 + cumulative length of all included Vendor TLVs

Vendor-Id

The Vendor-Id field is four octets, and contains the Vendor-Id
of the TLV. The high-order octet is 0 and the low-order 3
octets are the SMI Network Management Private Enterprise Code
of the Vendor in network byte order.

Vendor TLVs

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

4.2.9. Request-Action TLV

The Request-Action TLV MAY be sent by the peer in response to a
server's successful Result TLV. It allows the peer to request the
EAP server to negotiate additional EAP methods or process TLVs
specified in the response packet. The server MAY ignore this TLV.

The peer MAY send multiple Request-Action TLVs to the server. Two
Request TLVs MUST NOT occur in the same response packet if they have
the same Status value. The order of processing multiple Request TLVs
is implementation dependent. If the server process the optional
(non-fatal) items first, it is possible that the fatal items will
disappear at a later time. If the server process the fatal items
first, the communication time will be shorter.

The client MAY return a new set of Request-Action TLVs after one or
more of the requested items has been processed and the server has
signaled it wants to end the EAP conversation.

The Request-Action TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|         TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Status    |     Action    |             TLVs....          
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

M

Mandatory set to one (1)

R
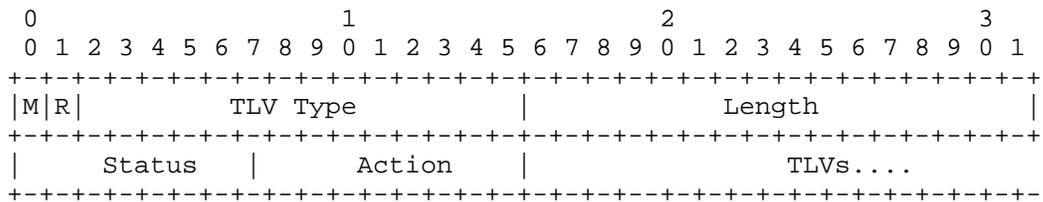
    Reserved, set to zero (0)

TLV Type

    8 for Request-Action TLV

Length

    2 + cumulative length of all included TLVs

Status

    The Status field is one octet. This indicates the result if
    the server does not process the action requested by the peer.
    Values include:

    1 Success

    2 Failure

Action

    The Action field is one octet. Values include:

    1 Process-TLV

    2 Negotiate-EAP

TLVs

    This field is of indefinite length. It contains TLVs that the
    peer wants the server to process.

4.2.10. EAP-Payload TLV

To allow piggybacking an EAP request or response with other TLVs, the
EAP-Payload TLV is defined, which includes an encapsulated EAP packet
and a list of optional TLVs. The optional TLVs are provided for
future extensibility to provide hints about the current EAP

authentication. Only one EAP-Payload TLV is allowed in a message.
The EAP-Payload TLV is defined as follows:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|           TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          EAP packet...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            TLVs...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   M

      Mandatory, set to (1)

   R
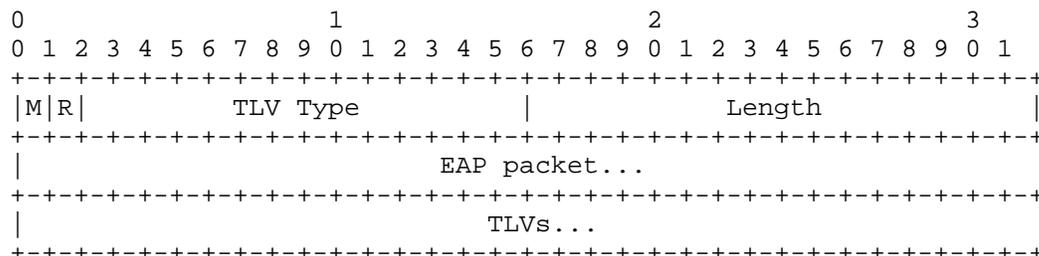
      Reserved, set to zero (0)

   TLV Type

      9 for EAP-Payload TLV

   Length

      length of embedded EAP packet + cumulative length of additional
      TLVs

   EAP packet

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

    TLVs

      This (optional) field contains a list of TLVs associated with
      the EAP packet field. The TLVs MUST NOT have the mandatory bit

          set. The total length of this field is equal to the Length
          field of the EAP-Payload TLV, minus the Length field in the EAP
          header of the EAP packet field.

4.2.11. Intermediate-Result TLV

   The Intermediate-Result TLV provides support for acknowledged
   intermediate Success and Failure messages between multiple inner EAP
   methods within EAP. An Intermediate-Result TLV indicating success
   MUST be accompanied by a Crypto-Binding TLV. The optional TLVs
   associated with this TLV are provided for future extensibility to
   provide hints about the current result. The Intermediate-Result TLV
   is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|         TLV Type          |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Status            |       TLVs...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

      M

         Mandatory, set to (1)

      R
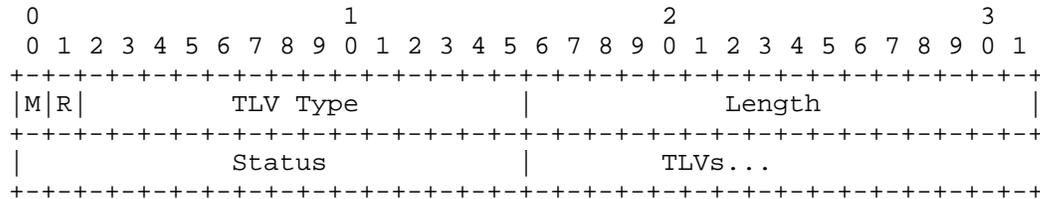
         Reserved, set to zero (0)

      TLV Type

         10 for Intermediate-Result TLV

      Length

         2 + cumulative length of the embedded associated TLVs

      Status

         The Status field is two octets. Values include:

       1 Success

       2 Failure

     TLVs

       This field is of indeterminate length, and contains zero or
       more of the TLVs associated with the Intermediate Result TLV.
       The TLVs in this field MUST NOT have the mandatory bit set.

4.2.12. PAC TLV Format

   The PAC TLV provides support for provisioning the Protected Access
   Credential (PAC) defined within [RFC4851]. The PAC TLV carries the
   PAC and related information within PAC attribute fields.
   Additionally, the PAC TLV MAY be used by the peer to request
   provisioning of a PAC of the type specified in the PAC Type PAC
   attribute. The PAC TLV MUST only be used in a protected tunnel
   providing encryption and integrity protection. A general PAC TLV
   format is defined as follows:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|        TLV Type          |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        PAC Attributes...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


     M
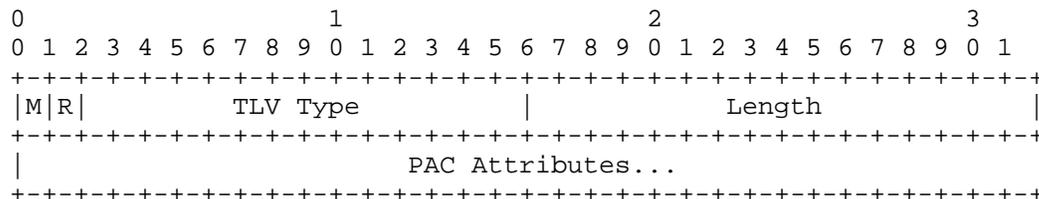
          0 - Non-mandatory TLV
          1 - Mandatory TLV

     R

          Reserved, set to zero (0)

     TLV Type

          11 - PAC TLV

Length

    Two octets containing the length of the PAC attributes
    field in octets.

PAC Attributes

    A list of PAC attributes in the TLV format.

4.2.12.1. Formats for PAC Attributes

Each PAC attribute in a PAC TLV is formatted as a TLV defined as
follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Value...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
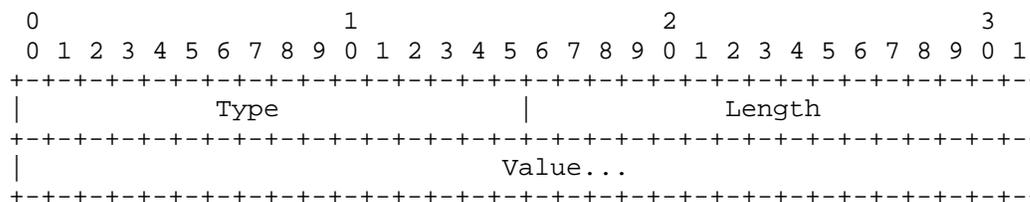
Type

    The Type field is two octets, denoting the attribute type.
    Allocated Types include:

            1 - PAC-Key
            2 - PAC-Opaque
            3 - PAC-Lifetime
            4 - A-ID
            5 - I-ID
            6 - Reserved
            7 - A-ID-Info
            8 - PAC-Acknowledgement
            9 - PAC-Info
            10 - PAC-Type

Length

    Two octets containing the length of the Value field in
    octets.

        Value

           The value of the PAC attribute.

4.2.12.2. PAC-Key

   The PAC-Key is a secret key distributed in a PAC attribute of type
   PAC-Key. The PAC-Key attribute is included within the PAC TLV
   whenever the server wishes to issue or renew a PAC that is bound to a
   key such as a Tunnel PAC. The key is a randomly generated octet
   string, which is 48 octets in length. The generator of this key is
   the issuer of the credential, which is identified by the Authority
   Identifier (A-ID).

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                             Key                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
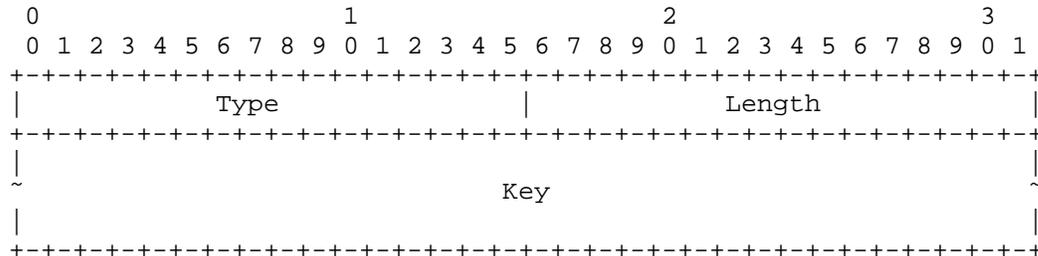
        Type

            1 - PAC-Key

        Length

            2-octet length indicating the length of the key

        Key

            The value of the PAC-Key.

4.2.12.3. PAC-Opaque

   The PAC-Opaque attribute is included within the PAC TLV whenever the
   server wishes to issue or renew a PAC.

   The PAC-Opaque is opaque to the peer and thus the peer MUST NOT
   attempt to interpret it. A peer that has been issued a PAC-Opaque by
   a server stores that data and presents it back to the server
   according to its PAC Type. The Tunnel PAC is used in the ClientHello
   SessionTicket extension field defined in [RFC5077]. If a peer has

opaque data issued to it by multiple servers, then it stores the data
issued by each server separately according to the A-ID. This
requirement allows the peer to maintain and use each opaque datum as
an independent PAC pairing, with a PAC-Key mapping to a PAC-Opaque
identified by the A-ID. As there is a one-to-one correspondence
between the PAC-Key and PAC-Opaque, the peer determines the PAC-Key
and corresponding PAC-Opaque based on the A-ID provided in the TEAP/
Start message and the A-ID provided in the PAC-Info when it was
provisioned with a PAC-Opaque.

