Network Working Group
Internet-Draft
Expires: April 22, 2006

N. Cam-Winget
D. McGrew
J. Salowey
H. Zhou
Cisco Systems

October 19, 2005

The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)
draft-cam-winget-eap-fast-03.txt

Status of this Memo

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

The need to provide user friendly and easily deployable network access solutions has heightened the need for strong mutual authentication protocols that internally use weak user credentials. This document defines the base protocol which consists of establishing a Transport Layer Security (TLS) tunnel as defined in [RFC2246] and then exchanging data in the form of type, length, value objects (TLV) to perform further authentication. [I-D.cam-winget-eap-fast-provisioning] defines extensions to provision an additional credential called a protected access credential (PAC) to optimize the EAP-FAST exchange. In addition to regular TLS ciphersuites and handshakes, EAP-FAST supports using a PAC with the TLS extension defined in [I-D.salowey-tls-ticket] in order to support fast re-establishment of the secure tunnel without having to maintain per-session state on the server as described in Section 3.2.2.

EAP-FAST's design motivations included:

- 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.

- Immunity to passive dictionary attacks: as many authentication protocols require the password to be explicitly provided (either in the clear 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.

- 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.

- Flexibility to enable support for most password authentication interfaces: as many different password interfaces (e.g. MSCHAP, LDAP, OTP, etc) exist to authenticate a peer, the protocol must provide this support seamlessly.

- 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:

- 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 (already existent) 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.

- 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 part 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 [I-D.salowey-tls-ticket].

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.

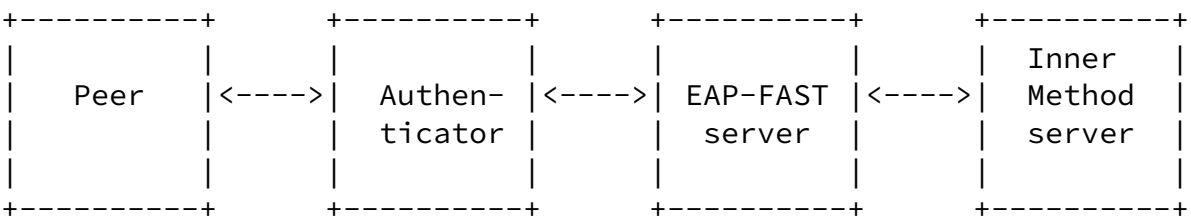
EAP-FAST makes use of the TLS enhancements in [I-D.salowey-tls-ticket] to enable an optimized TLS tunnel session resume while minimizing server state. In EAP-FAST the key and ticket used to establish the tunnel may be provisioned through mechanisms that do not involve the TLS handshake. It is RECOMMENDED that implementations support the capability to distribute the ticket and secret key within the EAP-FAST tunnel as specified in [I-D.cam-winget-eap-fast-provisioning]. The pre-shared secret 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 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 keys which can then communicated to the network access server (NAS).

2.1 Architectural Model

The network architectural model for EAP-FAST usage is shown below:



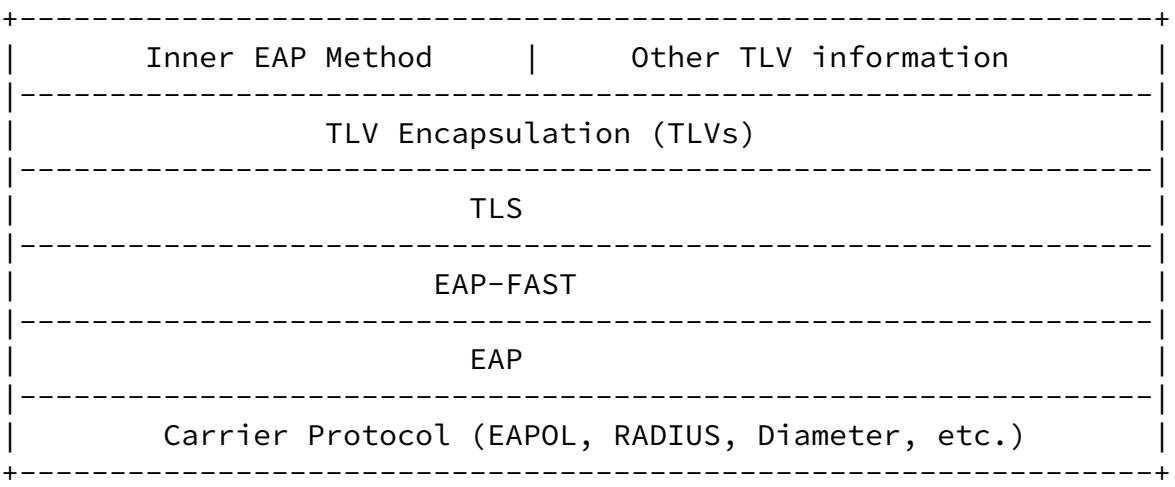
The entities depicted above are logical entities and may or may not correspond to separate network components. For example, the EAP-FAST server and Inner Method server might be a single entity; the authenticator and EAP-FAST server might be a single entity; or, the functions of the authenticator, EAP-FAST 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 7.4 provides an additional discussion of the implications of separating EAP-FAST server from the inner method server.

