              The Flexible Authentication via Secure Tunneling
              Extensible Authentication Protocol Method (EAP-FAST)

Status of This Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard of any kind. Distribution of this
   memo is unlimited.

Copyright Notice

Abstract

   This document defines the Extensible Authentication Protocol (EAP)
   based Flexible Authentication via Secure Tunneling (EAP-FAST)
   protocol. EAP-FAST is an 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 peer and the EAP server.

Table of Contents

1. Introduction

   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 1.0 [RFC2246], TLS 1.1 [RFC4346], or
   a successor version of TLS, using the latest version supported by
   both parties. Once the tunnel is established, the protocol further
   exchanges data in the form of type, length, and value objects (TLV)
   to perform further authentication. EAP-FAST supports the TLS
   extension defined in [RFC4507] to support fast re-establishment of
   the secure tunnel without having to maintain per-session state on the
   server. [EAP-PROV] defines EAP-FAST-based mechanisms to provision
   the credential for this extension which is called a Protected Access
   Credential (PAC).

   EAP-FAST'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 seamlessly.

   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.

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 many
   servers acting as the authentication servers for many peers. It
   is also highly desirable for a peer to use the same shared secret
   to secure a tunnel much the same way it uses the username and
   password to gain access to the network. The protocol must
   facilitate the use of a single strong shared secret by the peer
   while enabling the servers to minimize the per user and device
   state it must cache and manage.

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. 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, at most, three components: a
      shared secret, an opaque element, and optionally other
      information. 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. Finally, a PAC may optionally include other
      information that may be useful to the peer. The opaque part of
      the PAC is the same type of data as the ticket in [RFC4507] and
      the shared secret is used to derive the TLS master secret.

2. Protocol Overview

EAP-FAST is an authentication protocol similar to EAP-TLS [RFC2716]
that enables mutual authentication and cryptographic context
establishment by using the TLS handshake protocol. EAP-FAST allows
for the established TLS tunnel to be used for further authentication
exchanges. EAP-FAST makes use of TLVs to carry out the inner
authentication exchanges. The tunnel is then used to protect weaker
inner authentication methods, which may be based on passwords, and to
communicate the results of the authentication.

EAP-FAST makes use of the TLS enhancements in [RFC4507] to enable an
optimized TLS tunnel session resume while minimizing server state.
The secret key used in EAP-FAST is referred to as the Protected
Access Credential key (or PAC-Key); the PAC-Key is used to mutually
authenticate the peer and the server when securing a tunnel. The
ticket is referred to as the Protected Access Credential opaque data
(or PAC-Opaque). The secret key and ticket used to establish the
tunnel may be provisioned through mechanisms that do not involve the
TLS handshake. It is RECOMMENDED that implementations support the
capability to distribute the ticket and secret key within the EAP-
FAST tunnel as specified in [EAP-PROV].

The EAP-FAST 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 EAP-FAST, 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 EAP-FAST usage is shown below:

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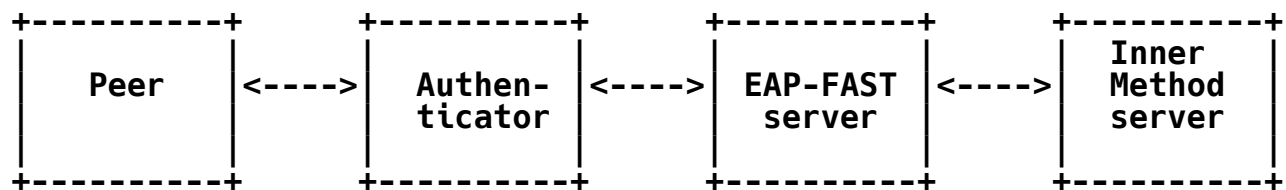
                    EAP-FAST Architectural Model

The entities depicted above are logical entities and may or may not
correspond to separate network components. For example, the EAP-FAST
server and inner method server might be a single entity; the
authenticator and EAP-FAST server might be a single entity; or the
functions of the authenticator, EAP-FAST server, and inner method

server might be combined into a single physical device. For example,
typical 802.11 deployments place the Authenticator in an access point
(AP) while a Radius server may provide the EAP-FAST 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 EAP-FAST server from the inner method server.