The PAC-Opaque attribute format is summarized as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
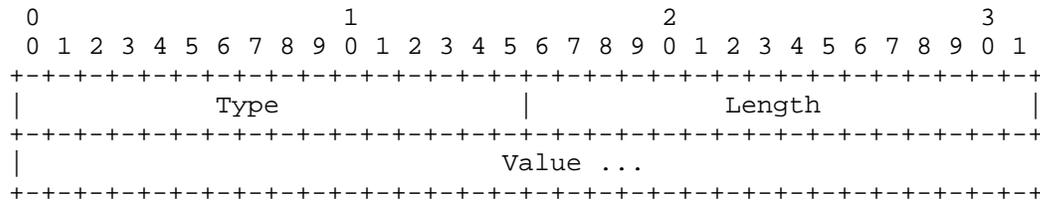
     Type

        2 - PAC-Opaque

     Length

        The Length filed is two octets, which contains the length of
        the Value field in octets.

     Value

        The Value field contains the actual data for the PAC-Opaque.
        It is specific to the server implementation.

4.2.12.4. PAC-Info

   The PAC-Info is comprised of a set of PAC attributes as defined in
   Section 4.2.12.1. The PAC-Info attribute MUST contain the A-ID,
   A-ID-Info, and PAC-Type attributes. Other attributes MAY be included
   in the PAC-Info to provide more information to the peer. The PAC-
   Info attribute MUST NOT contain the PAC-Key, PAC-Acknowledgement,
   PAC-Info, or PAC-Opaque attributes. The PAC-Info attribute is
   included within the PAC TLV whenever the server wishes to issue or
   renew a PAC.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Attributes...                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
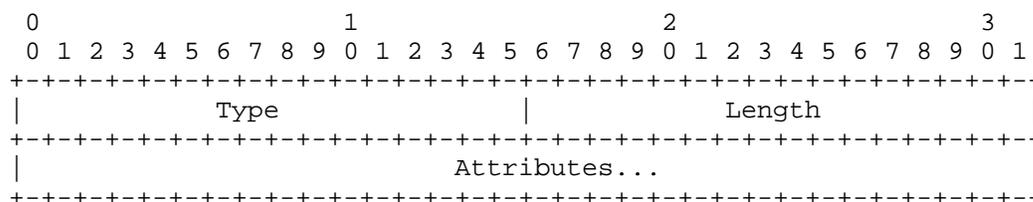
Type

    9 - PAC-Info

Length

    2-octet Length field containing the length of the attributes
    field in octets.

Attributes

    The attributes field contains a list of PAC attributes. Each
    mandatory and optional field type is defined as follows:

    3 - PAC-LIFETIME

       This is a 4-octet quantity representing the expiration time
       of the credential expressed as the number of seconds,
       excluding leap seconds, after midnight UTC, January 1, 1970.
       This attribute MAY be provided to the peer as part of the
       PAC-Info.

    4 - A-ID

       The A-ID is the identity of the authority that issued the
       PAC. The A-ID is intended to be unique across all issuing
       servers to avoid namespace collisions. The A-ID is used by
       the peer to determine which PAC to employ. The A-ID is
       treated as an opaque octet string. This attribute MUST be
       included in the PAC-Info attribute. The A-ID MUST match the
       Authority-ID the server used to establish the tunnel. One
       method for generating the A-ID is to use a high-quality
       random number generator to generate a random number. An
       alternate method would be to take the hash of the public key
       or public key certificate belonging a server represented by
       the A-ID.

5 - I-ID

Initiator identifier (I-ID) is the peer identity associated
with the credential. This identity is derived from the
inner EAP exchange or from the client-side authentication
during tunnel establishment if inner EAP method
authentication is not used. The server employs the I-ID in
the TEAP phase 2 conversation to validate that the same peer
identity used to execute TEAP phase 1 is also used in at
minimum one inner EAP method in TEAP phase 2. If the server
is enforcing the I-ID validation on the inner EAP method,
then the I-ID MUST be included in the PAC-Info, to enable
the peer to also enforce a unique PAC for each unique user.
If the I-ID is missing from the PAC-Info, it is assumed that
the Tunnel PAC can be used for multiple users and the peer
will not enforce the unique-Tunnel-PAC-per-user policy.

7 - A-ID-Info

Authority Identifier Information is intended to provide a
user-friendly name for the A-ID. It may contain the
enterprise name and server name in a human-readable format.
This TLV serves as an aid to the peer to better inform the
end-user about the A-ID. The name is encoded in UTF-8
[RFC3629] format. This attribute MUST be included in the
PAC-Info.

10 - PAC-type

The PAC-Type is intended to provide the type of PAC. This
attribute SHOULD be included in the PAC-Info. If the PAC-
Type is not present, then it defaults to a Tunnel PAC (Type
1).