2.2 Protocol Layering Model

EAP-FAST packets are encapsulated within EAP, and 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 beneath it. The following diagram clarifies the relationship between protocols:



The TLV layer is a payload with standard 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 EAPOL may be used to transport EAP between the peer and the authenticator; RADIUS or Diameter are used to transport EAP between the authenticator and the EAP-FAST server.

3. EAP-FAST Protocol

EAP-FAST authentication occurs in two phases. For the first phase EAP-FAST employs the TLS handshake to invoke an authenticated key agreement exchange to establish a protected tunnel. Once the tunnel is established phase 2 begins in which the peer and server can engage

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 three 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 modification of EAP-FAST version and specifically downgrade of an EAP-FAST version negotiated, the peers MUST exchange information on the EAP-FAST version number received during version negotiation using the Crypto-Binding TLV described in Section 3.3.2. The receiver of the Crypto-Binding TLV MUST verify that the version in the Crypto-Binding TLV matches the version sent in the EAP-FAST version negotiation.

3.2 EAP-FAST Authentication Phase 1: Tunnel Establishment

EAP-FAST is based on 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 [I-D.cam-winget-eap-fast-provisioning]. 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 which 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 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 specified TLS ciphersuites specified in Section 3.2. EAP-FAST implementations MUST also support the immediate re-negotiation 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 authenticate 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 Session ID, master secret and ciphersuite. The peer attempts to resume a session by including a valid Session ID from a previous handshake in its ClientHello message. If the server finds a match for the Session ID and is willing to establish a new connection using the specified session state, the server will respond with the same session ID and proceed with the 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 [I-D.salowey-tls-ticket]. 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 [I-D.cam-winget-eap-fast-provisioning]. 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 a PAC-Key MUST be delivered in a secure channel and stored securely.

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 may include the PAC-Key and the PAC's peer identity. The PAC-Opaque

format and contents are specific to the PAC issuing server. The PAC-Opaque is a public credential and MAY be presented in the clear, so an attacker MUST NOT be able to gain useful information from the PAC-Opaque itself.

3. PAC-Info: this is a variable length field used to provide at minimum, the authority identity of PAC issuer. Other useful but not mandatory information, such as the PAC-Key lifetime, may also be conveyed by the EAP Server to the peer during PAC provisioning or refreshment.

The use of the PAC is based on the SessionTicket extension defined in [I-D.salowey-tls-ticket]. The EAP Server initiates the TLS conversation as in the previous section. 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 SessionTicket extension is valid and 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 [I-D.salowey-tls-ticket] until the handshake completes or a fatal error occurs. After the abbreviated handshake completes the peer and server are ready to enter 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.

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 Server's intent of negotiating full or abbreviated TLS handshake by checking the next TLS handshake messages in the server response to 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 fall back 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 this 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 formed of 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 which 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 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, initially with machine credentials followed by user credentials.

EAP method messages are carried within EAP-Payload TLVs defined in Section 4.2.6. Upon method 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.

If both peer and server indicate success then the method is considered to have completed. If either indicates failure then the method is considered to have failed. The result of failure of a 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 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 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 3.4.2.

3.4 Error Handling

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

1. Errors in TLS layer are communicated via TLS alert messages in all phases of EAP-FAST.
2. The Intermediate-Result TLVs indicate 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.

3.4.1 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.

3.4.2 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 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.

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.

3.5 Fragmentation

A single TLS record may be up to 16384 octets in length, but a TLS message may span multiple TLS records, and a TLS certificate message may in principle be as long as 16MB. 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 as is provided in IPv4.

EAP-FAST fragmentation support is provided through addition of flag bits within the EAP-Response and EAP-Request packets, as well as a TLS Message Length field of four octets. Flags include the Length included (L), More fragments (M), and 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.

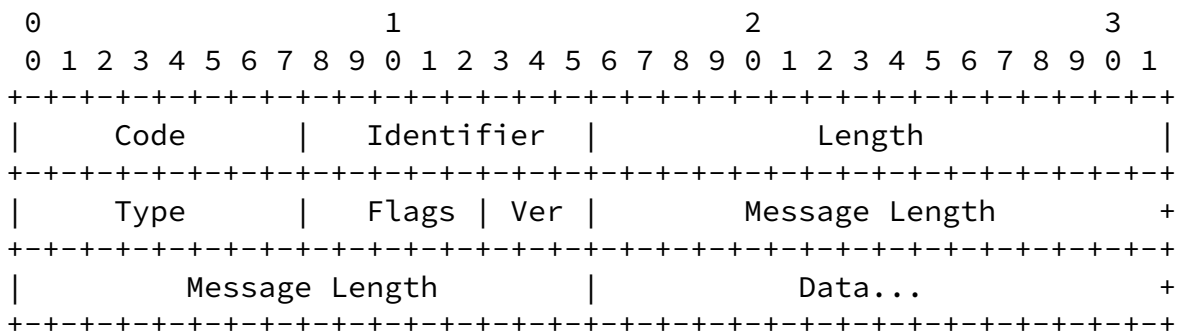
Similarly, when the EAP-FAST server receives an EAP-Response with the M bit set, it must respond with an EAP-Request with EAP-Type of EAP-FAST and no data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors 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.