2.2. Protocol Layering Model

EAP-FAST packets are encapsulated within EAP; EAP in turn requires a
carrier protocol for transport. EAP-FAST packets encapsulate TLS,
which is then used to encapsulate user authentication information.
Thus, EAP-FAST 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                                 |
---------------------------------------------------------------
|                         EAP-FAST                              |
---------------------------------------------------------------
|                           EAP                                 |
---------------------------------------------------------------
|    Carrier Protocol (EAP over LAN, RADIUS, Diameter, etc.)    |
+---------------------------------------------------------------+
```
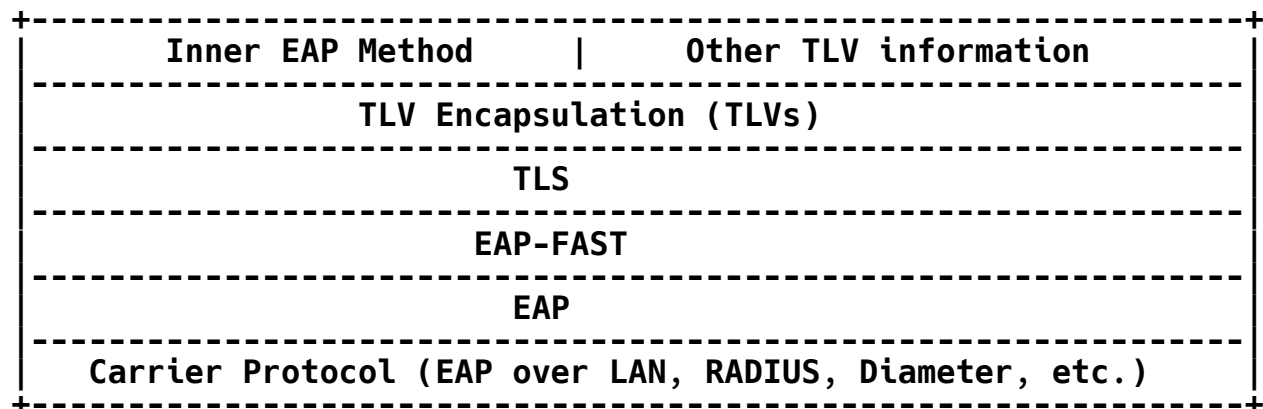
                        Protocol Layering Model

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 EAP-FAST protected tunnel must be encapsulated in a TLV layer.

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-FAST server.

3. EAP-FAST Protocol

   EAP-FAST authentication occurs in two phases. In the first phase,
   EAP-FAST 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, are the topic of this section. The format of
   EAP-FAST messages is given in Section 4 and the cryptographic
   calculations are given in Section 5.

3.1. Version Negotiation

   EAP-FAST packets contain a 3-bit version field, following the TLS
   Flags field, which enables EAP-FAST implementations to be backward
   compatible with previous versions of the protocol. This
   specification documents the EAP-FAST 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=EAP-FAST, 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=EAP-FAST, and the version
      number proposed by the EAP-FAST server.

      If the EAP-FAST peer does not support this version, it responds
      with an EAP-Response of EAP type=EAP-FAST and the highest
      supported version number.

      If the EAP-FAST server does not support the version number
      proposed by the EAP-FAST peer, it terminates the conversation.
      Otherwise the EAP-FAST conversation continues.

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

   The EAP-FAST version is not protected by TLS; and hence can be
   modified in transit. In order to detect a modification of the EAP-
   FAST version, the peers MUST exchange the EAP-FAST version number

received during version negotiation using the Crypto-Binding TLV
described in Section 4.2.8. 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 EAP-FAST version
negotiation.

3.2. EAP-FAST Authentication Phase 1: Tunnel Establishment

EAP-FAST is based on the TLS handshake [RFC2246] to establish an
authenticated and protected tunnel. The TLS version offered by the
peer and server MUST be TLS v1.0 or later. This version of the EAP-
FAST implementation MUST support the following TLS ciphersuites:

    TLS_RSA_WITH_RC4_128_SHA

    TLS_RSA_WITH_AES_128_CBC_SHA [RFC3268]

    TLS_DHE_RSA_WITH_AES_128_CBC_SHA [RFC3268]

Other ciphersuites MAY be supported. It is RECOMMENDED that
anonymous ciphersuites such as TLS_DH_anon_WITH_AES_128_CBC_SHA only
be used in the context of the provisioning described in [EAP-PROV].
Care must be taken to address potential man-in-the-middle attacks
when ciphersuites that do not provide authenticated tunnel
establishment are used. During the EAP-FAST Phase 1 conversation the
EAP-FAST endpoints MAY negotiate TLS compression.

The EAP server initiates the EAP-FAST conversation with an EAP
request containing an EAP-FAST/Start packet. This packet includes a
set Start (S) bit, the EAP-FAST version as specified in Section 3.1,
and an authority identity. The TLS payload in the initial packet is
empty. The authority identity (A-ID) 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 EAP-FAST the conversation continues with the peer sending an
EAP-Response packet with EAP type of EAP-FAST 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 EAP-FAST version negotiation is successful then the
EAP-FAST 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 EAP-FAST Phase 2 MAY be sent along with
server-finished handshake message to reduce the number of round
trips.

After the TLS session is established, another EAP exchange MAY occur
within the tunnel to authenticate the EAP peer. EAP-FAST
implementations MUST support client authentication during tunnel

establishment using the TLS ciphersuites specified in Section 3.2.
EAP-FAST implementations SHOULD also support the immediate
renegotiation of a TLS session to initiate a new handshake message
exchange under the protection of the current ciphersuite. This
allows support for protection of the peer's identity. Note that the
EAP peer does not need to authenticate as part of the TLS exchange,
but can alternatively be authenticated through additional EAP
exchanges carried out in Phase 2.

The EAP-FAST tunnel protects peer identity information 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.

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

EAP-FAST session resumption is achieved in the same manner TLS
achieves session resume. To support session resumption, the server
and peer must minimally cache the SessionID, master secret, and
ciphersuite. The peer attempts to resume a session by including a
valid SessionID from a previous handshake in its ClientHello message.
If the server finds a match for the SessionID and is willing to
establish a new connection using the specified session state, the
server will respond with the same SessionID and proceed with the EAP-
FAST Authentication Phase 1 tunnel establishment based on a TLS
abbreviated handshake. After a successful conclusion of the EAP-FAST
Authentication Phase 1 conversation, the conversation then continues
on to Phase 2.

3.2.2. TLS Session Resume Using a PAC

EAP-FAST supports the resumption of sessions based on client-side
state using techniques described in [RFC4507]. This version of EAP-
FAST does not support the provisioning of a ticket through the use of
the SessionTicket handshake message. Instead it supports the
provisioning of a ticket called a Protected Access Credential (PAC)
as described in [EAP-PROV]. 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 a 32-octet key used by the peer to establish the
    EAP-FAST Phase 1 tunnel. This key is used to derive the TLS
    premaster secret as described in Section 5.1. The PAC-Key is
    randomly generated by the EAP server to produce a strong entropy
    32-octet key. The PAC-Key is a secret and MUST be treated
    accordingly. For example, as the PAC-Key is a separate component
    provisioned by the server to establish a secure tunnel, the
    server may deliver this component protected by a secure channel,
    and it must be stored securely by the peer.

2. PAC-Opaque: this is a variable length field that is sent to the
    EAP server during the EAP-FAST Phase 1 tunnel establishment. 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. For example, the PAC-Opaque
    includes the PAC-Key and 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.

3. PAC-Info: this is a 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.

The use of the PAC is based on the SessionTicket extension defined in
[RFC4507]. The EAP server initiates the EAP-FAST conversation as
normal. Upon receiving the A-ID 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
EAP-FAST that the peer include an empty Session ID in a ClientHello
containing a PAC-Opaque. EAP-FAST does not currently support the
SessionTicket Handshake message so an empty SessionTicket extension
MUST NOT be included in the ClientHello. If the PAC-Opaque included
in the SessionTicket extension is valid and the EAP server permits
the abbreviated TLS handshake, it will select the ciphersuite allowed
to be used from information within the PAC 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 [RFC4507] until the handshake completes or a fatal error
occurs. After the abbreviated handshake completes, the peer and
server are ready to commence Phase 2. Note that when a PAC is used,

the TLS master secret is calculated from the PAC-Key, client random,
and server random as described in Section 5.1.

Specific details for the Tunnel PAC format, provisioning and security
considerations are best described in [EAP-PROV]

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 a empty Session ID
or a Session ID that is different than in the ClientHello, the server
may be falling 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 ciphersuites 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 A.

3.3. EAP-FAST Authentication Phase 2: Tunneled Authentication

The second portion of the EAP-FAST 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 protected termination exchange described in
Section 3.3.2. 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 through the use of the
Request-Action TLV.

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. EAP-FAST 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 EAP-FAST 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 4.2.6. If more than one method is going to be executed in
   the tunnel then, upon completion of a method, a 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 4.2.8 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. In the case where only one EAP method is executed in the
   tunnel, the Intermediate-Result TLV MUST NOT be sent with the Result
   TLV. In this case, the status of the inner EAP method is represented
   by the final Result TLV, which also represents the result of the
   whole EAP-FAST conversation. This is to maintain backward
   compatibility with existing implementations.

   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. Protected Termination and Acknowledged Result Indication

   A successful EAP-FAST Phase 2 conversation MUST always end in a
   successful Result TLV exchange. An EAP-FAST server may initiate the
   Result TLV exchange without initiating any EAP conversation in EAP-
   FAST 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. The format of the Result TLV is described in Section 4.2.2.

A server initiates a successful protected termination exchange by
sending a Result TLV indicating success. The server may send the
Result TLV along with an Intermediate-Result TLV and a Crypto-Binding
TLV. If the peer requires nothing more from the server it will
respond with a Result TLV indicating success accompanied by an
Intermediate-Result TLV and Crypto-Binding TLV if necessary. The
server then tears down the tunnel and sends a clear text EAP-Success.

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 along with
the Result TLV indicating what EAP Success/Failure result the peer
would expect if the requested action is not granted. The value of
the Request-Action TLV 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 may be determined based on the types of
credentials used during either the EAP-FAST tunnel creation or
authentication.

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 [RFC3280] (updated by [RFC4630]):

> 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. EAP-FAST Session Identifier

The EAP session identifier is constructed using the random values
provided by the peer and server during the TLS tunnel establishment.
The Session-Id is defined as follows:

```
   Session-Id  = 0x2B || client_random || server_random)
 client_random = 32 byte nonce generated by the peer
 server_random = 32 byte nonce generated by the server
```

3.6. Error Handling

EAP-FAST uses the following error handling rules summarized below:

1. Errors in the TLS layer are communicated via TLS alert messages
    in all phases of EAP-FAST.

2. The Intermediate-Result TLVs carry success or failure indications
    of the individual EAP methods in EAP-FAST 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 EAP-FAST server detects an error at any point in the TLS
Handshake or the TLS layer, the server SHOULD send an EAP-FAST
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 an EAP-FAST response to an alert message. The EAP-Response

packet sent by the peer may encapsulate a TLS ClientHello handshake
message, in which case the EAP-FAST server MAY allow the EAP-FAST
conversation to be restarted, or it MAY contain an EAP-FAST 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 EAP-
FAST server whether to allow restarts, and if so, how many times the
conversation can be restarted. An EAP-FAST Server implementing
restart capability SHOULD impose a limit on the number of restarts,
so as to protect against denial-of-service attacks.

If the EAP-FAST peer detects an error at any point in the TLS layer,
the EAP-FAST peer should send an EAP-FAST response encapsulating a
TLS record containing the appropriate TLS alert message. The server
may restart the conversation by sending an EAP-FAST request packet
encapsulating the TLS HelloRequest handshake message. The peer may
allow the EAP-FAST conversation to be restarted or it may terminate
the conversation by sending an EAP-FAST 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 EAP-FAST implementation MUST provide its own
support for fragmentation and reassembly.

Since EAP is an 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.

EAP-FAST 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 EAP-FAST Start (S) bits. The L
flag is set to indicate the presence of the four-octet TLS Message
Length field, and MUST be set for the first fragment of a fragmented
TLS message or set of messages. The M flag is set on all but the
last fragment. The S flag is set only within the EAP-FAST start
message sent from the EAP server to the peer. The TLS Message Length
field is four octets, and provides the total length of the TLS
message or set of messages that is being fragmented; this simplifies
buffer allocation.

When an EAP-FAST peer receives an EAP-Request packet with the M bit
set, it MUST respond with an EAP-Response with EAP-Type of EAP-FAST
and no data. This serves as a fragment ACK. The EAP server must
wait until it receives the EAP-Response before sending another
fragment. In order to prevent errors in processing of fragments, the
EAP server MUST increment the Identifier field for each fragment
contained within an EAP-Request, and the peer must include this
Identifier value in the fragment ACK contained within the EAP-
Response. Retransmitted fragments will contain the same Identifier
value.

Similarly, when the EAP-FAST server receives an EAP-Response with the
M bit set, it must respond with an EAP-Request with EAP-Type of EAP-
FAST 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.

4. Message Formats

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

4.1. EAP-FAST Message Format

   A summary of the EAP-FAST 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       |            Data...            +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
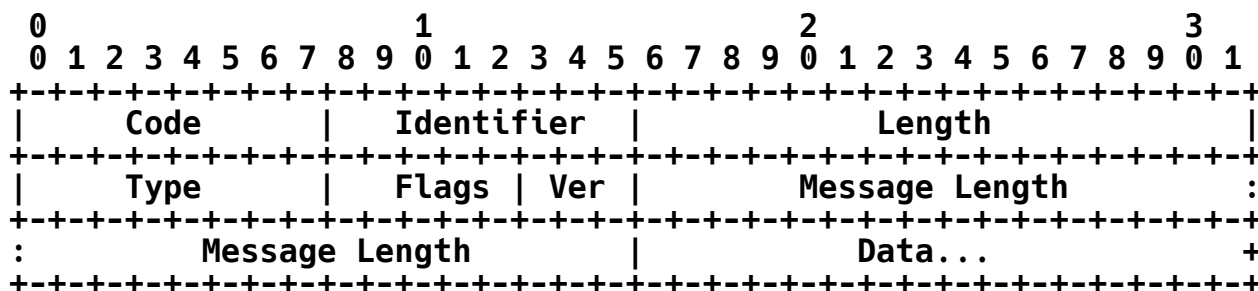
   Code

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

   Type

      43 for EAP-FAST

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   EAP-FAST start; set in an EAP-FAST 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 EAP-FAST.

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.

Data

In the case of an EAP-FAST Start request (i.e., when the S bit
is set) the Data field consists of the A-ID described in
Section 4.1.1. In other cases, when the Data field is present,
it consists of an encapsulated TLS packet in TLS record format.
An EAP-FAST packet with Flags and Version fields, but with zero
length data field, is used to indicate EAP-FAST acknowledgement
for either a fragmented message, a TLS Alert message or a TLS
Finished message.

4.1.1. Authority ID Data

```
 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 (0x04)        |              Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
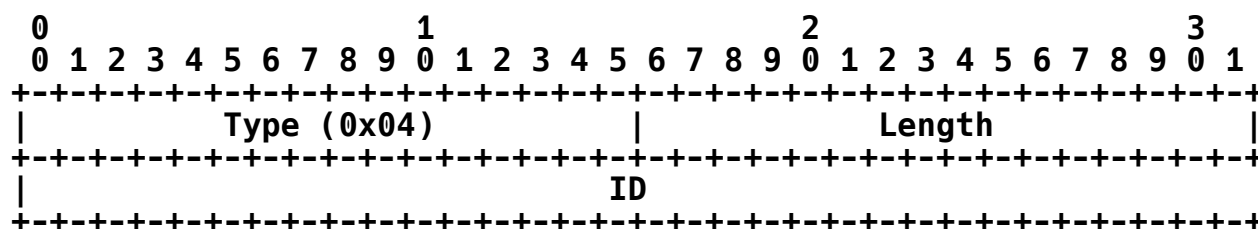
   Type

      The Type field is two octets. It is set to 0x0004 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. It should be unique across
      the deployment.

4.2. EAP-FAST 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.

   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:

    Result TLV
    NAK TLV
    Error TLV
    EAP-Payload TLV
    Intermediate-Result TLV
    Crypto-Binding TLV
    Request-Action 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...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
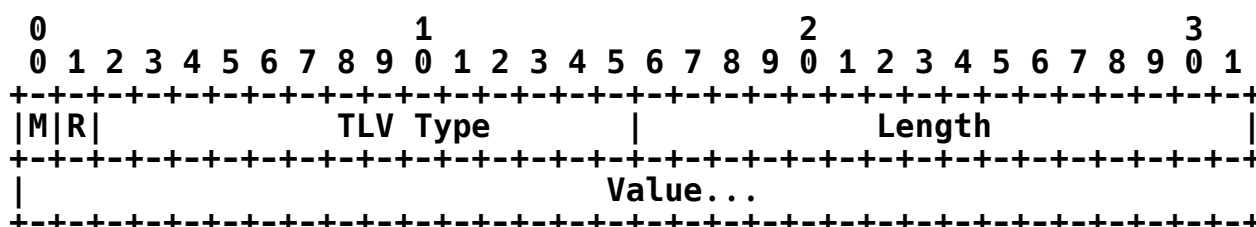
   M