4.2.12.5. PAC-Acknowledgement TLV

The PAC-Acknowledgement is used to acknowledge the receipt of the
Tunnel PAC by the peer. The peer includes the PAC-Acknowledgement
TLV in a PAC-TLV sent to the server to indicate the result of the
processing and storing of a newly provisioned Tunnel PAC. This TLV
is only used when Tunnel PAC is provisioned.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Result             |
```

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
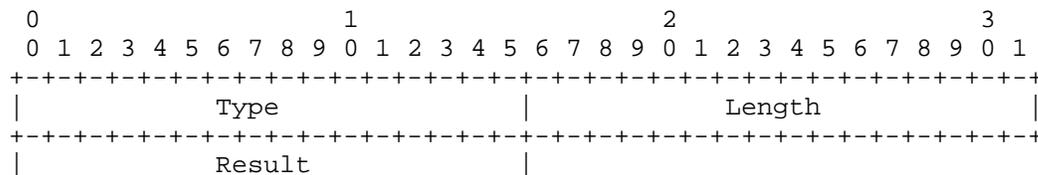
Type

8 - PAC-Acknowledgement

Length

The length of this field is two octets containing a value of 2.

Result

The resulting value MUST be one of the following:

1 - Success
2 - Failure

4.2.12.6. PAC-Type TLV

The PAC-Type TLV is a TLV intended to specify the PAC type. It is
included in a PAC-TLV sent by the peer to request PAC provisioning
from the server. Its format is described below:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Type              |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           PAC Type            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
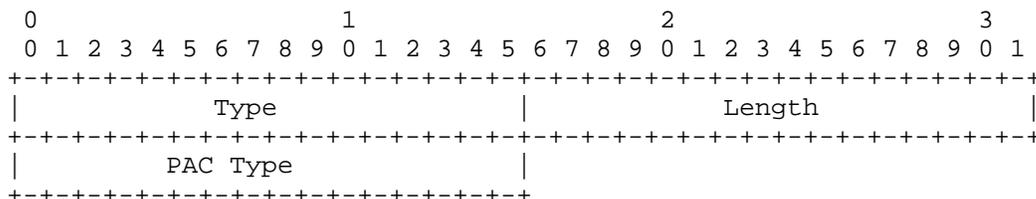

Type

10 - PAC-Type

Length

2-octet Length field with a value of 2

PAC Type

This 2-octet field defines the type of PAC being requested or provisioned. The following values are defined:

1 - Tunnel PAC

4.2.13. Crypto-Binding TLV

The Crypto-Binding TLV is used to prove that both the peer and server participated in the tunnel establishment and sequence of authentications. It also provides verification of the TEAP type, version negotiated, outer TLVs exchanged before the TLS tunnel establishment.

The Crypto-Binding TLV MUST be exchanged and verified before the final Result TLV exchange, regardless whether there is an inner EAP method authentication or not. It MUST be included with the Intermediate-Result TLV to perform Cryptographic Binding after each successful EAP method in a sequence of EAP methods, before proceeding with another inner EAP method.

The Crypto-Binding TLV is valid only if the following checks pass:

o The Crypto-Binding TLV version is supported

o The MAC verifies correctly

o The received version in the Crypto-Binding TLV matches the version sent by the receiver during the EAP version negotiation

o The subtype is set to the correct value

If any of the above checks fails, then the TLV is invalid. An invalid Crypto-Binding TLV is a fatal error and is handled as described in Section 3.6.2

The Crypto-Binding TLV is defined as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |M|R|          TLV Type           |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |    Reserved    |    Version    | Received Ver.|    Sub-Type    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                            Nonce                              ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                        Compound MAC                           ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

M

   Mandatory, set to (1)

R
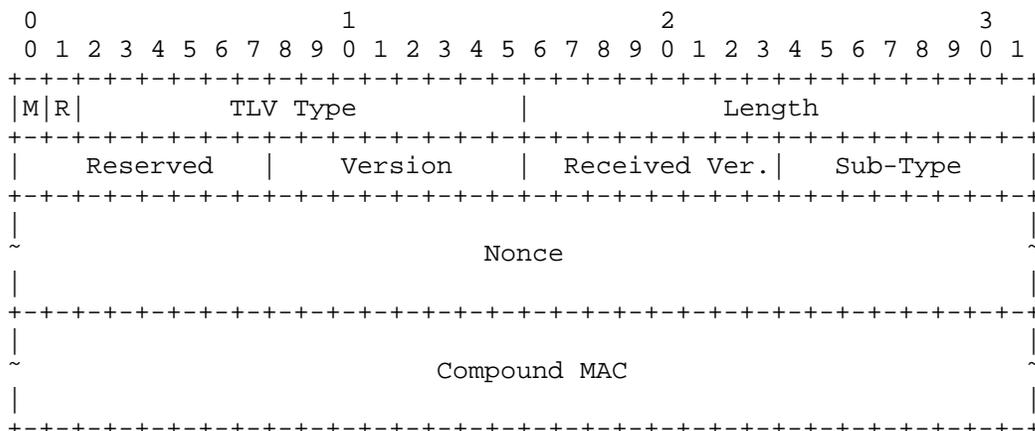
   Reserved, set to zero (0)

TLV Type

   12 for Crypto-Binding TLV

Length

   56

Reserved

   Reserved, set to zero (0)

Version

   The Version field is a single octet, which is set to the
   version of Crypto-Binding TLV the EAP method is using. For an
   implementation compliant with this version of TEAP, the version

number MUST be set to 1.

Received Version

The Received Version field is a single octet and MUST be set to
the EAP version number received during version negotiation.
Note that this field only provides protection against downgrade
attacks, where a version of EAP requiring support for this TLV
is required on both sides.

Sub-Type

The Sub-Type field is one octet. Defined values include

0 Binding Request

1 Binding Response

Nonce

The Nonce field is 32 octets. It contains a 256-bit nonce that
is temporally unique, used for compound MAC key derivation at
each end. The nonce in a request MUST have its least
significant bit set to 0 and the nonce in a response MUST have
the same value as the request nonce except the least
significant bit MUST be set to 1.

Compound MAC

The Compound MAC field is 20 octets. This can be the Server
MAC (B1_MAC) or the Client MAC (B2_MAC). The computation of
the MAC is described in Section 5.3.

4.2.14. Basic-Password-Auth-Req TLV

The Basic-Password-Auth-Req TLV is used by the authentication server
to request a username and password from the peer. It contains an
optional user prompt message for the request. The peer is expected
to obtain the username and password and send them in a Basic-
Password-Auth-Resp TLV.

The Basic-Password-Auth-Req TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|          TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Prompt ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   M

      0 (Optional)

   R
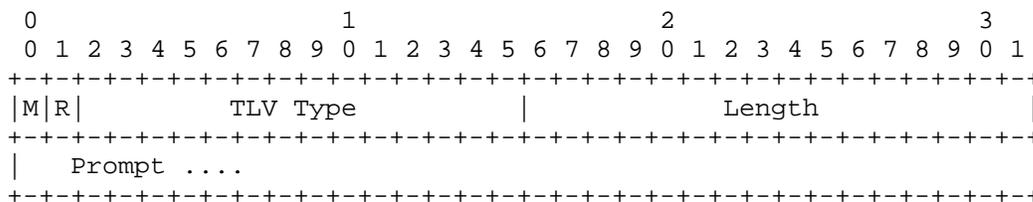
      Reserved, set to zero (0)

   TLV Type

      13 for Basic-Password-Auth-Req TLV

   Length

      variable

   Prompt

      optional user prompt message in UTF-8 format

4.2.15. Basic-Password-Auth-Resp TLV

   The Basic-Password-Auth-Resp TLV is used by the peer to respond to a
   Basic-Password-Auth-Req TLV with a username and password. The TLV
   contains a username and password. The username and password are in
   UTF-8 format.

   The Basic-Password-Auth-Resp TLV is defined as follows:

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |M|R|         TLV Type          |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Userlen     |               Username                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        ...      Username     ...
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Passlen     |             Password                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        ...      Password     ...
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   M

      0 (Optional)

   R
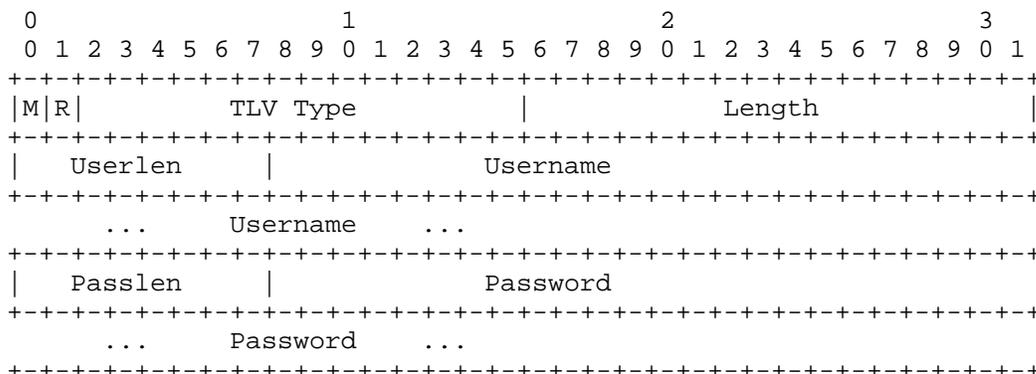
      Reserved, set to zero (0)

   TLV Type

      14 for Basic-Password-Auth-Resp TLV

   Length

      variable

   Userlen

      Length of Username field in octets

   Username

      Username in UTF-8 format

Passlen

Length of Password field in octets

Password

Password in UTF-8 format

4.2.16. PKCS#7 TLV

The PKCS#7 TLV is used by the EAP server to deliver (a)
certificate(s) to the peer. The format consists of a certificate or
certificate chain in a degenerate certificates-only PKCS#7 SignedData
Content as defined in [RFC2311]. When used in response to a Trusted-
Server-Root TLV request from the peer, the EAP server MUST send the
PKCS#7 TLV inside a Trusted-Server-Root TLV. When used in response
to a PKCS#10 certificate enrollment request from the peer, the EAP
server MUST send the PKCS#7 TLV without a Trusted-Server-Root TLV.
The PKCS#7 TLV is always marked as optional, which cannot be
responded to with a NAK TLV. TEAP implementations that support the
Trusted-Server-Root TLV or the PKCS#10 TLV MUST support this TLV.
Peers MUST NOT assume that the certificates in a PKCS#7 TLV are in
any order. TEAP Servers SHOULD include all intermediate certificates
needed to form complete certificate paths to one or more trust
anchors, and not just return the newly issued certificate(s). TEAP
Servers MAY return CRLs in the CRL bag. TEAP Servers MAY return
self-signed certificates. Peers that handle self-signed certificates
or trust anchors MUST NOT implicitly trust these certificates merely
due to their presence in the certificate bag. Note: Peer's are
advised to take great care in deciding whether to use a received
certificate as a trust anchor. The authenticated nature of the
tunnel in which a PKCS#7 bag is received can provide a level of