Code

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.

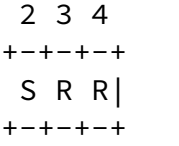
Length

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



L Length included
M More fragments
S EAP-FAST start
R Reserved (must be zero)

L bit (length included) is set to indicate the presence of the four octet Message Length field, and MUST be set for the

first fragment of a fragmented TLS message or set of messages. The M bit (more fragments) is set on all but the last fragment. The S bit (EAP-FAST Start) is set in an EAP-FAST Start message.

Ver

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

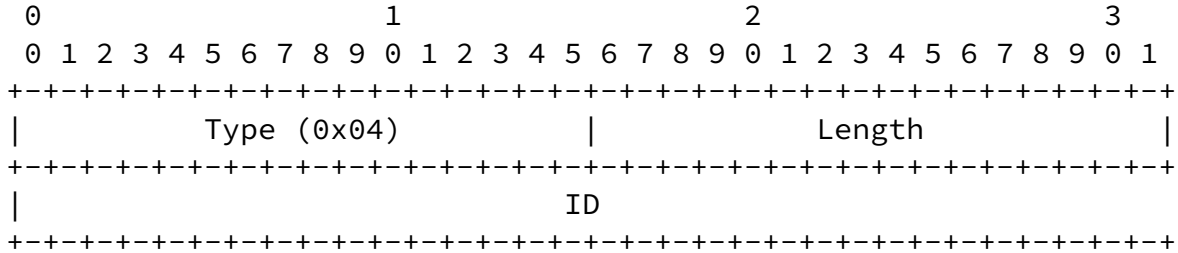
Message Length

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 a 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 to 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



Type

0x04 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 which 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.

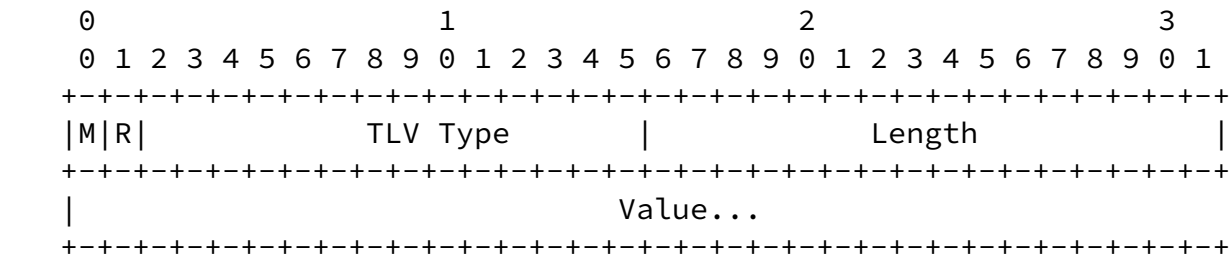
Note that a peer or server may support a TLV with the mandatory bit set, but may not understand the contents. The appropriate response to a supported TLV with content that is not understood is defined by the individual TLV specification.

EAP implementations compliant with this specification MUST support TLV exchanges, as well as 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.



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

4 NAK TLV
5 Error TLV
7 Vendor-Specific TLV
9 EAP-Payload TLV
10 Intermediate-Result TLV
11 PAC TLV [I-D.cam-winget-eap-fast-provisioning]
12 Crypto-Binding TLV
18 Server-Trusted-Root TLV [I-D.cam-winget-eap-fast-provisioning]
19 Request-Action TLV
20 PKCS#7 TLV [I-D.cam-winget-eap-fast-provisioning]

Length

The length of the Value field in octets.

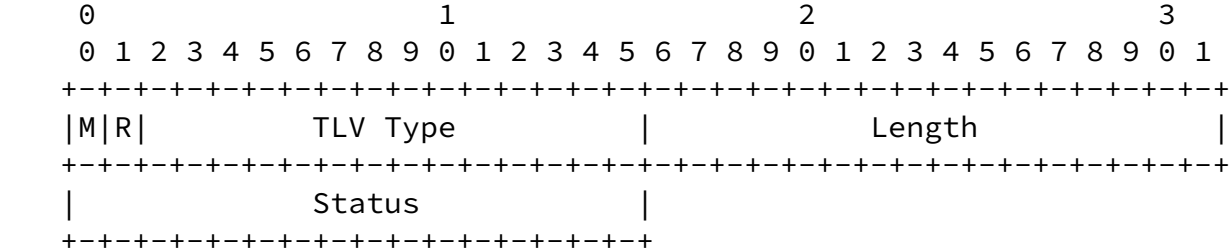
Value

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 3.4.2. An Result TLV indicating failure MUST NOT be accompanied by the following TLVs: NAK, EAP-Payload TLV, or Crypto-