      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   Reserved
          1   Reserved
          2   Reserved
          3   Result TLV              (Section 4.2.2)
          4   NAK TLV                 (Section 4.2.3)
          5   Error TLV               (Section 4.2.4)
          7   Vendor-Specific TLV     (Section 4.2.5)
          9   EAP-Payload TLV         (Section 4.2.6)
          10  Intermediate-Result TLV (Section 4.2.7)
          11  PAC TLV                 [EAP-PROV]
          12  Crypto-Binding TLV      (Section 4.2.8)
          18  Server-Trusted-Root TLV [EAP-PROV]
          19  Request-Action TLV      (Section 4.2.9)
          20  PKCS#7 TLV              [EAP-PROV]
```

   Length

      The length of the Value field in octets.

   Value

      The value of the TLV.

4.2.2. Result TLV

   The Result TLV provides support for acknowledged success and failure
   messages for protected termination within EAP-FAST. 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 3.3.2
   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
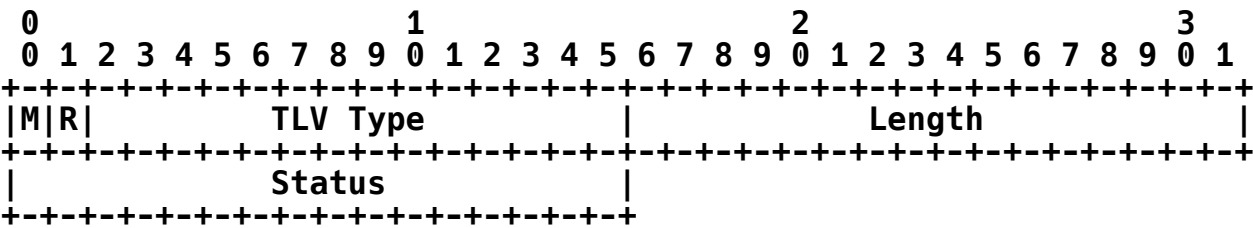
      Reserved, set to zero (0)

   TLV Type

      3 for Result TLV

   Length

      2

   Status

      The Status field is two octets. Values include:

      1 Success

      2 Failure

4.2.3. NAK TLV

   The NAK TLV allows a peer to detect TLVs that are not supported by
   the other peer. An EAP-FAST 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
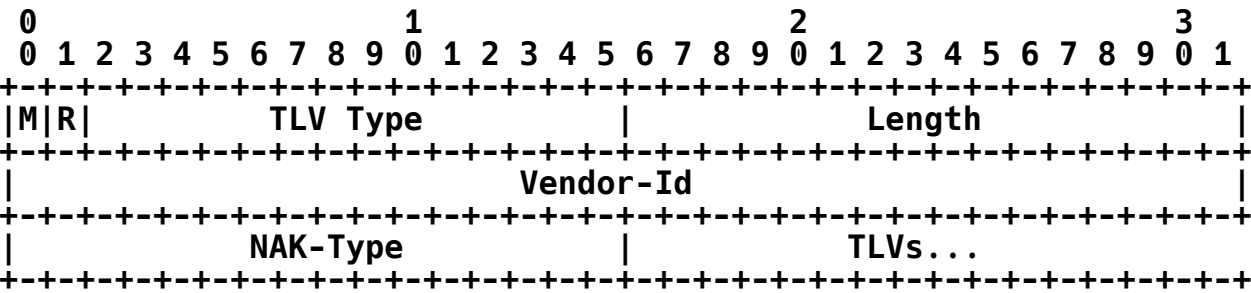
      Reserved, set to zero (0)

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.

4.2.4. Error TLV

   The Error TLV allows an EAP peer or server to indicate errors to the
   other party. An EAP-FAST 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
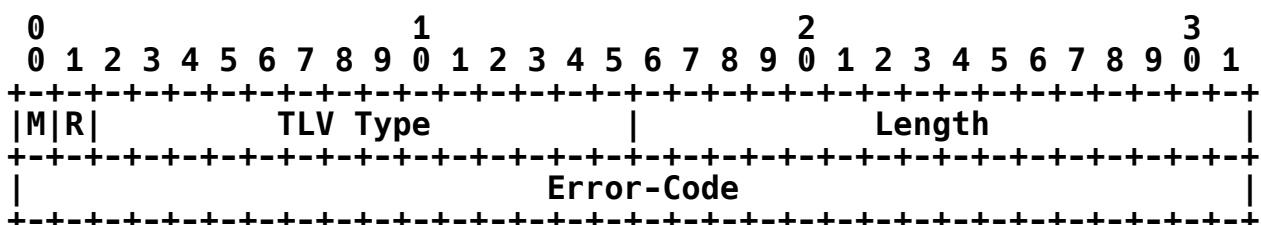
      Reserved, set to zero (0)

   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

4.2.5. 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...                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
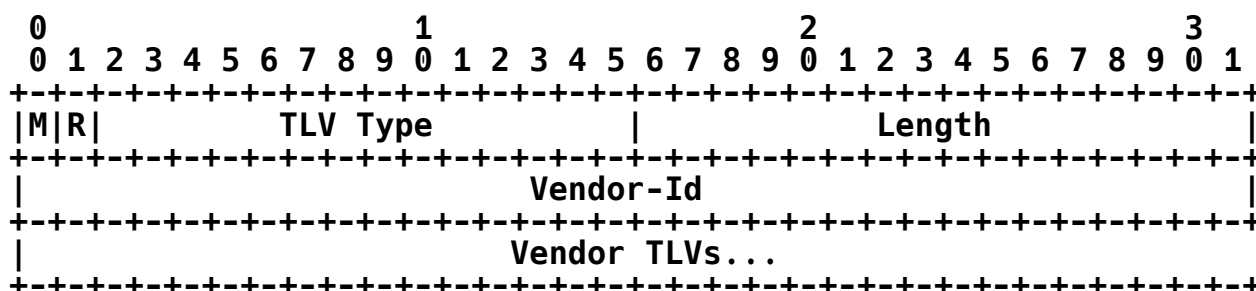
   M

      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.

4.2.6. 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
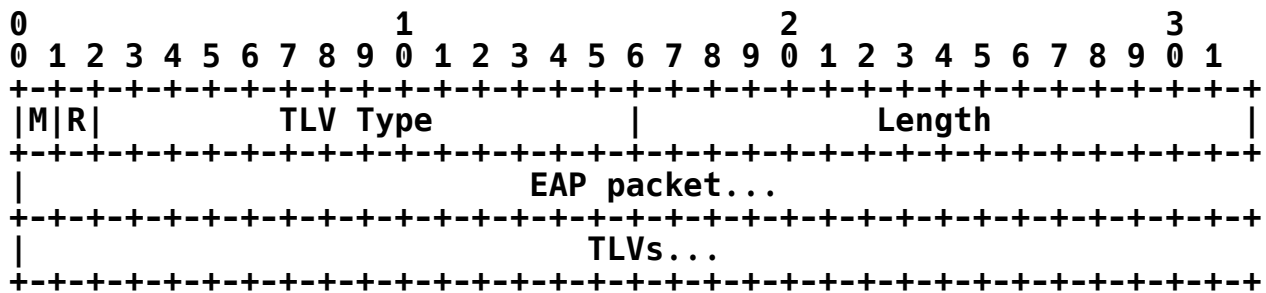
      Reserved, set to zero (0)

   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 field contains a list of zero or more 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.

4.2.7. 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.

4.2.8. 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 EAP-FAST
   version negotiated before TLS tunnel establishment, see Section 3.1.

   The Crypto-Binding TLV MUST be included with the Intermediate-Result
   TLV to perform Cryptographic Binding after each successful EAP method
   in a sequence of EAP methods. The Crypto-Binding TLV can be issued
   at other times as well.

   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 fail, 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
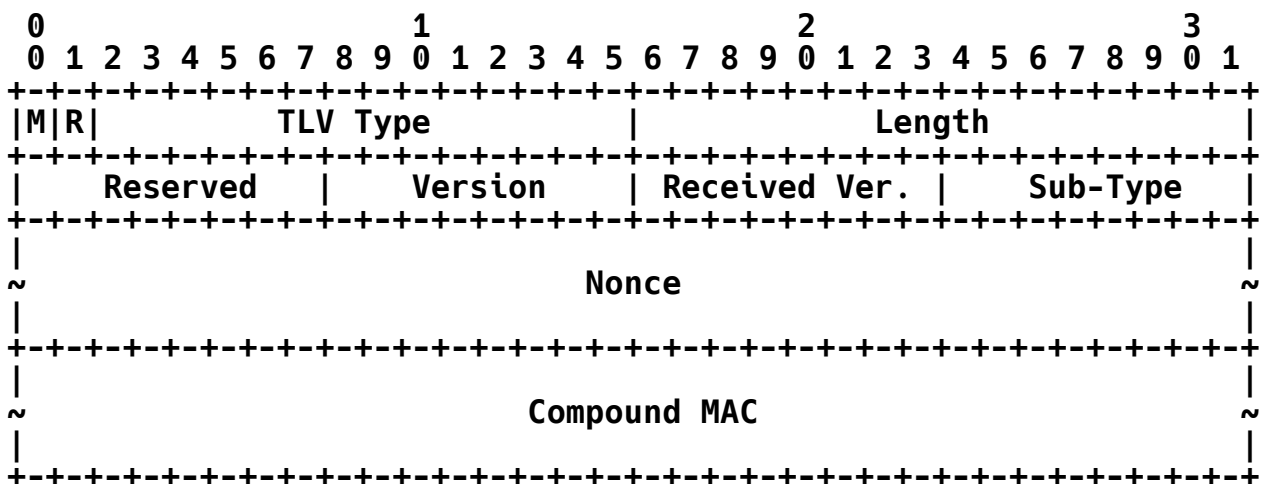
   Reserved, set to zero (0)

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 EAP-FAST, 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.

4.2.9. Request-Action TLV

   The Request-Action TLV MAY be sent by the peer along with a Result
   TLV 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 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             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Action              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   M

      Mandatory set to one (1)

   R
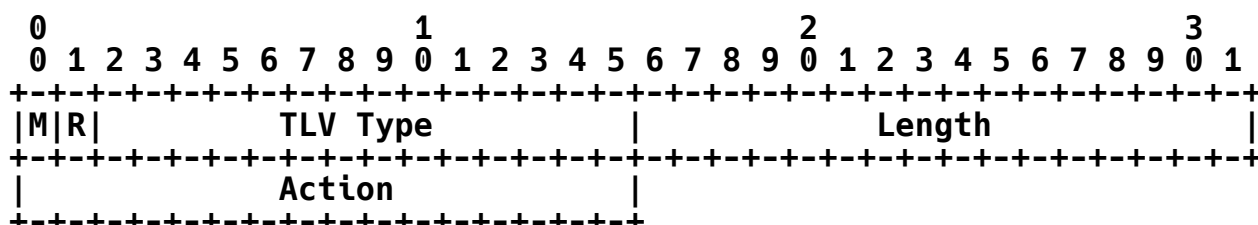
      Reserved, set to zero (0)

   TLV Type

      19 for Request-Action TLV

   Length

      2

   Action

      The Action field is two octets. Values include:

         Process-TLV

         Negotiate-EAP

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 an EAP-FAST Request, Response is an EAP-FAST
   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	0-1	0-1	Intermediate-Result
0-1	0-1	0	0	EAP-Payload
0-1	0-1	1	1	Result
0-1	0-1	0-1	0-1	Crypto-Binding
0+	0+	0+	0+	Error
0+	0+	0	0	NAK
0+	0+	0+	0+	Vendor-Specific [NOTE1]
0	0-1	0-1	0-1	Request-Action

   [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. EAP-FAST Authentication Phase 1: Key Derivations

   The EAP-FAST Authentication tunnel key is calculated similarly to the
   TLS key calculation with an additional 40 octets (referred to as the
   session_key_seed) generated. The additional session_key_seed is used
   in the Session Key calculation in the EAP-FAST Tunneled
   Authentication conversation.

To generate the key material required for the EAP-FAST Authentication
tunnel, the following construction from [RFC4346] is used:

```
key_block = PRF(master_secret, "key expansion",
    server_random + client_random)
```

where '+' denotes concatenation.

The PRF function used to generate keying material is defined by
[RFC4346].

For example, if the EAP-FAST Authentication employs 128-bit RC4 and
SHA1, the key_block is 112 octets long and is partitioned as follows:

```
client_write_MAC_secret[20]
server_write_MAC_secret[20]
client_write_key[16]
server_write_key[16]
client_write_IV[0]
server_write_IV[0]
session_key_seed[40]
```

The session_key_seed is used by the EAP-FAST Authentication Phase 2
conversation to both cryptographically bind the inner method(s) to
the tunnel as well as generate the resulting EAP-FAST session keys.
The other quantities are used as they are defined in [RFC4346].

The master_secret is generated as specified in TLS unless a PAC is
used to establish the TLS tunnel. When a PAC is used to establish
the TLS tunnel, the master_secret is calculated from the specified
client_random, server_random, and PAC-Key as follows:

```
master_secret = T-PRF(PAC-Key, "PAC to master secret label hash",
    server_random + client_random, 48)
```

where T-PRF is described in Section 5.5.

5.2. Intermediate Compound Key Derivations

The session_key_seed derived as part of EAP-FAST Phase 2 is used in
EAP-FAST 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 n inner EAP methods. The
inner EAP method(s) may provide Master Session Keys, MSK1..MSKn,
corresponding to inner methods 1 through n. 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 MSK, then MSKi 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] = T-PRF(S-IMCK[j-1], "Inner Methods Compound Keys",
        MSK[j], 60)
    S-IMCK[j] = first 40 octets of IMCK[j]
    CMK[j] = last 20 octets of IMCK[j]
```

where T-PRF is described in Section 5.5.