authenticity to the certificates contained therein. Peers are
advised to take into account the implied authority of the EAP server
and to constrain the trust it can achieve through the trust anchor
received in a PKCS#7 TLV.

The PKCS#7 TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|          TLV Type          |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           PKCS #7 Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

   M

      0 - Optional TLV

   R
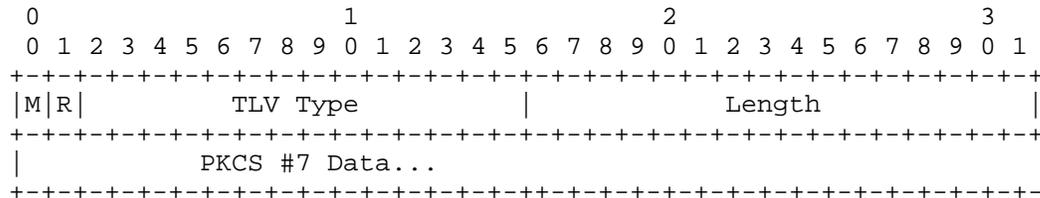
      Reserved, set to zero (0)

   TLV Type

      15 - PKCS#7 TLV

   Length

      The length of the PKCS #7 Data field.

   PKCS #7 Data

      This field contains the X.509 certificate or certificate chain
      in a Certificates-Only PKCS#7 SignedData message.

4.2.17. PKCS#10 TLV

   The PKCS#10 TLV is used by the peer to initiate the "simple PKI"
   Request/Response from [RFC5272]. The format of the request is as
   specified in Section 6.4 of [RFC4945]. The PKCS#10 TLV is always
   marked as optional, which cannot be responded to with a NAK TLV.

   The PKCS#10 TLV is defined as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|        TLV Type          |             Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            PKCS #10 Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

   M

      0 - Optional TLV

   R
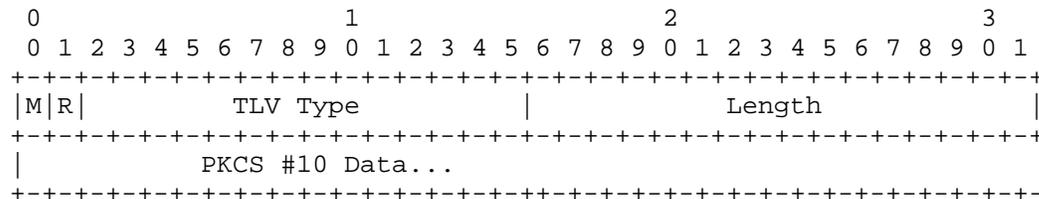
      Reserved, set to zero (0)

   TLV Type

      16 - PKCS#10 TLV

   Length

      The length of the PKCS #10 Data field.

   PKCS #10 Data

      This field contains the PKCS#10 certificate request.

4.2.18. Trusted-Server-Root TLV

   Trusted-Server-Root TLV facilitates the request and delivery of a
   trusted server root certificate. The Trusted-Server-Root TLV can be
   exchanged in regular TEAP authentication mode or provisioning mode.
   The Trusted-Server-Root TLV is always marked as optional, and cannot
   be responded to with a Negative Acknowledgement (NAK) TLV. The
   Trusted-Server-Root TLV MUST only be sent as an inner TLV (inside the
   protection of the tunnel).

   After the peer has determined that it has successfully authenticated
   the EAP server and validated the Crypto-Binding TLV, it MAY send one
   or more Trusted-Server-Root TLVs (marked as optional) to request the
   trusted server root certificates from the EAP server. The EAP server
   MAY send one or more root certificates with a Public Key
   Cryptographic System #7 (PKCS#7) TLV inside Server-Trusted-Root TLV.
   The EAP server MAY also choose not to honor the request.

   The Trusted-Server-Root TLV allows the peer to send a request to the
   EAP server for a list of trusted roots. The server may respond with
   one or more root certificates in PKCS#7 [RFC2315] format.

   If the EAP server sets the credential format to PKCS#7-Server-
   Certificate-Root, then the Trusted-Server-Root TLV should contain the
   root of the certificate chain of the certificate issued to the EAP
   server packaged in a PKCS#7 TLV. If the Server certificate is a
   self-signed certificate, then the root is the self-signed
   certificate.

   If the Trusted-Server-Root TLV credential format contains a value
   unknown to the peer, then the EAP peer should ignore the TLV.

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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|R|        TLV Type         |             Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Credential-Format    |    Cred TLVs...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

M
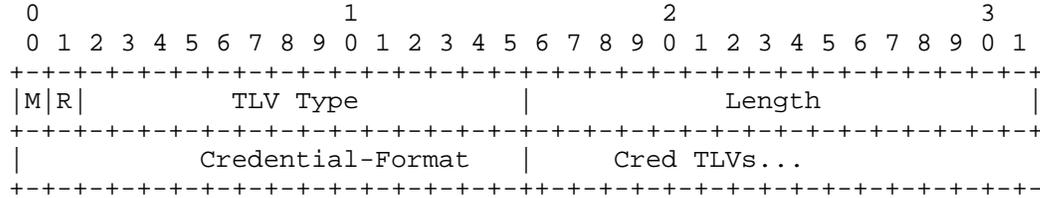
> 0 - Non-mandatory TLV

R

> Reserved, set to zero (0)

TLV Type

> 17 - Trusted-Server-Root TLV [RFC4851]

Length

> >=2 octets

Credential-Format

> The Credential-Format field is two octets. Values include:
>
> > 1 - PKCS#7-Server-Certificate-Root

Cred TLVs

> This field is of indefinite length. It contains TLVs
> associated with the credential format. The peer may leave
> this field empty when using this TLV to request server
> trust roots.

4.3. Table of TLVs

The following table provides a guide to which TLVs may be found in
which kinds of messages, and in what quantity. The messages are as
follows: Request is a TEAP Request, Response is a TEAP Response,
Success is a message containing a successful Result TLV, and Failure
is a message containing a failed Result TLV.

Request	Response	Success	Failure	TLVs
0-1	0-1	0	0	Authority-ID
0-1	0-1	0	0	Identity-Type
0-1	0-1	1	1	Result
0+	0+	0	0	NAK
0+	0+	0+	0+	Error
0-1	0-1	0	0	Channel-Binding
0+	0+	0+	0+	Vendor-Specific [NOTE1]
0	0-1	0-1	0-1	Request-Action
0-1	0-1	0	0	EAP-Payload
0-1	0-1	0-1	0-1	Intermediate-Result
0+	0+	0+	0	PAC-TLV
0-1	0-1	0-1	0-1	Crypto-Binding
0-1	0	0	0	Basic-Password-Auth-Req
0	0-1	0	0	Basic-Password-Auth-Resp
0-1	0	0-1	0	PKCS#7
0	0-1	0	0	PKCS#10
0-1	0-1	0-1	0	Server-Trusted-Root

[NOTE1] Vendor TLVs (included in Vendor-Specific TLVs) sent with a
Result TLV MUST be marked as optional.

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

0 This TLV MUST NOT be present in the message.

0+ Zero or more instances of this TLV MAY be present in the message.

0-1 Zero or one instance of this TLV MAY be present in the message.

1 Exactly one instance of this TLV MUST be present in the message.

5. Cryptographic Calculations

5.1. TEAP Authentication Phase 1: Key Derivations

   With TEAPv1, the TLS master secret is generated as specified in TLS.
   If a PAC is used then the master secret is obtained as described in
   [RFC5077].

   TEAPv1 makes use of the TLS Keying Material Exporters defined in
   [RFC5705] to derive the session_key_seed. The Label used in the
   derivation is "teap seesion key seed". The length of the session key
   seed material is 40 octets. No context data is used in the export
   process.

   The session_key_seed is used by the TEAP Authentication Phase 2

conversation to both cryptographically bind the inner method(s) to
the tunnel as well as generate the resulting TEAP session keys. The
other quantities are used as they are defined in [RFC5246].

5.2. Intermediate Compound Key Derivations

The session_key_seed derived as part of TEAP Phase 2 is used in TEAP
Phase 2 to generate an Intermediate Compound Key (IMCK) used to
verify the integrity of the TLS tunnel after each successful inner
authentication and in the generation of Master Session Key (MSK) and
Extended Master Session Key (EMSK) defined in [RFC3748]. Note that
the IMCK must be recalculated after each successful inner EAP method.

The first step in these calculations is the generation of the base
compound key, IMCK[n] from the session_key_seed and any session keys
derived from the successful execution of nth inner EAP methods. The
inner EAP method(s) may provide Inner Method Session Keys (IMSK),
IMSK1..IMSKn, corresponding to inner method 1 through n.

If an inner method supports export of an Extended Master Session Key
(EMSK), then the IMSK is derived from the EMSK as defined in
[RFC5295]. The usage label used is "TEAPbindkey@ietf.org" and the
length is 64 octets. Optional data parameter is not used in the
derivation.

    IMSK = First 32 octets of KDF(EMSK, "TEAPbindkey@ietf.org" | "\0"
    | 64)

    where the KDF is defined in [RFC5295].

If an inner method does not support export of an Extended Master
Session Key (EMSK), then IMSK is the MSK of the inner method. The
MSK is truncated at 32 octets if it is longer than 32 octets or
padded to a length of 32 octets with zeros if it is less than 32
octets.

If the ith inner method does not generate an EMSK or MSK, then IMSKi
is set to zero (e.g., MSKi = 32 octets of 0x00s). If an inner method
fails, then it is not included in this calculation. The derivations
of S-IMCK is as follows:

```
      S-IMCK[0] = session_key_seed
      For j = 1 to n-1 do
            IMCK[j] = TLS-PRF(S-IMCK[j-1], "Inner Methods Compound Keys",
                  IMSK[j], 60)
            S-IMCK[j] = first 40 octets of IMCK[j]
            CMK[j] = last 20 octets of IMCK[j]
```

where TLS-PRF is the PRF negotiated as part of TLS handshake
[RFC5246].

5.3. Computing the Compound MAC

For authentication methods that generate keying material, further
protection against man-in-the-middle attacks is provided through
cryptographically binding keying material established by both TEAP
Phase 1 and TEAP Phase 2 conversations. After each successful inner
EAP authentication, EAP MSKs are cryptographically combined with key
material from TEAP Phase 1 to generate a compound session key, CMK.
The CMK is used to calculate the Compound MAC as part of the Crypto-
Binding TLV described in Section 4.2.13, which helps provide
assurance that the same entities are involved in all communications
in TEAP. During the calculation of the Compound-MAC the MAC field is
filled with zeros.

The Compound MAC computation is as follows:

```
      CMK = CMK[j]
      Compound-MAC = HMAC-HASH( CMK, BUFFER )
```

where j is the number of the last successfully executed inner EAP
method, HASH is the default hash function or the alternative hash
function negotiated in TLS 1.2 [RFC5246], and BUFFER is created after
concatenating these fields in the following order:

1 The entire Crypto-Binding TLV attribute with the MAC field zeroed
   out.

2 The EAP Type sent by the other party in the first TEAP message.

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

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

5.4. EAP Master Session Key Generation

   TEAP Authentication assures the master session key (MSK) and Extended
   Master Session Key (EMSK) output from the EAP method are the result
   of all authentication conversations by generating an Intermediate
   Compound Key (IMCK). The IMCK is mutually derived by the peer and
   the server as described in Section 5.2 by combining the MSKs from
   inner EAP methods with key material from TEAP Phase 1. The resulting
   MSK and EMSK are generated as part of the IMCKn key hierarchy as
   follows:

      MSK = TLS-PRF(S-IMCK[j], "Session Key Generating Function", 64)
      EMSK = TLS-PRF(S-IMCK[j],
             "Extended Session Key Generating Function", 64)

   where j is the number of the last successfully executed inner EAP
   method.

   The EMSK is typically only known to the TEAP peer and server and is
   not provided to a third party. The derivation of additional keys and
   transportation of these keys to a third party is outside the scope of
   this document.

   If no EAP methods have been negotiated inside the tunnel or no EAP
   methods have been successfully completed inside the tunnel, the MSK
   and EMSK will be generated directly from the session_key_seed meaning
   S-IMCK = session_key_seed.

6. IANA Considerations

   This section provides guidance to the Internet Assigned Numbers
   Authority (IANA) regarding registration of values related to the TEAP
   protocol, in accordance with BCP 26, [RFC5226].

   The EAP Method Type number for TEAP needs to be assigned.

   The document defines a registry for TEAP TLV types, which may be
   assigned by Specification Required as defined in [RFC5226].
   Section 4.2 defines the TLV types that initially populate the
   registry. A summary of the TEAP TLV types is given below:

      0 Unassigned

1 Authority-ID TLV

2 Identity-Type TLV

3 Result TLV

4 NAK TLV

5 Error TLV

6 Channel-Binding TLV

7 Vendor-Specific TLV

8 Request-Action TLV

9 EAP-Payload TLV

10 Intermediate-Result TLV

11 PAC TLV

12 Crypto-Binding TLV

13 Basic-Password-Auth-Req TLV

14 Basic-Password-Auth-Resp TLV

15 PKCS#7 TLV

16 PKCS#10 TLV

17 Trusted-Server-Root TLV

The Identity-Type defined in Section 4.2.3 contains an Identity Type
code which is assigned on a Specification Required basis as defined
in [RFC5226]. The initial types defined are:

1 User

2 Machine

The Result TLV defined in Section 4.2.4, Request-Action TLV defined
in Section 4.2.9, and Intermediate-Result TLV defined in
Section 4.2.11 contain a Status code which is assigned on a
Specification Required basis as defined in [RFC5226]. The initial
types defined are:

1 Success

2 Failure

The Error-TLV defined in Section 4.2.6 requires an error-code. TEAP
Error-TLV error-codes are assigned based on Specification Required as
defined in [RFC5226].   The initial list of error codes is as follows:

2001 Tunnel_Compromise_Error

2002 Unexpected_TLVs_Exchanged

2003 Unsupported_Algorithm_In_CertificateSigning_Request

2004 Unsupported_Extension_In_CertificateSigning_Request

2005 Bad_Identity_In_CertificateSigning_Request

2006 Bad_CertificateSigning_Request

2007 Internal_CA_Error

2008 General_PKI_Error

The Request-Action TLV defined in Section 4.2.9 contains an action
code which is assigned on a Specification Required basis as defined
in [RFC5226].   The initial actions defined are:

1 Process-TLV

2 Negotiate-EAP

The PAC Attribute defined in Section 4.2.12.1 contains a Type code
which is assigned on a Specification Required basis as defined in
[RFC5226].   The initial types defined are:

1 PAC-key

2 PAC-Opaque

3 PAC-Lifetime

4 A-ID

      5 I-ID

      6 Reserved

      7 A-ID-Info

      8 PAC-Acknowledgement

      9 PAC-Info

      10 PAC-Type

      The PAC-Type defined in Section 4.2.12.6 contains a Type code which
      is assigned on a Specification Required basis as defined in
      [RFC5226]. The initial types defined are:

      1 Tunnel PAC

      The Trusted-Server-Root TLV defined in Section 4.2.18 contains a
      Credential-Format code which is assigned on a Specification Required
      basis as defined in [RFC5226]. The initial types defined are:

      1 PKCS#7-Server-Certificate-Root

      The various values under Vendor-Specific TLV are assigned by Private
      Use and do not need to be assigned by IANA.

      TEAP makes use of the TLS Keying Material Exporters defined in
      [RFC5705]. The Label used in the derivation as defined in
      Section 5.1 is "teap seesion key seed".

      TEAP registers a TEAP binding usage label from the "USRK Key Labels"
      name space defined in [RFC5295] with a value "TEAPbindkey@ietf.org".

7. Security Considerations

      TEAP is designed with a focus on wireless media, where the medium
      itself is inherent to eavesdropping. Whereas in wired media, an
      attacker would have to gain physical access to the wired medium;
      wireless media enables anyone to capture information as it is
      transmitted over the air, enabling passive attacks. Thus, physical
      security can not be assumed and security vulnerabilities are far
      greater. The threat model used for the security evaluation of TEAP
      is defined in the EAP [RFC3748].

7.1. Mutual Authentication and Integrity Protection

   TEAP as a whole, provides message and integrity protection by
   establishing a secure tunnel for protecting the authentication
   method(s). The confidentiality and integrity protection is defined
   by TLS and provides the same security strengths afforded by TLS
   employing a strong entropy shared master secret. The integrity of
   the key generating authentication methods executed within the TEAP
   tunnel is verified through the calculation of the Crypto-Binding TLV.
   This ensures that the tunnel endpoints are the same as the inner
   method endpoints.

   The Result TLV is protected and conveys the true Success or Failure
   of TEAP, and should be used as the indicator of its success or
   failure respectively. However, as EAP must terminate with a clear
   text EAP Success or Failure, a peer will also receive a clear text
   EAP Success or Failure. The received clear text EAP Success or
   Failure must match that received in the Result TLV; the peer SHOULD
   silently discard those clear text EAP success or failure messages
   that do not coincide with the status sent in the protected Result
   TLV.

7.2. Method Negotiation

   As is true for any negotiated EAP protocol, NAK packets used to
   suggest an alternate authentication method are sent unprotected and
   as such, are subject to spoofing. During unprotected EAP method
   negotiation, NAK packets may be interjected as active attacks to
   negotiate down to a weaker form of authentication, such as EAP-MD5
   (which only provides one-way authentication and does not derive a
   key). Both the peer and server should have a method selection policy
   that prevents them from negotiating down to weaker methods. Inner
   method negotiation resists attacks because it is protected by the
   mutually authenticated TLS tunnel established. Selection of TEAP as
   an authentication method does not limit the potential inner
   authentication methods, so TEAP should be selected when available.

   An attacker cannot readily determine the inner EAP method used,
   except perhaps by traffic analysis. It is also important that peer
   implementations limit the use of credentials with an unauthenticated
   or unauthorized server.

7.3. Separation of Phase 1 and Phase 2 Servers

   Separation of the TEAP Phase 1 from the Phase 2 conversation is NOT
   RECOMMENDED. Allowing the Phase 1 conversation to be terminated at a
   different server than the Phase 2 conversation can introduce
   vulnerabilities if there is not a proper trust relationship and

protection for the protocol between the two servers. Some
vulnerabilities include:

o Loss of identity protection

o Offline dictionary attacks

o Lack of policy enforcement

o Man-in-the-middle attacks (as described in
   [I-D.hartman-emu-mutual-crypto-bind])

There may be cases where a trust relationship exists between the
Phase 1 and Phase 2 servers, such as on a campus or between two
offices within the same company, where there is no danger in
revealing the inner identity and credentials of the peer to entities
between the two servers. In these cases, using a proxy solution
without end-to-end protection of TEAP MAY be used. The TEAP
encrypting/decrypting gateway SHOULD, at a minimum, provide support
for IPsec or similar protection in order to provide confidentiality
for the portion of the conversation between the gateway and the EAP
server. In addition, separation of the inner and outer method
servers allows for crypto-binding based on the inner method MSK to be
thwarted as described in [I-D.hartman-emu-mutual-crypto-bind].
Implentor and deployment SHOULD adopt various mitigation strategies
described in [I-D.hartman-emu-mutual-crypto-bind]. If the inner
method is deriving EMSK, then this threat is mitigated as TEAP
utilizes the mutual crypto-binding based on EMSK as described in
[I-D.hartman-emu-mutual-crypto-bind].

7.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies

TEAP addresses the known deficiencies and weaknesses in the EAP
method. By employing a shared secret between the peer and server to
establish a secured tunnel, TEAP enables:

o Per packet confidentiality and integrity protection

o User identity protection

o Better support for notification messages

o Protected EAP inner method negotiation

o Sequencing of EAP methods

o Strong mutually derived master session keys

   o Acknowledged success/failure indication

   o Faster re-authentications through session resumption

   o Mitigation of dictionary attacks

   o Mitigation of man-in-the-middle attacks

   o Mitigation of some denial-of-service attacks

   It should be noted that TEAP, as in many other authentication
   protocols, a denial-of-service attack can be mounted by adversaries
   sending erroneous traffic to disrupt the protocol. This is a problem
   in many authentication or key agreement protocols and is therefore
   noted for TEAP as well.

   TEAP was designed with a focus on protected authentication methods
   that typically rely on weak credentials, such as password-based
   secrets. To that extent, the TEAP Authentication mitigates several
   vulnerabilities, such as dictionary attacks, by protecting the weak
   credential-based authentication method. The protection is based on
   strong cryptographic algorithms in TLS to provide message
   confidentiality and integrity. The keys derived for the protection
   relies on strong random challenges provided by both peer and server
   as well as an established key with strong entropy. Implementations
   should follow the recommendation in [RFC4086] when generating random
   numbers.

7.4.1. User Identity Protection and Verification

   The initial identity request response exchange is sent in cleartext
   outside the protection of TEAP. Typically the Network Access
   Identifier (NAI) [RFC4282] in the identity response is useful only
   for the realm information that is used to route the authentication
   requests to the right EAP server. This means that the identity
   response may contain an anonymous identity and just contain realm
   information. In other cases, the identity exchange may be eliminated
   altogether if there are other means for establishing the destination
   realm of the request. In no case should an intermediary place any
   trust in the identity information in the identity response since it
   is unauthenticated and may not have any relevance to the
   authenticated identity. TEAP implementations should not attempt to
   compare any identity disclosed in the initial cleartext EAP Identity
   response packet with those Identities authenticated in Phase 2.

   Identity request-response exchanges sent after the TEAP tunnel is
   established are protected from modification and eavesdropping by
   attackers.

Note that since TLS client certificates are sent in the clear, if identity protection is required, then it is possible for the TLS authentication to be re-negotiated after the first server authentication. To accomplish this, the server will typically not request a certificate in the server_hello, then after the server_finished message is sent, and before TEAP Phase 2, the server MAY send a TLS hello_request. This allows the client to perform client authentication by sending a client_hello if it wants to, or send a no_renegotiation alert to the server indicating that it wants to continue with TEAP Phase 2 instead. Assuming that the client permits renegotiation by sending a client_hello, then the server will respond with server_hello, a certificate and certificate_request messages. The client replies with certificate, client_key_exchange and certificate_verify messages. Since this re-negotiation occurs within the encrypted TLS channel, it does not reveal client certificate details. It is possible to perform certificate authentication using an EAP method (for example: EAP-TLS) within the TLS session in TEAP Phase 2 instead of using TLS handshake renegotiation.

7.4.2. Dictionary Attack Resistance

TEAP was designed with a focus on protected authentication methods that typically rely on weak credentials, such as password-based secrets. TEAP mitigates dictionary attacks by allowing the establishment of a mutually authenticated encrypted TLS tunnel providing confidentiality and integrity to protect the weak credential based authentication method.

7.4.3. Protection against Man-in-the-Middle Attacks

Allowing methods to be executed both with and without the protection of a secure tunnel opens up a possibility of a man-in-the-middle attack. To avoid man-in-the-middle attacks it is recommended to always deploy authentication methods with protection of TEAP. TEAP provides protection from man-in-the-middle attacks even if a deployment chooses to execute inner EAP methods both with and without TEAP protection, TEAP prevents this attack in two ways:

1. By using the PAC-Key to mutually authenticate the peer and server during TEAP Authentication Phase 1 establishment of a secure tunnel.

2. By using the keys generated by the inner authentication method (if the inner methods are key generating) in the crypto-binding exchange and in the generation of the key material exported by the EAP method described in Section 5.

7.4.4. PAC Binding to User Identity

   A PAC may be bound to a user identity. A compliant implementation of
   TEAP MUST validate that an identity obtained in the PAC-Opaque field
   matches at minimum one of the identities provided in the TEAP Phase 2
   authentication method. This validation provides another binding to
   ensure that the intended peer (based on identity) has successfully
   completed the TEAP Phase 1 and proved identity in the Phase 2
   conversations.

7.5. Protecting against Forged Clear Text EAP Packets

   EAP Success and EAP Failure packets are, in general, sent in clear
   text and may be forged by an attacker without detection. Forged EAP
   Failure packets can be used to attempt to convince an EAP peer to
   disconnect. Forged EAP Success packets may be used to attempt to
   convince a peer that authentication has succeeded, even though the
   authenticator has not authenticated itself to the peer.

   By providing message confidentiality and integrity, TEAP provides
   protection against these attacks. Once the peer and AS initiate the
   TEAP Authentication Phase 2, compliant TEAP implementations must
   silently discard all clear text EAP messages, unless both the TEAP
   peer and server have indicated success or failure using a protected
   mechanism. Protected mechanisms include TLS alert mechanism and the
   protected termination mechanism described in Section 3.3.3.

   The success/failure decisions within the TEAP tunnel indicate the
   final decision of the TEAP authentication conversation. After a
   success/failure result has been indicated by a protected mechanism,
   the TEAP peer can process unprotected EAP Success and EAP Failure
   messages; however the peer MUST ignore any unprotected EAP success or
   failure messages where the result does not match the result of the
   protected mechanism.

   To abide by [RFC3748], the server must send a clear text EAP Success
   or EAP Failure packet to terminate the EAP conversation. However,
   since EAP Success and EAP Failure packets are not retransmitted, the
   final packet may be lost. While a TEAP protected EAP Success or EAP
   Failure packet should not be a final packet in a TEAP conversation,
   it may occur based on the conditions stated above, so an EAP peer
   should not rely upon the unprotected EAP success and failure
   messages.

7.6. Server Certificate Validation

   As part of the TLS negotiation, the server presents a certificate to
   the peer. 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. When performing server certificate validation implementations MUST provide support rules in [RFC5280] for validating certificates against a known trust anchor. In addition, implementations SHOULD support matching the realm portion of the client's NAI against a SubjectAltName of type dNSName within the server certificate. Please note that in the case where the EAP authentication is remote, 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 domain and whether the authenticator can be authorized by a server in that domain.

7.7. Tunnel PAC Considerations

Since the Tunnel PAC is stored by the peer, special care should be given to the overall security of the peer. The Tunnel PAC must be securely stored by the peer to prevent theft or forgery of any of the Tunnel PAC components. In particular, the peer must securely store the PAC-Key and protect it from disclosure or modification. Disclosure of the PAC-Key enables an attacker to establish the TEAP tunnel; however, disclosure of the PAC-Key does not reveal the peer or server identity or compromise any other peer's PAC credentials. Modification of the PAC-Key or PAC-Opaque components of the Tunnel PAC may also lead to denial of service as the tunnel establishment will fail. The PAC-Opaque component is the effective TLS ticket extension used to establish the tunnel using the techniques of [RFC5077]. Thus, the security considerations defined by [RFC5077] also apply to the PAC- Opaque. The PAC-Info may contain information about the Tunnel PAC such as the identity of the PAC issuer and the Tunnel PAC lifetime for use in the management of the Tunnel PAC. The PAC-Info should be securely stored by the peer to protect it from disclosure and modification.

7.8. Security Claims

This section provides the needed security claim requirement for EAP [RFC3748].

Auth. mechanism: Certificate based, shared secret based and
                          various tunneled authentication mechanisms.

```
      Ciphersuite negotiation: Yes

      Mutual authentication:   Yes

      Integrity protection:    Yes, Any method executed within the TEAP
                               tunnel is integrity protected.  The
                               cleartext EAP headers outside the tunnel are
                               not integrity protected.

      Replay protection:       Yes

      Confidentiality:         Yes

      Key derivation:          Yes

      Key strength:            See Note 1 below.