Binding TLV. Result TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

3 for Result TLV

Length

2

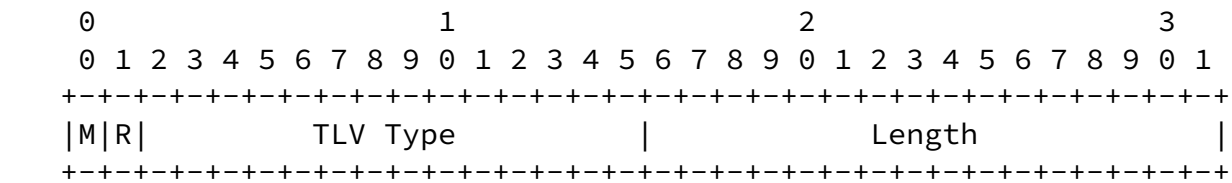
Status

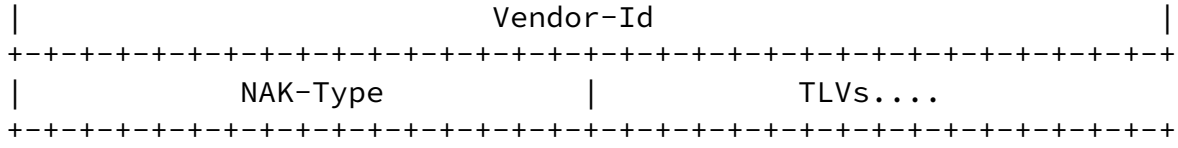
The Status field is two octets. Values include:

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:





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 3 octets are the 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.

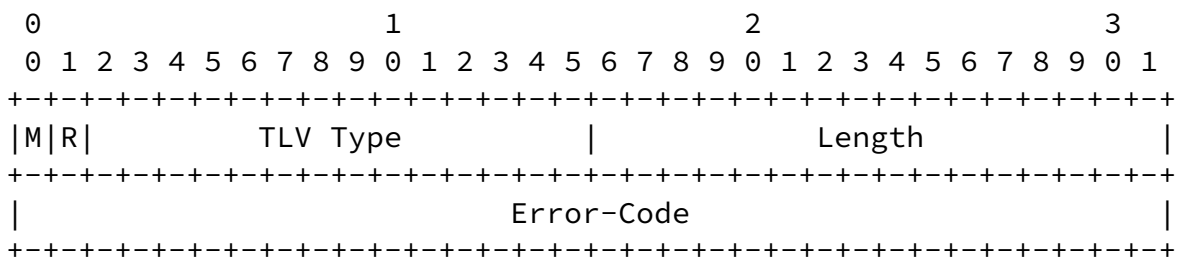
TLVs

This field contains a list of 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 3.4.2. The Error TLV is defined as follows:



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:

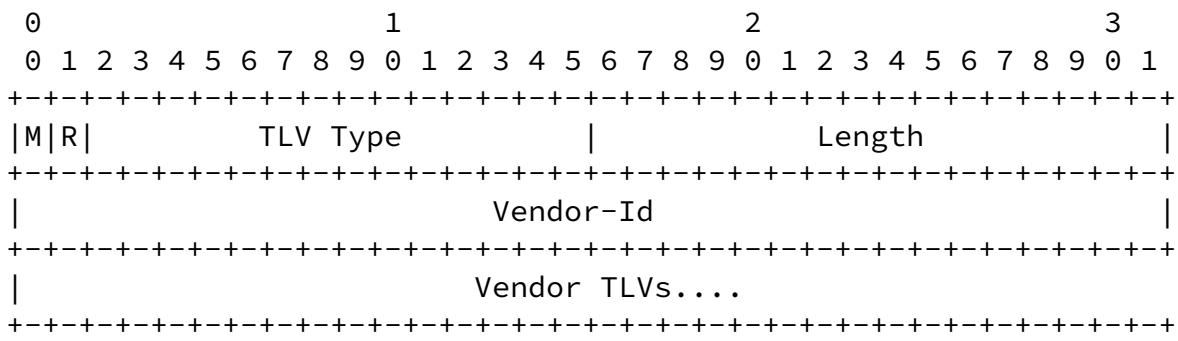
2001 Tunnel_Compromise_Error
2002 Unexpected_TLVs_Exchanged

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. The 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:



M

0 or 1

R

Reserved, set to zero (0)