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 EAP-
FAST Phase 1 and EAP-FAST Phase 2 conversations. After each
successful inner EAP authentication, EAP MSKs are cryptographically
combined with key material from EAP-FAST 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 4.2.8,
which helps provide assurance that the same entities are involved in
all communications in EAP-FAST. 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-SHA1( CMK, Crypto-Binding TLV )
```

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

5.4. EAP Master Session Key Generation

   EAP-FAST 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 EAP-FAST Phase 1.
   The resulting MSK and EMSK are generated as part of the IMCKn key
   hierarchy as follows:

       MSK = T-PRF(S-IMCK[j], "Session Key Generating Function", 64)
       EMSK = T-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 EAP-FAST 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.

5.5. T-PRF

   EAP-FAST employs the following PRF prototype and definition:

       T-PRF = F(key, label, seed, outputlength)

   Where label is intended to be a unique label for each different use
   of the T-PRF. The outputlength parameter is a two-octet value that
   is represented in big endian order. Also note that the seed value
   may be optional and may be omitted as in the case of the MSK
   derivation described in Section 5.4.

To generate the desired outputlength octets of key material, the
T-PRF is calculated as follows:

    S = label + 0x00 + seed
    T-PRF output = T1 + T2 + T3 + ... + Tn
    T1 = HMAC-SHA1 (key, S + outputlength + 0x01)
    T2 = HMAC-SHA1 (key, T1 + S + outputlength + 0x02)
    T3 = HMAC-SHA1 (key, T2 + S + outputlength + 0x03)
    Tn = HMAC-SHA1 (key, Tn-1 + S + outputlength + 0xnn)

where '+' indicates concatenation. Each Ti generates 20-octets of
keying material. The last Tn may be truncated to accommodate the
desired length specified by outputlength.

6. IANA Considerations

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

EAP-FAST has already been assigned the EAP Method Type number 43.

The document defines a registry for EAP-FAST TLV types, which may be
assigned by Specification Required as defined in [RFC2434].
Section 4.2 defines the TLV types that initially populate the
registry. A summary of the EAP-FAST TLV types is given below:

0 Reserved
1 Reserved
2 Reserved
3 Result TLV
4 NAK TLV
5 Error TLV
7 Vendor-Specific TLV
9 EAP-Payload TLV
10 Intermediate-Result TLV
11 PAC TLV [EAP-PROV]
12 Crypto-Binding TLV
18 Server-Trusted-Root TLV [EAP-PROV]
19 Request-Action TLV
20 PKCS#7 TLV [EAP-PROV]

The Error-TLV defined in Section 4.2.4 requires an error-code. EAP-
FAST Error-TLV error-codes are assigned based on specifications
required as defined in [RFC2434]. The initial list of error codes is
as follows:

        2001 Tunnel_Compromise_Error

        2002 Unexpected_TLVs_Exchanged

   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 [RFC2434]. The initial actions defined are:

      1 Process-TLV

      2 Negotiate-EAP

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

7. Security Considerations

   EAP-FAST 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 EAP-
   FAST is defined in the EAP [RFC3748].

7.1. Mutual Authentication and Integrity Protection

   EAP-FAST 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 EAP-
   FAST 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 EAP-FAST, 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 EAP-FAST
   as an authentication method does not limit the potential inner
   authentication methods, so EAP-FAST 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 EAP-FAST 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

   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 EAP-FAST MAY be used. The EAP-FAST
   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.

7.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies

   EAP-FAST 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, EAP-FAST 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 with EAP-FAST, 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 EAP-FAST as well.

   EAP-FAST was designed with a focus on protected authentication
   methods that typically rely on weak credentials, such as password-
   based secrets. To that extent, the EAP-FAST 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 EAP-FAST. 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 an may not have any relevance to the authenticated
identity. EAP-FAST 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 EAP-FAST 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 EAP-FAST 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 EAP-FAST 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 EAP-FAST Phase 2 instead of using TLS
handshake renegotiation.

7.4.2. Dictionary Attack Resistance

EAP-FAST was designed with a focus on protected authentication
methods that typically rely on weak credentials, such as password-
based secrets. EAP-FAST 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 EAP-FAST.
EAP-FAST provides protection from man-in-the-middle attacks even if a
deployment chooses to execute inner EAP methods both with and without
EAP-FAST protection, EAP-FAST prevents this attack in two ways:

   1. By using the PAC-Key to mutually authenticate the peer and server
       during EAP-FAST 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
   EAP-FAST MUST validate that an identity obtained in the PAC-Opaque
   field matches at minimum one of the identities provided in the EAP-
   FAST Phase 2 authentication method. This validation provides another
   binding to ensure that the intended peer (based on identity) has
   successfully completed the EAP-FAST 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, EAP-FAST provides
   protection against these attacks. Once the peer and AS initiate the
   EAP-FAST Authentication Phase 2, compliant EAP-FAST implementations
   must silently discard all clear text EAP messages, unless both the
   EAP-FAST 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 3.3.2.

   The success/failure decisions within the EAP-FAST tunnel indicate the
   final decision of the EAP-FAST authentication conversation. After a
   success/failure result has been indicated by a protected mechanism,
   the EAP-FAST 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 an EAP-FAST protected EAP Success or
EAP Failure packet should not be a final packet in an EAP-FAST
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. 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 EAP-FAST 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 [RFC4507]. Thus, the
security considerations defined by [RFC4507] 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 EAP-FAST
                             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: No, but TLVs could be defined for this.

   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.SP800-57].
       [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 EAP-FAST design and protocol specification is based on the ideas and hard efforts of 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, and Simon Josefsson. Helpful review comments were provided by Russ Housley, Jari Arkko, Bernard Aboba, Ilan Frenkel, and Jeremy Steiglitz.

9. References

9.1. Normative References

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

[RFC2246]   Dierks, T. and C. Allen, "The TLS Protocol
                     Version 1.0", RFC 2246, January 1999.

[RFC2434]   Narten, T. and H. Alvestrand, "Guidelines for
                     Writing an IANA Considerations Section in RFCs",
                     BCP 26, RFC 2434, October 1998.

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

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

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

[RFC4507]   Salowey, J., Zhou, H., Eronen, P., and H.
                     Tschofenig, "Transport Layer Security (TLS)
                     Session Resumption without Server-Side State",
                     RFC 4507, May 2006.

9.2. Informative References

   [EAP-PROV]   Cam-Winget, N., "Dynamic Provisioning using EAP-
                           FAST", Work in Progress, January 2007.

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

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

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

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

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

   [RFC4630]   Housley, R. and S. Santesson, "Update to
                           DirectoryString Processing in the Internet X.509
                           Public Key Infrastructure Certificate and
                           Certificate Revocation List (CRL) Profile",
                           RFC 4630, August 2006.

Appendix A. Examples

   In the following examples the version field in EAP Fast is always
   assumed to be 1. The S, M, and L bits are assumed to be 0 unless
   otherwise specified.

A.1. Successful Authentication

   The following exchanges show a successful EAP-FAST authentication
   with optional PAC refreshment; the conversation will appear as
   follows:

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

                               <- EAP-Request/EAP-FAST
                               (S=1, A-ID)

    EAP-Response/EAP-FAST
    (TLS client_hello with
     PAC-Opaque in SessionTicket extension)->

                               <- EAP-Request/EAP-FAST
                               (TLS server_hello,
                                TLS change_cipher_spec,
                                TLS finished)

    EAP-Response/EAP-FAST
    (TLS change_cipher_spec,
     TLS finished) ->

    TLS channel established
    (Subsequent messages sent within the TLS channel,
                         encapsulated within EAP-FAST)

                            <- EAP Payload TLV
                            (EAP-Request/EAP-GTC(Challenge))

    EAP Payload TLV (EAP-Response/
    EAP-GTC(Response with both
    user name and password)) ->

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

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


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

                                 <- Result TLV (Success)
                                    [Optional PAC TLV]

        Result TLV (Success)
        [PAC TLV Acknowledgment] ->

        TLS channel torn down
        (messages sent in clear text)

                                 <- EAP-Success
```

A.2. Failed Authentication

   The following exchanges show a failed EAP-FAST 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-FAST
                                 (S=1, A-ID)

        EAP-Response/EAP-FAST
        (TLS client_hello with
         PAC-Opaque in SessionTicket extension)->

                                 <- EAP-Request/EAP-FAST
                                 (TLS server_hello,
                                  TLS change_cipher_spec,
                                  TLS finished)

        EAP-Response/EAP-FAST
        (TLS change_cipher_spec,
         TLS finished) ->
```

```
              TLS channel established
              (Subsequent messages sent within the TLS channel,
                                 encapsulated within EAP-FAST)

                                 <- EAP Payload TLV (EAP-Request/
                                    EAP-GTC (Challenge))