      Dictionary attack prot.: Yes

      Fast reconnect:          Yes

      Cryptographic binding:   Yes

      Session independence:    Yes

      Fragmentation:           Yes

      Key Hierarchy:           Yes

      Channel binding:         Yes
```

      Notes

   1. BCP 86 [RFC3766] offers advice on appropriate key sizes. The
       National Institute for Standards and Technology (NIST) also
       offers advice on appropriate key sizes in [NIST-SP-800-57].
       [RFC3766] Section 5 advises use of the following required RSA or
       DH module and DSA subgroup size in bits, for a given level of
       attack resistance in bits. Based on the table below, a 2048-bit
       RSA key is required to provide 128-bit equivalent key strength:

Attack Resistance (bits)	RSA or DH Modulus size (bits)	DSA subgroup size (bits)
70	947	129
80	1228	148
90	1553	167
100	1926	186
150	4575	284
200	8719	383
250	14596	482

8. Acknowledgements

   The TEAP v1 design and protocol specification is based on EAP-FAST
   [RFC4851], which included the ideas and hard efforts of Nancy Cam-
   Winget, David McGrew, Joe Salowey, Hao Zhou, Pad Jakkahalli, Mark
   Krischer, Doug Smith, and Glen Zorn of Cisco Systems, Inc.

   The TLV processing was inspired from work on the Protected Extensible
   Authentication Protocol version 2 (PEAPv2) with Ashwin Palekar, Dan
   Smith, Sean Turner and Simon Josefsson.

   Helpful review comments were provided by Russ Housley, Jari Arkko,
   Ilan Frenkel, Jeremy Steiglitz, Dan Harkins, Sam Hartman, and Jim
   Schaad.

9. References

9.1. Normative References

   [I-D.ietf-emu-chbind]   Hartman, S., Clancy, T., and K.
                                    Hoeper, "Channel Binding
                                    Support for EAP Methods",
                                    draft-ietf-emu-chbind-15 (work
                                    in progress), May 2012.

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

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

   [RFC4851]   Cam-Winget, N., McGrew, D.,

                                        Salowey, J., and H. Zhou, "The
                                        Flexible Authentication via
                                        Secure Tunneling Extensible
                                        Authentication Protocol Method
                                        (EAP-FAST)", RFC 4851,
                                        May 2007.

          [RFC5077]   Salowey, J., Zhou, H., Eronen,
                                        P., and H. Tschofenig,
                                        "Transport Layer Security (TLS)
                                        Session Resumption without
                                        Server-Side State", RFC 5077,
                                        January 2008.

          [RFC5226]   Narten, T. and H. Alvestrand,
                                        "Guidelines for Writing an IANA
                                        Considerations Section in
                                        RFCs", BCP 26, RFC 5226,
                                        May 2008.

          [RFC5246]   Dierks, T. and E. Rescorla,
                                        "The Transport Layer Security
                                        (TLS) Protocol Version 1.2",
                                        RFC 5246, August 2008.

          [RFC5295]   Salowey, J., Dondeti, L.,
                                        Narayanan, V., and M. Nakhjiri,
                                        "Specification for the
                                        Derivation of Root Keys from an
                                        Extended Master Session Key
                                        (EMSK)", RFC 5295, August 2008.

          [RFC5705]   Rescorla, E., "Keying Material
                                        Exporters for Transport Layer
                                        Security (TLS)", RFC 5705,
                                        March 2010.

          [RFC5746]   Rescorla, E., Ray, M.,
                                        Dispensa, S., and N. Oskov,
                                        "Transport Layer Security (TLS)
                                        Renegotiation Indication
                                        Extension", RFC 5746,
                                        February 2010.

          [RFC5929]   Altman, J., Williams, N., and
                                        L. Zhu, "Channel Bindings for
                                        TLS", RFC 5929, July 2010.

9.2. Informative References

   [I-D.hartman-emu-mutual-crypto-bind] Hartman, S., Wasserman, M., and
                                         D. Zhang, "EAP Mutual
                                         Cryptographic Binding", draft-
                                         hartman-emu-mutual-crypto-bind-
                                         00 (work in progress),
                                         March 2012.

   [I-D.ietf-emu-eaptunnel-req]   Zhou, H., Salowey, J., Hoeper,
                                         K., and S. Hanna, "Requirements
                                         for a Tunnel Based EAP Method",
                                         draft-ietf-emu-eaptunnel-req-09
                                         (work in progress),
                                         December 2010.

   [IEEE.802-1X.2004]   "Local and Metropolitan Area
                                         Networks: Port-Based Network
                                         Access Control", IEEE Standard
                                         802.1X, December 2004.

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

   [PEAP]   Microsoft Corporation, ""[MS-
                                         PEAP]: Protected Extensible
                                         Authentication Protocol (PEAP)
                                         Specification"", August 2009.

   [RFC2311]   Dusse, S., Hoffman, P.,
                                         Ramsdell, B., Lundblade, L.,
                                         and L. Repka, "S/MIME Version 2
                                         Message Specification",
                                         RFC 2311, March 1998.

   [RFC2315]   Kaliski, B., "PKCS #7:
                                         Cryptographic Message Syntax
                                         Version 1.5", RFC 2315,
                                         March 1998.

   [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.

    [RFC2986]   Nystrom, M. and B. Kaliski,
                                  "PKCS #10: Certification
                                  Request Syntax Specification
                                  Version 1.7", RFC 2986,
                                  November 2000.

    [RFC3579]   Aboba, B. and P. Calhoun,
                                  "RADIUS (Remote Authentication
                                  Dial In User Service) Support
                                  For Extensible Authentication
                                  Protocol (EAP)", RFC 3579,
                                  September 2003.

    [RFC3629]   Yergeau, F., "UTF-8, a
                                  transformation format of ISO
                                  10646", STD 63, RFC 3629,
                                  November 2003.

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

    [RFC4072]   Eronen, P., Hiller, T., and G.
                                  Zorn, "Diameter Extensible
                                  Authentication Protocol (EAP)
                                  Application", RFC 4072,
                                  August 2005.

    [RFC4086]   Eastlake, D., Schiller, J., and
                                  S. Crocker, "Randomness
                                  Requirements for Security",
                                  BCP 106, RFC 4086, June 2005.

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

    [RFC4945]   Korver, B., "The Internet IP
                                  Security PKI Profile of IKEv1/
                                  ISAKMP, IKEv2, and PKIX",
                                  RFC 4945, August 2007.

    [RFC5247]   Aboba, B., Simon, D., and P.

                                    Eronen, "Extensible
                                    Authentication Protocol (EAP)
                                    Key Management Framework",
                                    RFC 5247, August 2008.

   [RFC5272]   Schaad, J. and M. Myers,
                                    "Certificate Management over
                                    CMS (CMC)", RFC 5272,
                                    June 2008.

   [RFC5280]   Cooper, D., Santesson, S.,
                                    Farrell, S., Boeyen, S.,
                                    Housley, R., and W. Polk,
                                    "Internet X.509 Public Key
                                    Infrastructure Certificate and
                                    Certificate Revocation List
                                    (CRL) Profile", RFC 5280,
                                    May 2008.

   [RFC5281]   Funk, P. and S. Blake-Wilson,
                                    "Extensible Authentication
                                    Protocol Tunneled Transport
                                    Layer Security Authenticated
                                    Protocol Version 0 (EAP-
                                    TTLSv0)", RFC 5281,
                                    August 2008.

   [RFC5421]   Cam-Winget, N. and H. Zhou,
                                    "Basic Password Exchange within
                                    the Flexible Authentication via
                                    Secure Tunneling Extensible
                                    Authentication Protocol (EAP-
                                    FAST)", RFC 5421, March 2009.

   [RFC5931]   Harkins, D. and G. Zorn,
                                    "Extensible Authentication
                                    Protocol (EAP) Authentication
                                    Using Only a Password",
                                    RFC 5931, August 2010.

   [RFC6066]   Eastlake, D., "Transport Layer
                                    Security (TLS) Extensions:
                                    Extension Definitions",
                                    RFC 6066, January 2011.

   [RFC6124]   Sheffer, Y., Zorn, G.,
                                    Tschofenig, H., and S. Fluhrer,
                                    "An EAP Authentication Method

                                   Based on the Encrypted Key
                                   Exchange (EKE) Protocol",
                                   RFC 6124, February 2011.

Appendix A. Evaluation Against Tunnel Based EAP Method Requirements

   This section evaluates all tunnel based EAP method requirements
   described in [I-D.ietf-emu-eaptunnel-req] against TEAP version 1.

A.1. Requirement 4.1.1 RFC Compliance

   TEAP v1 meets this requirement by being compliant to RFC 3748, RFC
   4017, RFC 5247, and RFC 4962. It is also compliant with the
   "cryptographic algorithm agility" requirement by leveraging TLS 1.2
   for all cryptographic algorithm negotiation.

A.2. Requirement 4.2.1 TLS Requirements

   Requirement 4.2.1 states:

   The tunnel based method MUST support TLS version 1.2 [RFC5246] and
   may support earlier versions greater than SSL 2.0 to enable the
   possibility of backwards compatibility.

   TEAP v1 meets this requirement by mandating TLS version 1.2 support
   as defined in Section 3.2.

A.3. Requirement 4.2.1.1.1 Cipher Suite Negotiation

   Requirement 4.2.1.1.1 states:

   Hence, the tunnel method MUST provide integrity protected cipher
   suite negotiation with secure integrity algorithms and integrity
   keys.

   TEAP v1 meets this requirement by using TLS to provide protected
   cipher suite negotiation.

A.4. Requirement 4.2.1.1.2 Tunnel Data Protection Algorithms

   Requirement 4.2.1.1.2 states:

   The tunnel method MUST provide at least one mandatory to implement
   cipher suite that provides the equivalent security of 128-bit AES for
   encryption and message authentication.

   TEAP v1 meets this requirement by mandating
   TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory to implement cipher suite

   as defined in Section 3.2.

A.5.   Requirement 4.2.1.1.3 Tunnel Authentication and Key Establishment

   TEAP v1 meets this requirement by mandating
   TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory to implement cipher suite
   which provides certificate-based authentication of the server and is
   approved by NIST. The mandatory to implement cipher suites only
   include cipher suites that use strong cryptographic algorithms. They
   do not include cipher suites providing mutually anonymous
   authentication or static Diffie-Hellman cipher suites as defined in
   Section 3.2.

A.6.   Requirement 4.2.1.2 Tunnel Replay Protection

   TEAP v1 meets this requirement by using TLS to provide sufficient
   replay protection.

A.7.   Requirement 4.2.1.3 TLS Extensions

   TEAP v1 meets this requirement by allowing TLS extensions, such as
   TLS Certificate Status Request extension [RFC6066] and SessionTicket
   extension [RFC5077] to be used during TLS tunnel establishment.

A.8.   Requirement 4.2.1.4 Peer Identity Privacy

   TEAP v1 meets this requirement by establishment of the TLS tunnel and
   protection of inner method specific identities. In addition, the
   peer certificate can be sent confidentially (i.e. encrypted).

A.9.   Requirement 4.2.1.5 Session Resumption

   TEAP v1 meets this requirement by mandating support of TLS session
   resumption as defined in Section 3.2.1 and TLS Session Resume Using a
   PAC as defined in Section 3.2.2 .

A.10. Requirement 4.2.2 Fragmentation

   TEAP v1 meets this requirement by leveraging fragmentation support
   provided by TLS as defined in Section 3.7.

A.11. Requirement 4.2.3 Protection of Data External to Tunnel

   TEAP v1 meets this requirement by including TEAP version number
   received in the computation of crypto-binding TLV as defined in
   Section 4.2.13.

A.12. Requirement 4.3.1 Extensible Attribute Types

   TEAP v1 meets this requirement by using an extensible TLV data layer
   inside the tunnel as defined in Section 4.2.

A.13. Requirement 4.3.2 Request/Challenge Response Operation

   TEAP v1 meets this requirement by allowing multiple TLVs to be sent
   in a single EAP request or response packet, while maintaining the
   half-duplex operation typical of EAP.

A.14. Requirement 4.3.3 Indicating Criticality of Attributes

   TEAP v1 meets this requirement by having a mandatory bit in TLV to
   indicate whether it is mandatory to support or not as defined in
   Section 4.2.

A.15. Requirement 4.3.4 Vendor Specific Support

   TEAP v1 meets this requirement by having a Vendor-Specific TLV to
   allow vendors to define their own attributes as defined in
   Section 4.2.8.

A.16. Requirement 4.3.5 Result Indication

   TEAP v1 meets this requirement by having a Result TLV to exchange the
   final result of the EAP authentication so both the peer and server
   have a synchronized state as defined in Section 4.2.4.

A.17. Requirement 4.3.6 Internationalization of Display Strings

   TEAP v1 meets this requirement by supporting UTF-8 format in Basic-
   Password-Auth-Req TLV as defined in Section 4.2.14 and Basic-
   Password-Auth-Resp TLV as defined in Section 4.2.15.

A.18. Requirement 4.4 EAP Channel Binding Requirements

   TEAP v1 meets this requirement by having a Channel-Binding TLV to
   exchange the EAP channel binding data as defined in Section 4.2.7.

A.19. Requirement 4.5.1.1 Confidentiality and Integrity

   TEAP v1 meets this requirement by running the password authentication
   inside a protected TLS tunnel.

A.20.   Requirement 4.5.1.2 Authentication of Server

   TEAP v1 meets this requirement by mandating authentication of the
   server before establishment of the protected TLS and then running
   inner password authentication as defined in Section 3.2.

A.21.   Requirement 4.5.1.3 Server Certificate Revocation Checking

   TEAP v1 meets this requirement by supporting TLS Certificate Status
   Request extension [RFC6066] during tunnel establishment.

A.22.   Requirement 4.5.2 Internationalization

   TEAP v1 meets this requirement by supporting UTF-8 format in Basic-
   Password-Auth-Req TLV as defined in Section 4.2.14 and Basic-
   Password-Auth-Resp TLV as defined in Section 4.2.15.

A.23.   Requirement 4.5.3 Meta-data

   TEAP v1 meets this requirement by supporting Identity-Type TLV as
   defined in Section 4.2.3 to indicate whether the authentication is
   for a user or a machine.

A.24.   Requirement 4.5.4 Password Change

   TEAP v1 meets this requirement by supporting multiple Basic-Password-
   Auth-Req TLV and Basic-Password-Auth-Resp TLV exchanges within a
   single EAP authentication, which allows "housekeeping"" functions
   such as password change.

A.25.   Requirement 4.6.1 Method Negotiation

   TEAP v1 meets this requirement by supporting inner EAP method
   negotiation within the protected TLS tunnel.

A.26.   Requirement 4.6.2 Chained Methods

   TEAP v1 meets this requirement by supporting inner EAP method
   chaining within protected TLS tunnel as defined in Section 3.3.1.

A.27.   Requirement 4.6.3 Cryptographic Binding with the TLS Tunnel

   TEAP v1 meets this requirement by supporting cryptographic binding of
   the inner EAP method keys with the keys derived from the TLS tunnel
   as defined in Section 4.2.13.

A.28. Requirement 4.6.4 Peer Initiated

   TEAP v1 meets this requirement by supporting Request-Action TLV as
   defined in Section 4.2.9 to allow peer to initiate another inner EAP
   method.

A.29. Requirement 4.6.5 Method Meta-data

   TEAP v1 meets this requirement by supporting Identity-Type TLV as
   defined in Section 4.2.3 to indicate whether the authentication is
   for a user or a machine.

Appendix B. Major Differences from EAP-FAST

   This document is a new standard tunnel EAP method based on revision
   of the EAP-FAST version 1 [RFC4851] which contains improved
   flexibility, particularly for negotiation of cryptographic
   algorithms. The major changes are:

   1. The EAP method name have been changed from EAP-FAST to TEAP,
       hence it would require a new EAP method type to be assigned.

   2. This version of TEAP MUST support TLS 1.2 [RFC5246].

   3. The key derivation now makes use of TLS keying material exporters
       [RFC5705] and the PRF and hash function negotiated in TLS. This
       is to simplify implementation and better support cryptographic
       algorithm agility.

   4. TEAP is in full conformance with TLS Ticket extension [RFC5077]
       as described in Section 3.2.2.

   5. Support of passing optional outer TLVs in the first two message
       exchanges, in addition to the Authority-ID TLV data in EAP-FAST.

   6. Basic password authentication on the TLV level has been added in
       addition to the existing inner EAP method.

   7. Additional TLV types have been defined to support EAP channel
       binding and meta-data. They are Identity-Type TLV and Channel-
       Binding TLVs, defined in Section 4.2.

Appendix C. Examples

C.1. Successful Authentication

   The following exchanges show a successful TEAP authentication with
   basic password authentication and optional PAC refreshment, the

conversation will appear as follows:

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

                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TEAP Start, S bit set, Authority-ID)

     EAP-Response/
     EAP-Type=TEAP, V=1
     (TLS client_hello with
      PAC-Opaque in SessionTicket extension)->

                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TLS server_hello,
                               (TLS change_cipher_spec,
                                TLS finished)



     EAP-Response/
     EAP-Type=TEAP, V=1 ->
     (TLS change_cipher_spec,
      TLS finished)

     TLS channel established
     (messages sent within the TLS channel)

                               <- Basic-Password-Auth-Req TLV, Challenge

     Basic-Password-Auth-Resp TLV, Response with both
     user name and password) ->

     optional additional exchanges (new pin mode,
     password change etc.) ...

                               <- Crypto-Binding TLV (Request),
                                      Result TLV (Success),
                                   (Optional PAC TLV)
```

```
        Crypto-Binding TLV(Response),
        Result TLV (Success),
        (PAC TLV Acknowledgment) ->

        TLS channel torn down
        (messages sent in clear text)

                              <- EAP-Success
```