TLV Type

7 for Vendor Specific TLV

Length

>=4

Vendor-Id

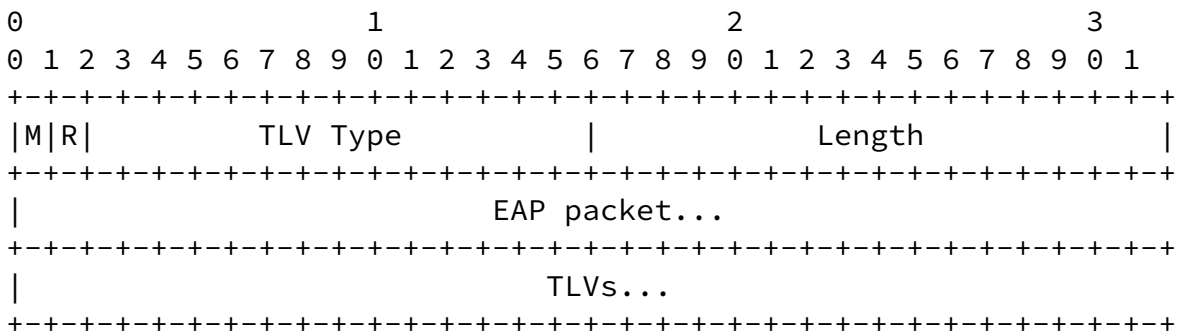
The Vendor-Id field is four octets, and contains the Vendor-Id of the TLV. The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order.

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 EAP request and 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:



M

Mandatory, set to (1)

R

Reserved, set to zero (0)

TLV Type

9 for EAP-Payload TLV

Length

>=0

EAP packet

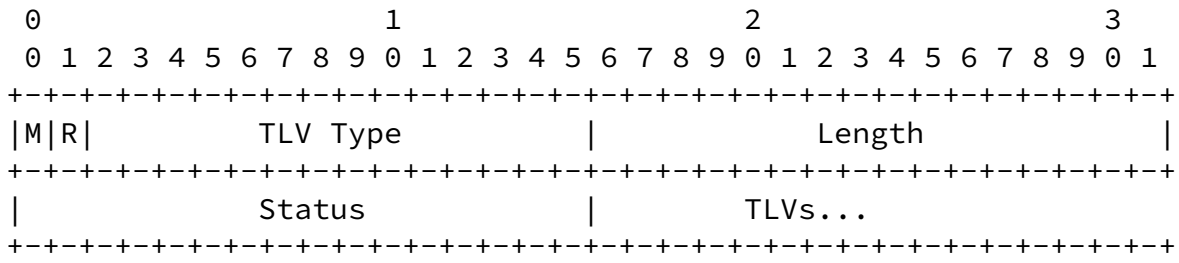
This field contains a complete EAP packet, including the EAP header (Code, Identifier, Length, Type) fields. The length of this field is determined by the Length field of the encapsulated EAP packet.

TLVs

This (optional) field contains a list of TLVs associated with the EAP packet field. The TLVs MUST NOT have the mandatory bit set. The total length of this field is equal to the Length field of the EAP-Payload TLV, minus the Length field in the EAP header of the EAP packet field.

4.2.7 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:



M

Mandatory, set to (1)

R

Reserved, set to zero (0)

TLV Type

10 for Intermediate-Result TLV

Length

>=2

Status

The Status field is two octets. Values include:

1 Success
2 Failure

TLVs

This (optional) field is of indeterminate length, and contains 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.

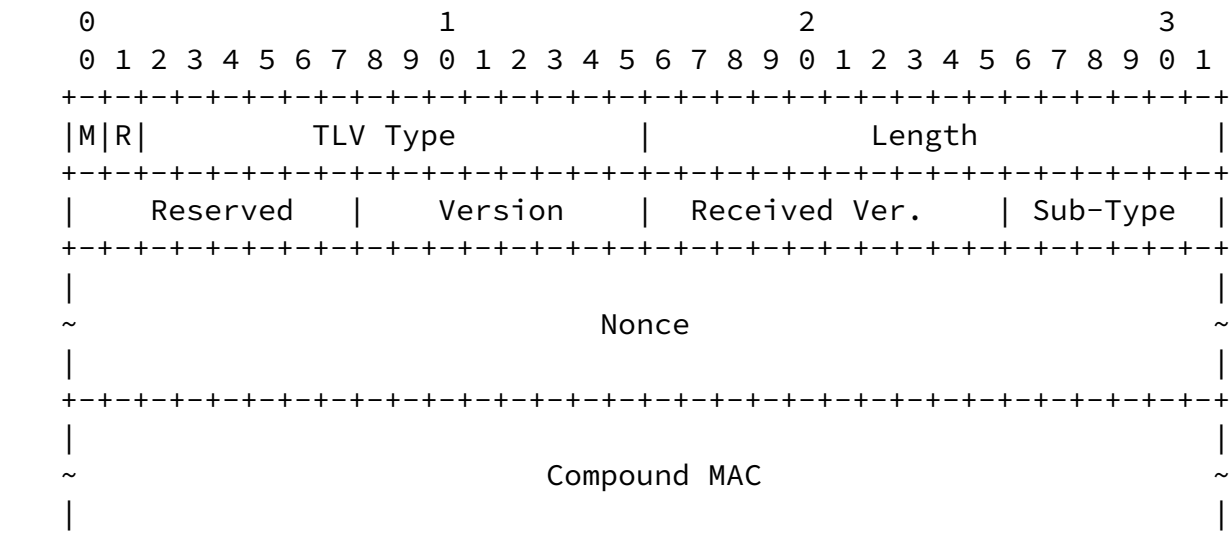
The Crypto-Binding TLV MUST be included with Intermediate-Result TLV to perform Cryptographic Binding after each successful EAP method in a sequence of EAP methods. The Crypto-Binding TLV can be issued at other times as well.