              EAP Payload TLV (EAP-Response/
              EAP-GTC (Response with both
              user name and password)) ->

                                 <- EAP Payload TLV (EAP-Request/
                                    EAP-GTC (error message))

              EAP Payload TLV (EAP-Response/
              EAP-GTC (empty data packet to
              acknowledge unrecoverable error)) ->

                                 <- Result TLV (Failure)

              Result TLV (Failure) ->

              TLS channel torn down
              (messages sent in clear text)

                                 <- EAP-Failure
```

A.3. Full TLS Handshake using Certificate-based Ciphersuite

   In the case where an abbreviated TLS handshake is tried and failed,
   and a fallback to certificate-based full TLS handshake occurs within
   EAP-FAST 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-FAST
                               (S=1, A-ID)
```

```
     EAP-Response/EAP-FAST
     (TLS client_hello
      with PAC-Opaque extension)->

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

                               <- EAP-Request/EAP-FAST
                               (TLS server_hello,
                                TLS certificate,
                               [TLS server_key_exchange,]
                               [TLS certificate_request,]
                                TLS server_hello_done)
     EAP-Response/EAP-FAST
     ([TLS certificate,]
      TLS client_key_exchange,
     [TLS certificate_verify,]
      TLS change_cipher_spec,
      TLS finished) ->
                               <- EAP-Request/EAP-FAST
                               (TLS change_cipher_spec,
                                TLS finished,
                                EAP-Payload-TLV
                                (EAP-Request/Identity))

     // TLS channel established
        (Subsequent messages sent within the TLS channel,
                            encapsulated within EAP-FAST)

     // 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/Method X)

     EAP-Payload-TLV
     (EAP-Response/Method X) ->
```

```
      // Method X exchanges followed by Protected Termination

                                <- Crypto-Binding TLV (Version=1,
                                EAP-FAST Version=1, Nonce,
                                CompoundMAC),
                                Result TLV (Success)

      Crypto-Binding TLV (Version=1,
      EAP-FAST Version=1, Nonce,
      CompoundMAC),
      Result-TLV (Success) ->

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

                                <- EAP-Success
```

A.4. Client Authentication during Phase 1 with Identity Privacy

   In the case where a certificate-based TLS handshake occurs within
   EAP-FAST Phase 1, and client certificate authentication and identity
   privacy is desired, 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-FAST
                                (S=1, A-ID)
      EAP-Response/EAP-FAST
      (TLS client_hello)->
                                <- EAP-Request/EAP-FAST
                                (TLS server_hello,
                                 TLS certificate,
                                [TLS server_key_exchange,]
                                [TLS certificate_request,]
                                 TLS server_hello_done)
      EAP-Response/EAP-FAST
      (TLS client_key_exchange,
       TLS change_cipher_spec,
       TLS finished) ->
```

```
                            <- EAP-Request/EAP-FAST
                            (TLS change_cipher_spec,
                             TLS finished,TLS Hello-Request)

   // TLS channel established
      (Subsequent messages sent within the TLS channel,
                          encapsulated within EAP-FAST)

   // TLS Hello-Request is piggybacked to the TLS Finished as
      Handshake Data and protected by the TLS tunnel

   // Subsequent messages are protected by the TLS Tunnel

   EAP-Response/EAP-FAST
   (TLS client_hello) ->


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

                            <- EAP-Request/EAP-FAST
                              (TLS change_cipher_spec,
                               TLS finished,
                               Result TLV (Success))

   EAP-Response/EAP-FAST
   (Result-TLV (Success)) ->

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

                            <- EAP-Success
```

A.5. Fragmentation and Reassembly

   In the case where EAP-FAST fragmentation is required, the
   conversation will appear as follows:

```
      Authenticating Peer        Authenticator
      -------------------        -------------
                                 <- EAP-Request/
                                 Identity
      EAP-Response/
      Identity (MyID) ->
                                 <- EAP-Request/EAP-FAST
                                 (S=1, A-ID)

      EAP-Response/EAP-FAST
      (TLS client_hello)->
                                 <- EAP-Request/EAP-FAST
                                 (L=1,M=1, TLS server_hello,
                                  TLS certificate,
                                 [TLS server_key_exchange,]
                                 [TLS certificate_request,])


      EAP-Response/EAP-FAST ->

                                  <- EAP-Request/EAP-FAST
                                  (M=1,
                                  [TLS certificate_request(con't),])
      EAP-Response/EAP-FAST ->
                                  <- EAP-Request/EAP-FAST
                                  ([TLS certificate_request(con't),]
                                  TLS server_hello_done)
      EAP-Response/EAP-FAST,
      (L=1,M=1,[TLS certificate,])->

                                  <- EAP-Request/EAP-FAST
      EAP-Response/EAP-FAST
      ([TLS certificate(con't),]
       TLS client_key_exchange,
      [TLS certificate_verify,]
       TLS change_cipher_spec,
       TLS finished))->
                                 <- EAP-Request/EAP-FAST
                                 ( TLS change_cipher_spec,
                                  TLS finished,
                                  EAP-Payload-TLV
                                  (EAP-Request/Identity))
```

```
         // TLS channel established
            (Subsequent messages sent within the TLS channel,
                                  encapsulated within EAP-FAST)

         // 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/Method X)

         EAP-Payload-TLV
         (EAP-Response/Method X) ->

         // Method X exchanges followed by Protected Termination

                                   <- Crypto-Binding TLV (Version=1,
                                   EAP-FAST Version=1, Nonce,
                                   CompoundMAC),
                                   Result TLV (Success)

         Crypto-Binding TLV (Version=1,
         EAP-FAST Version=1, Nonce,
         CompoundMAC),
         Result-TLV (Success) ->

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

                                   <- EAP-Success
```

A.6. Sequence of EAP Methods

   Where EAP-FAST 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-FAST
                                 (S=1, A-ID)
```

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

      // TLS channel established
         (Subsequent messages sent within the TLS channel,
                           encapsulated within EAP-FAST)

      // 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/Method X)

      EAP-Payload-TLV
      (EAP-Response/Method X) ->

            // Optional additional X Method exchanges...

                                <- EAP-Payload-TLV
                                 (EAP-Request/Method X)

      EAP-Payload-TLV
      (EAP-Response/EAP-Type X)->
```

```
                                 <- Intermediate Result TLV (Success),
                                  Crypto-Binding TLV (Version=1
                                  EAP-FAST Version=1, Nonce,
                                  CompoundMAC),
                                  EAP Payload TLV (EAP-Request/Method Y)
```

    // Next EAP conversation started after successful completion
       of previous method X. The Intermediate-Result and Crypto-
       Binding TLVs are sent in this 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 (Version=1,
EAP-FAST Version=1, Nonce,
CompoundMAC),
EAP-Payload-TLV (EAP-Response/Method Y) ->
```

            // Optional additional Y Method exchanges...

```
                             <- EAP Payload TLV
                                (EAP-Request/Method Y)
```

```
EAP Payload TLV
(EAP-Response/Method Y) ->
```

```
                             <- Intermediate-Result-TLV (Success),
                                Crypto-Binding TLV (Version=1
                                EAP-FAST Version=1, Nonce,
                                CompoundMAC),
                                Result TLV (Success)
```

```
Intermediate-Result-TLV (Success),
Crypto-Binding TLV (Version=1,
EAP-FAST Version=1, Nonce,
CompoundMAC),
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
```