C.2. Failed Authentication

   The following exchanges show a failed TEAP authentication due to
   wrong user credentials, the conversation will appear as follows:

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

      EAP-Response/
      Identity (MyID1) ->


                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TEAP Start, S bit set, Authority-ID)

      EAP-Response/
      EAP-Type=TEAP, V=1
      (TLS client_hello with
       PAC-Opaque in SessionTicket extension)->

                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TLS server_hello,
                               (TLS change_cipher_spec,
                                TLS finished)

      EAP-Response/
      EAP-Type=TEAP, V=1 ->
      (TLS change_cipher_spec,
       TLS finished)

      TLS channel established
      (messages sent within the TLS channel)

                            <- Basic-Password-Auth-Req TLV, Challenge

      Basic-Password-Auth-Resp TLV, Response with both
      user name and password) ->

                               <- Result TLV (Failure)

      Result TLV (Failure) ->

      TLS channel torn down
      (messages sent in clear text)

                               <- EAP-Failure
```

C.3. Full TLS Handshake using Certificate-based Cipher Suite

   In the case where an abbreviated TLS handshake is tried and failed
   and falls back to certificate based full TLS handshake occurs within
   TEAP Phase 1, the conversation will appear as follows:

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

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

                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TEAP Start, S bit set, Authority-ID)
     EAP-Response/
     EAP-Type=TEAP, V=1
     (TLS client_hello
     [PAC-Opaque extension])->

     // Peer sends PAC-Opaque of Tunnel PAC along with a list of
        ciphersuites supported. If the server rejects the PAC-
        Opaque, if falls through to the full TLS handshake

                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS server_hello,
                               TLS certificate,
                              [TLS server_key_exchange,]
                              [TLS certificate_request,]
                               TLS server_hello_done)
     EAP-Response/
     EAP-Type=TEAP, V=1
     ([TLS certificate,]
      TLS client_key_exchange,
     [TLS certificate_verify,]
      TLS change_cipher_spec,
      TLS finished) ->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS change_cipher_spec,
                               TLS finished,
                               EAP-Payload-TLV[EAP-Request/
                               Identity])
```

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

        // First EAP Payload TLV is piggybacked to the TLS Finished as
           Application Data and protected by the TLS tunnel

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

        // identity protected by TLS.

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

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

        // Method X exchanges followed by Protected Termination

                              <- Intermediate-Result-TLV (Success),
                                 Crypto-Binding TLV (Request),
                                 Result TLV (Success)

        Intermediate-Result-TLV (Success),
        Crypto-Binding TLV (Response),
        Result-TLV (Success) ->

        // TLS channel torn down
        (messages sent in clear text)

                                 <- EAP-Success
```

C.4. Client authentication during Phase 1 with identity privacy

   In the case where a certificate based TLS handshake occurs within
   TEAP Phase 1, and client certificate authentication and identity
   privacy is desired, therefore TLS renegotiation is being used to
   transmit the peer credentials in the protected TLS tunnel, the
   conversation will appear as follows:

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

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

```
     full user identity.

                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TEAP Start, S bit set, Authority-ID)
     EAP-Response/
     EAP-Type=TEAP, V=1
     (TLS client_hello)->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS server_hello,
                               TLS certificate,
                              [TLS server_key_exchange,]
                              [TLS certificate_request,]
                               TLS server_hello_done)
     EAP-Response/
     EAP-Type=TEAP, V=1
     (TLS client_key_exchange,
      TLS change_cipher_spec,
      TLS finished) ->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS change_cipher_spec,
                               TLS finished,
                               EAP-Payload-TLV[EAP-Request/
                               Identity])

     // TLS channel established
        (EAP Payload messages sent within the TLS channel)

     // peer sends TLS client_hello to request TLS renegotiation

     TLS client_hello ->


                              <- TLS server_hello,
                               TLS certificate,
                               [TLS server_key_exchange,]
                               [TLS certificate_request,]
                               TLS server_hello_done
     [TLS certificate,]
      TLS client_key_exchange,
     [TLS certificate_verify,]
      TLS change_cipher_spec,
      TLS finished ->

                              <- TLS change_cipher_spec,
                                 TLS finished,
```

```
                                   Crypto-Binding TLV (Request),
                                   Result TLV (Success)

        Crypto-Binding TLV (Response),
        Result-TLV (Success)) ->

        //TLS channel torn down
        (messages sent in clear text)

                                   <- EAP-Success
```

C.5. Fragmentation and Reassembly

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

```
        Authenticating Peer      Authenticator
        -------------------      -------------
                                 <- EAP-Request/
                                 Identity
        EAP-Response/
        Identity (MyID) ->
                                 <- EAP-Request/
                                 EAP-Type=TEAP, V=1
                                 (TEAP Start, S bit set, Authority-ID)

        EAP-Response/
        EAP-Type=TEAP, V=1
        (TLS client_hello)->
                                 <- EAP-Request/
                                 EAP-Type=TEAP, V=1
                                 (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=TEAP, V=1 ->

                                 <- EAP-Request/
                                    EAP-Type=TEAP, V=1
                                 (Fragment 2: M bit set)
        EAP-Response/
        EAP-Type=TEAP, V=1 ->
                                 <- EAP-Request/
```

```
                                EAP-Type=TEAP, V=1
                                (Fragment 3)
        EAP-Response/
        EAP-Type=TEAP, V=1
        ([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=TEAP, V=1
        EAP-Response/
        EAP-Type=TEAP, V=1
        (Fragment 2)->
                                 <- EAP-Request/
                                 EAP-Type=TEAP, V=1
                                 (TLS change_cipher_spec,
                                  TLS finished,
                                 [EAP-Payload-TLV[
                                 EAP-Request/Identity]])

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

        // First EAP Payload TLV is piggybacked to the TLS Finished as
           Application Data and protected by the TLS tunnel

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

        // identity protected by TLS.