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

- The Crypto-Binding TLV version is supported
- The MAC verifies correctly
- The received version in the Crypto-Binding TLV matches the version sent by the receiver during the EAP version negotiation
- 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 3.4.2

The Crypto-Binding TLV is defined as follows:



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 implementation compliant with this version of EAP-FAST, the version number MUST 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 two octets. Defined values are

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.

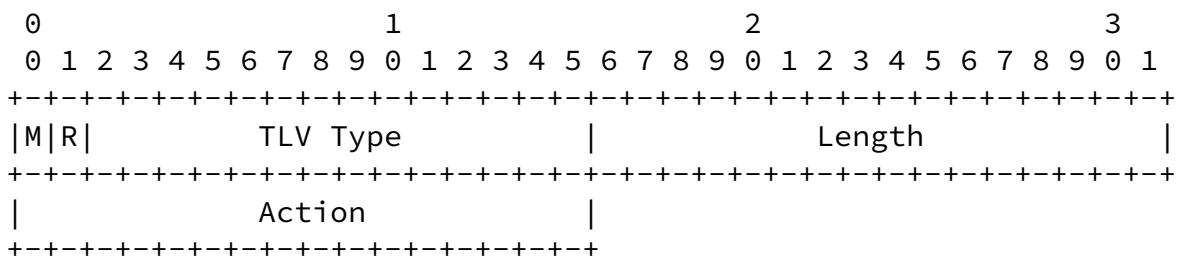
Compound MAC

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:



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:

1 Process-TLV
2 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 EAP-FAST Authentication tunnel, the following construction from [RFC2246] 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 [RFC2246].

For example, if the EAP-FAST Authentication employs 128bit RC4 and SHA1, the key_block is 112 bytes 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 [RFC2246].

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 bytes if it is longer than 32 bytes or padded to a length of 32 bytes with zeros if it is less than 32 bytes. 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-IMSK is as follow:

```
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 all communications in EAP-FAST.

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 session 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 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. outputlength 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 octet length of key material, the T-PRF is calculated as follows:

```
S = label + 0x00 + seed
T-PRF output = T1 + T2 + T3 + T4 + ... + Tn
T1 = HMAC-SHA1 (key, S + outputlength + 0x01)
T2 = HMAC-SHA1 (key, T1 + S + outputlength + 0x02)
T3 = HMAC-SHA1 (key, T2 + S + outputlength + 0x03)
T4 = HMAC-SHA1 (key, T3 + S + outputlength + 0x04)
```

Where '+' indicates concatenation and "\0" is a NULL character. 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 protocol, in accordance with BCP 26, [RFC2434].

There is a name space in EAP-FAST that requires registration: TLV Types. All these numbers may be assigned by Specification Required as defined in BCP 26.

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 that 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 that defined by TLS [RFC 2246] and provides the same security strengths afforded by TLS employing a strong entropy shared master secret.

When EAP-FAST is invoked for enabling network access, mutual authentication is first achieved by proof of a mutually shared unique PAC-Key during the tunnel establishment and optional inner method authentication. Further, the Crypto-Binding TLV is enforced to be run after any EAP method that supports (mutual) authentication ensuring that it was the same peer and EAP server that communicated in all transpired methods (including tunnel establishment).

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 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). Since a subsequent protected EAP conversation can take place within the TLS session, selection of EAP-FAST as an authentication method does not limit the potential secondary authentication methods. As a result, the only legitimate reason for a peer to NAK EAP-FAST as an authentication method is that it does not support it. Where the additional security of EAP-FAST is required, the server shall best determine how to respond to a NAK as this is beyond the scope of this specification.

Inner method negotiation is protected by the mutually authenticated TLS tunnel established in EAP-FAST and immune to attacks. An attacker cannot readily determine the EAP method used, except perhaps by traffic analysis.

7.3 Separation of the EAP Server and the Authenticator

When EAP-FAST is successfully invoked to gain network access, the EAP endpoints will mutually authenticate, and derive a session key for subsequent use in link layer security. Since it is presumed that the peer and EAP client reside on the same machine, it is necessary for the EAP client module to pass the session key to the link layer encryption module.

As EAP-FAST is defined to achieve mutual authentication between a peer and AS, it will not achieve direct authentication to the Authenticator (which is true for most if not all currently specified EAP methods).

It is implied that there is an established trust between Authenticator and AS before the AS securely distributes the session keys to the authenticator. Using the transitive property and the authenticator to AS trust assumption, if the AS trusts the authenticator and distributes the session key to the authenticator, and the peer has successfully gained authorization by mutually deriving fresh session keys, the peer may then presume trust with the authenticator who can prove it has those session keys. Note however, that this presumed trust does not authenticate the authenticator to the peer, it merely proves that the AS has a trust relationship with said authenticator. Further, it is presumed that a secure mechanism is used by the AS to distribute the session key to the authenticator.

In the case of the AS and the home AAA server logical model, similar security properties hold as that between the AS and authenticator. Though in general, it is highly recommended that the AAA server be reside on the same host as the AS. In both cases, the presumed trust between authenticator and AS as well as AS and AAA server as well as the security in the transport (such as IPsec) and key delivery (such as NIST approved key wrapping) mechanisms for these links are outside the scope of the EAP-FAST specification. Without these presumed trusts and secure transport mechanisms, security vulnerabilities will exist.

7.4 Separation of Phase 1 and Phase 2 Servers

Separation of the EAP-FAST Phase 1 from the Phase 2 conversation is not recommended. Without a trust relationship and proper protection (such as IPsec) for RADIUS, by allowing a the Phase 1 conversation to be terminated at a different (proxy) AS (AS1) than the Phase 2 conversation (terminated at AS2), vulnerabilities are introduced since clear text transmission between AS1 and AS2 ensue. Some vulnerabilities include:

- Loss of identity protection

- Offline dictionary attacks
- Lack of policy enforcement

In order to find the proper EAP-FAST destination, the peer SHOULD place a Network Access Identifier (NAI) conforming to [RFC2486] in the clear-text Identity Response.

There may be cases where a natural trust relationship exists between the (foreign) authentication server and final EAP server, such as on a campus or between two offices within the same company, where there is no danger in revealing the identity of the station to the authentication server. In these cases, using a proxy solution without end to end protection of EAP-FAST MAY be used. The EAP-FAST encrypting/decrypting gateway SHOULD provide support for IPsec protection of RADIUS in order to provide confidentiality for the portion of the conversation between the gateway and the EAP server, as described in [RFC3162].

7.5 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:

- Per packet confidentiality and integrity protection
- User identity protection
- Better support for notification messages
- Protected EAP inner method negotiation
- Sequencing of EAP methods
- Strong mutually derived master session keys
- Acknowledged success/failure indication
- Faster re-authentications through session resumption
- Mitigation of dictionary attacks
- Mitigation of man-in-the-middle attacks
- Mitigation of some denial of service attacks

It should be noted that EAP-FAST as in many other authentication protocols, a denial of service attack can be easily mounted by adversaries imposing as either peer or AS and failing to present the proper credential. This is an inherent problem in most authentication or key agreement protocols and is so noted for EAP-FAST as well.

EAP-FAST protection addresses a number of weaknesses present in LEAP, PEAPv1, EAP-TTLS and the inner EAP methods used in the EAP- FAST Authentication Phase 2 conversation. These weaknesses have been described in draft-puthenkulam-eap-binding-03.txt.

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 respectively. The keys derived for the protection relies on strong random challenges provided by both peer and AS as well as a shared secret with strong entropy (minimally 32 octets). It is recommended that peers provide strong random number generators that can satisfy the criteria as that described by NIST Special Publication 800-22b (e.g. NIST SP800-22b). The AS acting as the PAC distributor must generate unique and randomly generated 32 octet keys for each peer.

7.5.1 User Identity Protection and Verification

As EAP-FAST employs TLS to establish a secure tunnel, the initial Identity request/response may be omitted as it must be transmitted in the clear and thus subject to snooping and forgery. It may be omitted also in deployments where it is known that all users are required to authenticate with EAP-FAST. Alternately, an anonymous identity may be used in the Identity response.