A.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-FAST
                               (S=1, A-ID)
   EAP-Response/EAP-FAST
   (TLS client_hello without
   PAC-Opaque extension)->
                               <- EAP-Request/EAP-FAST
                               (TLS Server Key Exchange,
                                TLS Server Hello Done)
   EAP-Response/EAP-FAST
   (TLS Client Key Exchange,
    TLS change_cipher_spec,
    TLS finished)->

                               <- EAP-Request/EAP-FAST
                               (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-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)) ->

                             <- Crypto-Binding TLV (Version=1,
                                EAP-FAST Version=1, Nonce,
                                CompoundMAC),
                                Result TLV (Success)

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

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

                             <- EAP-Failure
```

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

   Where EAP-FAST 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-FAST
                                (S=1, A-ID)

     EAP-Response/EAP-FAST
     (TLS client_hello)->
                                <- EAP-Request/EAP-FAST
                                (TLS server_hello,
                                 TLS certificate,
                                [TLS server_key_exchange,]
                                [TLS certificate_request,]
                                 TLS server_hello_done)
```

```
     EAP-Response/EAP-FAST
     ([TLS certificate,]
      TLS client_key_exchange,
     [TLS certificate_verify,]
      TLS change_cipher_spec,
      TLS finished) ->
                                <- EAP-Request/EAP-FAST
                                 (TLS change_cipher_spec,
                                  TLS finished,
                                  EAP-Payload-TLV
                                  (EAP-Request/Identity))

     // TLS channel established
        (Subsequent messages sent within the TLS channel,
                            encapsulated within EAP-FAST)

     // 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/Method X)

     EAP-Payload-TLV
     (EAP-Response/Method X) ->

                                 <- EAP-Payload-TLV
                                (EAP-Request/Method X)

     EAP-Payload-TLV
     (EAP-Response/Method X)->

                                <- Intermediate Result TLV (Success),
                                 Crypto-Binding TLV (Version=1
                                 EAP-FAST Version=1, Nonce,
                                 CompoundMAC),
                                 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 this 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 (Version=1,
     EAP-FAST Version=1, Nonce,
     CompoundMAC),
     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
```

Appendix B. Test Vectors

B.1. Key Derivation

    PAC KEY:

    0B 97 39 0F 37 51 78 09 81 1E FD 9C 6E 65 94 2B
    63 2C E9 53 89 38 08 BA 36 0B 03 7C D1 85 E4 14

    Server_hello Random

    3F FB 11 C4 6C BF A5 7A 54 40 DA E8 22 D3 11 D3
    F7 6D E4 1D D9 33 E5 93 70 97 EB A9 B3 66 F4 2A

    Client_hello Random

    00 00 00 02 6A 66 43 2A 8D 14 43 2C EC 58 2D 2F
    C7 9C 33 64 BA 04 AD 3A 52 54 D6 A5 79 AD 1E 00

    Master_secret = T-PRF(PAC-Key,
                    "PAC to master secret label hash",
                     server_random + Client_random,
                     48)

    4A 1A 51 2C 01 60 BC 02 3C CF BC 83 3F 03 BC 64
    88 C1 31 2F 0B A9 A2 77 16 A8 D8 E8 BD C9 D2 29
    38 4B 7A 85 BE 16 4D 27 33 D5 24 79 87 B1 C5 A2

    Key_block = PRF(Master_secret,
                  "key expansion",
                   server_random + Client_random)

    59 59 BE 8E 41 3A 77 74 8B B2 E5 D3 60 AC 4D 35
    DF FB C8 1E 9C 24 9C 8B 0E C3 1D 72 C8 84 9D 57
    48 51 2E 45 97 6C 88 70 BE 5F 01 D3 64 E7 4C BB
    11 24 E3 49 E2 3B CD EF 7A B3 05 39 5D 64 8A 44
    11 B6 69 88 34 2E 8E 29 D6 4B 7D 72 17 59 28 05
    AF F9 B7 FF 66 6D A1 96 8F 0B 5E 06 46 7A 44 84
    64 C1 C8 0C 96 44 09 98 FF 92 A8 B4 C6 42 28 71

    Session Key Seed

    D6 4B 7D 72 17 59 28 05 AF F9 B7 FF 66 6D A1 96
    8F 0B 5E 06 46 7A 44 84 64 C1 C8 0C 96 44 09 98
    FF 92 A8 B4 C6 42 28 71

```
IMCK = T-PRF(SKS,
             "Inner Methods Compound Keys",
             ISK,
             60)

     Note: ISK is 32 octets 0's.

16 15 3C 3F 21 55 EF D9 7F 34 AE C8 1A 4E 66 80
4C C3 76 F2 8A A9 6F 96 C2 54 5F 8C AB 65 02 E1
18 40 7B 56 BE EA A7 C5 76 5D 8F 0B C5 07 C6 B9
04 D0 69 56 72 8B 6B B8 15 EC 57 7B

[SIMCK 1]
16 15 3C 3F 21 55 EF D9 7F 34 AE C8 1A 4E 66 80
4C C3 76 F2 8A A9 6F 96 C2 54 5F 8C AB 65 02 E1
18 40 7B 56 BE EA A7 C5


MSK = T-PRF(S-IMCKn,
            "Session Key Generating Function",
            64);

4D 83 A9 BE 6F 8A 74 ED 6A 02 66 0A 63 4D 2C 33
C2 DA 60 15 C6 37 04 51 90 38 63 DA 54 3E 14 B9
27 99 18 1E 07 BF 0F 5A 5E 3C 32 93 80 8C 6C 49
67 ED 24 FE 45 40 A0 59 5E 37 C2 E9 D0 5D 0A E3


EMSK = T-PRF(S-IMCKn,
             "Extended Session Key Generating Function",
             64);

3A D4 AB DB 76 B2 7F 3B EA 32 2C 2B 74 F4 28 55
EF 2D BA 78 C9 57 2F 0D 06 CD 51 7C 20 93 98 A9
76 EA 70 21 D7 0E 25 54 97 ED B2 8A F6 ED FD 0A
2A E7 A1 58 90 10 50 44 B3 82 85 DB 06 14 D2 F9
```

B.2. Crypto-Binding MIC

    [Compound MAC Key 1]
    76 5D 8F 0B C5 07 C6 B9 04 D0 69 56 72 8B 6B B8
    15 EC 57 7B

    [Crypto-Binding TLV]
    80 0C 00 38 00 01 01 00 D8 6A 8C 68 3C 32 31 A8 56 63 B6 40 21 FE
    21 14 4E E7 54 20 79 2D 42 62 C9 BF 53 7F 54 FD AC 58 43 24 6E 30
    92 17 6D CF E6 E0 69 EB 33 61 6A CC 05 C5 5B B7

    [Server Nonce]
    D8 6A 8C 68 3C 32 31 A8 56 63 B6 40 21 FE 21 14
    4E E7 54 20 79 2D 42 62 C9 BF 53 7F 54 FD AC 58

    [Compound MAC]
    43 24 6E 30 92 17 6D CF E6 E0 69 EB 33 61 6A CC
    05 C5 5B B7

Authors' Addresses

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

   EMail: ncamwing@cisco.com

   David McGrew
   Cisco Systems
   San Jose, CA 95134
   US

   EMail: mcgrew@cisco.com

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

   EMail: jsalowey@cisco.com

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

   EMail: hzhou@cisco.com

Full Copyright Statement

Intellectual Property

Acknowledgement