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

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

        // Method X exchanges followed by Protected Termination

                            <- Intermediate-Result-TLV (Success),
                               Crypto-Binding TLV (Request),
                               Result TLV (Success)

        Intermediate-Result-TLV (Success),
        Crypto-Binding TLV (Response),
```

```
        Result-TLV (Success) ->

        // TLS channel torn down
        (messages sent in clear text)

                                  <- EAP-Success
```

C.6. Sequence of EAP Methods

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

```
      Authenticating Peer      Authenticator
      -------------------      -------------
                               <- EAP-Request/
                               Identity
      EAP-Response/
      Identity (MyID1) ->
                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TEAP Start, S bit set, Authority-ID)

      EAP-Response/
      EAP-Type=TEAP, V=1
      (TLS client_hello)->
                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TLS server_hello,
                                TLS certificate,
                               [TLS server_key_exchange,]
                               [TLS certificate_request,]
                                TLS server_hello_done)
      EAP-Response/
      EAP-Type=TEAP, V=1
      ([TLS certificate,]
       TLS client_key_exchange,
      [TLS certificate_verify,]
       TLS change_cipher_spec,
       TLS finished) ->
                                <- EAP-Request/
                                EAP-Type=TEAP, V=1
                                (TLS change_cipher_spec,
                                 TLS finished,
                                 Identity-Type TLV,
                                EAP-Payload-TLV[
                                EAP-Request/Identity])
```

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

        // First EAP Payload TLV is piggybacked to the TLS Finished as
           Application Data and protected by the TLS tunnel

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

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

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

              // Optional additional X Method exchanges...

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

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

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

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

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

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

              // Optional additional Y Method exchanges...

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

        EAP Payload TLV
```

```
        [EAP-Type=Y] ->

                               <- Intermediate-Result-TLV (Success),
                                  Crypto-Binding TLV (Request),
                                  Result TLV (Success)

        Intermediate-Result-TLV (Success),
        Crypto-Binding TLV (Response),
        Result-TLV (Success) ->

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

        // TLS channel torn down (messages sent in clear text)

                               <- EAP-Success
```

C.7. Failed Crypto-binding

   The following exchanges show a failed crypto-binding validation. The
   conversation will appear as follows:

```
   Authenticating Peer      Authenticator
   -------------------      -------------
                            <- EAP-Request/
                            Identity
   EAP-Response/
   Identity (MyID1) ->
                            <- EAP-Request/
                            EAP-Type=TEAP, V=1
                            (TEAP Start, S bit set, Authority-ID)

   EAP-Response/
   EAP-Type=TEAP, V=1
   (TLS client_hello without
   PAC-Opaque extension)->
                            <- EAP-Request/
                            EAP-Type=TEAP, V=1
                            (TLS Server Key Exchange
                             TLS Server Hello Done)
   EAP-Response/
   EAP-Type=TEAP, V=1 ->
   (TLS Client Key Exchange
    TLS change_cipher_spec,
    TLS finished)
```

```
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS change_cipher_spec
                               TLS finished)
                               EAP-Payload-TLV[
                               EAP-Request/Identity])

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

     // First EAP Payload TLV is piggybacked to the TLS Finished as
        Application Data and protected by the TLS tunnel

  EAP-Payload TLV/
  EAP Identity Response ->

                         <-  EAP Payload TLV, EAP-Request,
                             (EAP-MSCHAPV2, Challenge)

  EAP Payload TLV, EAP-Response,
  (EAP-MSCHAPV2, Response) ->

                         <-  EAP Payload TLV, EAP-Request,
                             (EAP-MSCHAPV2, Success Request)

  EAP Payload TLV, EAP-Response,
  (EAP-MSCHAPV2, Success Response) ->

                         <- Intermediate-Result-TLV (Success),
                            Crypto-Binding TLV (Request),
                               Result TLV (Success)

     Intermediate-Result-TLV (Success),
     Result TLV (Failure)
     Error TLV with
     (Error Code = 2001) ->

  // TLS channel torn down
     (messages sent in clear text)

                         <- EAP-Failure
```

C.8. Sequence of EAP Method with Vendor-Specific TLV Exchange

   When TEAP is negotiated, with a sequence of EAP method followed by
   Vendor-Specific TLV exchange, the conversation will occur as follows:

      Authenticating Peer   Authenticator

```
        ------------------     -------------
                                <- EAP-Request/
                                Identity
        EAP-Response/
        Identity (MyID1) ->
                                <- EAP-Request/
                                EAP-Type=TEAP, V=1
                                (TEAP Start, S bit set, Authority-ID)

        EAP-Response/
        EAP-Type=TEAP, V=1
        (TLS client_hello)->
                                <- EAP-Request/
                                EAP-Type=TEAP, V=1
                                (TLS server_hello,
                                 TLS certificate,
                          [TLS server_key_exchange,]
                          [TLS certificate_request,]
                               TLS server_hello_done)

        EAP-Response/
        EAP-Type=TEAP, V=1
        ([TLS certificate,]
         TLS client_key_exchange,
        [TLS certificate_verify,]
         TLS change_cipher_spec,
         TLS finished) ->
                                 <- EAP-Request/
                                 EAP-Type=TEAP, V=1
                                 (TLS change_cipher_spec,
                                  TLS finished,
                                 EAP-Payload-TLV[
                                 EAP-Request/Identity])

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

        // First EAP Payload TLV is piggybacked to the TLS Finished as
           Application Data and protected by the TLS tunnel

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

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

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

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

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

                            <- Intermediate Result TLV (Success),
                             Crypto-Binding TLV (Request),
                             Vendor-Specific TLV,

     // Vendor Specific TLV exchange started after successful
        completion of previous method X. The Intermediate-Result
        and Crypto-Binding TLVs are sent with Vendor Specific TLV
        in next packet to minimize round-trips.

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

     Intermediate Result TLV (Success),
     Crypto-Binding TLV (Response),
     Vendor-Specific TLV ->

        // Optional additional Vendor-Specific TLV exchanges...

                            <- Vendor-Specific TLV

     Vendor Specific TLV ->
                            <- Result TLV (Success)

     Result-TLV (Success) ->

     // TLS channel torn down (messages sent in clear text)

                            <- EAP-Success
```

C.9. Peer Requests Inner Method After Server Sends Result TLV

   In the case where the peer is authenticated during Pahse 1 and server
   sends back result TLV, but the peers wants to request another inner
   method, the conversation will appear as follows:

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

     // Identity sent in the clear. May be a hint to help route
```

        the authentication request to EAP server, instead of the
        full user identity.

```
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TEAP Start, S bit set, Authority-ID)
        EAP-Response/
        EAP-Type=TEAP, V=1
        (TLS client_hello)->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS server_hello,
                               TLS certificate,
                              [TLS server_key_exchange,]
                              [TLS certificate_request,]
                               TLS server_hello_done)
        EAP-Response/
        EAP-Type=TEAP, V=1
        [TLS certificate,]
         TLS client_key_exchange,
        [TLS certificate_verify,]
         TLS change_cipher_spec,
         TLS finished ->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS change_cipher_spec,
                               TLS finished,
                               Crypto-Binding TLV (Request),
                                Result TLV (Success))

        // TLS channel established
           (TLV Payload messages sent within the TLS channel)

         Crypto-Binding TLV(Response),
         Request-Action TLV
         (Status=Failure, Action=Negotiate-EAP)->

                                 <- EAP-Payload-TLV
                                 [EAP-Request/Identity]

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

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

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

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

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

                              <- Intermediate Result TLV (Success),
                                    Crypto-Binding TLV (Request),
                                 Result TLV (Success)

      Intermediate Result TLV (Success),
      Crypto-Binding TLV (Response),
      Result-TLV (Success)) ->

      //TLS channel torn down
      (messages sent in clear text)

                              <- EAP-Success
```

C.10. Channel Binding

   The following exchanges show a successful TEAP authentication with
   basic password authentication and channel binding using Request-
   Action TLV, the conversation will appear as follows:

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

                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TEAP Start, S bit set, Authority-ID)

      EAP-Response/
      EAP-Type=TEAP, V=1
      (TLS client_hello with
       PAC-Opaque in SessionTicket extension)->

                               <- EAP-Request/
                               EAP-Type=TEAP, V=1
                               (TLS server_hello,
                               (TLS change_cipher_spec,
                                TLS finished)

      EAP-Response/
```

```
        EAP-Type=TEAP, V=1 ->
        (TLS change_cipher_spec,
         TLS finished)

        TLS channel established
        (messages sent within the TLS channel)

                              <- Basic-Password-Auth-Req TLV, Challenge

        Basic-Password-Auth-Resp TLV, Response with both
        user name and password) ->

        optional additional exchanges (new pin mode,
        password change etc.) ...

                              <- Crypto-Binding TLV (Request),
                                    Result TLV (Success),

        Crypto-Binding TLV(Response),
        Request-Action TLV
        (Status=Failure, Action=Process-TLV,
        TLV=Channel-Binding TLV)->

                                <- Channel-Binding TLV (Response),
                                      Result TLV (Success),

        Result-TLV (Success) ->

        TLS channel torn down
        (messages sent in clear text)

                              <- EAP-Success
```

Appendix D. Major Differences from Previous Revisions

D.1. Changes from -02

   1 Section 3.3.3, clarified protected termination and use of crypto-
      binding TLV.

   2 Section 3.5, changed Session ID to use tls-unique and added
      reference to RFC5247.

   3 Section 3.9, added the use of tls-unique to the certificate
      enrollment request.

4   Section 4.2.9, modified Request-Action TLV to inlcude Status code
    and optional TLVs.

5   Section 3.4, clarified that all authenticated Peer-Ids need to be
    exported.

6   Section 5.1, changed TLS Keying Material Exporter label to "teap
    seesion key seed".

7   Section 5.2, changed Intermediate Compound Key Derivation from MSK
    to EMSK generated by inner method.

8   Section 6, added missing IANA considerations.

9   Section 7.3, added more security considerations for separation of
    Phase 1 and Phase 2 servers.

10 Appendix C, updated examples with Request-Action TLV, channel
    binding, and sending certificate after TLS renegotiation.

D.2. Changes from -01

1   In Version Negotiation section, clarified what the peer needs to
    do if the supported version is higher than what the server
    proposed.

2   Section 3.2, clarified the requirement for using anonymous cipher
    suites.

3   Clarified that Crypto-binding TLV is always exchanged and
    validated, even without inner methods.

4   Section 3.4, clarified that all authenticated Peer-Ids need to be
    exported.

5   Clarified that channel-binding TLV can be used to transmit data
    bidirectionally.

6   Updated obsolete RFC references

7   Renumbered TLVs to eliminate gaps

8   Updated examples with basic password authentication TLVs.

9   Added Certificate Provisioning Within the Tunnel.

    10 Added Server Unauthenticated Provisioning Mode.

D.3. Changes from -00

    1 Changed protocol name to TEAP: Tunnel EAP Method

    2 Changed version of protocol to version 1

    3 Revised introduction

    4 Moved differences section to appendix

    5 Revised design goals section

    6 Revised PAC definition

    7 Revised protocol description to be in line with RFC 5077 PAC
       distribution

    8 Revised EAP Sequences Section

    9 Added section on PAC provisioning within tunnel

    10 Added outer TLVs to the message format

    11 Renumbered TLVs

    12 Included PAC TLVs

    13 Added Authority ID TLV

    14 Added PKCS#7 and server trust root TLV definitions

    15 Added PKCS#10 TLV

    16 PKCS#10 TLV

    17 Added EAP-Type and outer TLVs to crypto binding compound MAC

Authors' Addresses

   Hao Zhou
   Cisco Systems
   4125 Highlander Parkway
   Richfield, OH   44286
   US

   EMail: hzhou@cisco.com

   Nancy Cam-Winget
   Cisco Systems
   3625 Cisco Way
   San Jose, CA   95134
   US

   EMail: ncamwing@cisco.com

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

   EMail: jsalowey@cisco.com

   Stephen Hanna
   Juniper Networks
   79 Parsons Street
   Brighton, MA   02135
   US

   EMail: shanna@juniper.net