If the initial Identity request/response has been tampered with, the AS may be unable to verify the peer's identity. For example, the peer's user name may not be valid or may not be within a realm handled by the AS. Rather than attempting to proxy the authentication to the server within the correct realm, the AS should terminate the conversation.

The EAP-FAST peer can present the server with multiple identities. This includes the claim of identity within the initial EAP- Response/ Identity (MyID) packet, which is typically used to route the EAP conversation to the appropriate home back end AS. There may also be subsequent EAP-Response/Identity packets sent by the peer once the secure tunnel has been established.

The PAC-Opaque field conveyed by the peer to the AS contains the peer's identity that should be validated with at least one identity presented in the EAP-FAST Authentication Phase 2 conversation. This ensures that the PAC-Key is employed by the intended peer. Though EAP-FAST implementations should not attempt to compare the EAP-FAST Authentication Phase 1 Identity disclosed in the EAP Identity response packet with those Identities claimed in Phase 2; the AS should match the identity disclosed in the PAC-Opaque field with at

least one identity disclosed in EAP-FAST Authentication Phase 2.

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. Alternatively, if identity protection is required, then it is possible to perform certificate authentication using an EAP method (for example: EAP-TLS) within the TLS session in EAP-FAST Phase 2.

To accomplish TLS restart, 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.}

7.5.2 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. To that extent, by establishing a mutually authenticated protected tunnel, EAP-FAST mitigates dictionary attacks by protecting the weak credential based authentication method. The protection is based on strong cryptographic algorithms such as RC4 and HMAC-SHA1 to provide message confidentiality and integrity respectively. The keys derived for the protection relies on strong random challenges provided by both peer and AS as well as a strong entropy (minimally 32 octet) shared secret. The AS acting as the PAC distributor MUST generate unique and randomly generated 32 octet keys for each peer.

7.5.3 Protection against MitM Attacks

The recommended solution is to always deploy authentication methods with protection of EAP-FAST. If a deployment chooses to allow an EAP method protected by EAP-FAST without protection of EAP-FAST at the same time, then this opens up a possibility of a Compound Authentication Binding man-in-the-middle attack [MITM].

A man-in-the-middle can spoof the client to authenticate to it instead of the real EAP server; and forward the authentication to the real server over a protected tunnel. Since the attacker has access

to the keys derived from the tunnel, it can gain access to the network. EAP-FAST prevents this attack in two ways:

1. An adversary must have the corresponding peer's PAC-Key to mutually authenticate during EAP-FAST Authentication Phase 1 establishment of a secure tunnel; and
2. By using the keys generated by the inner authentication method in the crypto-binding exchange described in above protected termination section 6.5.

Both compound MAC and compound session key approaches are used to prevent the aforementioned man-in-the-middle attack. Both the peer and the EAP server MUST derive compound MAC and compound session keys using the procedure described in Section 6.7. As a strong PAC-Key is used to establish mutual authentication in EAP-FAST Phase 1, this attack is also prevented if the inner authentication method does not generate keys. Thus, most EAP authentication methods are protected from these MitM attacks when protected by EAP-FAST.

To summarize, EAP-FAST Authentication mitigates most MitM attacks in the following ways:

1. Identity binding with PAC-Key: in presenting the PAC-Opaque field to the AS, a peer is presenting an authenticated credential. With the user identity serving as another validation point for the inner EAP authentication method, a MitM may not interject and impersonate itself as the peer unless it has recovered the PAC-Key as well as the PAC-Opaque field. Thus, the PAC-Key binding to an Identity prevents an adversary from interjection regardless of whether the authentication method generates session keys.
2. Cryptographic binding of EAP-FAST Phase 1 and all methods within Phase 2: by cryptographically binding key material generated in

all methods, peer and AS are assured that they were the sole participants of all transpired methods.

7.5.4 PAC Validation with User Credentials

The PAC-Opaque field is consumed by the AS during a network access EAP-FAST invocation to both acquire and validate the authenticity of the PAC credential. However, during the EAP-FAST Phase 1 conversation it validates the peer based on the secret, PAC-Key and not on the identity. Further, since the EAP-FAST Phase 1 conversation occurs in clear text, it is feasible for an adversary to acquire a PAC-Opaque credential.

While a PAC-Opaque credential can be easily acquired, the shared secret, PAC-Key is not discernible from the PAC-Opaque field. Thus, an adversary must resort to a brute force attack to gain the PAC- Key

from PAC-Opaque information.

Another feasible scenario due to the clear text transmission is the spoofing of the PAC-Opaque field. While the PAC-Opaque is authenticated to mitigate forgery, a denial of service and potential user lockout (based on deployment configurations that may choose to lock a peer after a configurable number of failed attempts) is feasible.

The final validation and binding of the PAC credential is the identity validation in the EAP-FAST Phase 2 conversation. A compliant implementation of EAP-FAST MUST match the identity presented to the AS in the PAC-Opaque field with 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. This validation helps mitigate the MitM attack as described in Section 12.5.3.

7.6 Protecting against Forged Clear Text EAP Packets

As described earlier, EAP Success and EAP Failure packets are in general sent in clear text and may be forged by an attacker without fear of detection. Forged EAP Failure packets can be used to

convince an EAP peer to disconnect. Forged EAP Success packets may be used by any rogue to convince a peer to let itself access the network, 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 6.5.

The success/failure decisions sent by a protected mechanism 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 message; 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 RFC 3748, the AS must send a clear text EAP Success or

EAP Failure packet to terminate the EAP conversation, so that no response is possible. However, since EAP Success and EAP Failure packets are not retransmitted, if the final packet is lost, then authentication will fail. As a result, where packet loss is expected to be non-negligible, unacknowledged success/failure indications lack robustness.

While an EAP-FAST protected EAP Success or EAP Failure packet should not be a final packet in an EAP-FAST conversation, it may be feasible based on the conditions stated above and construed as an optimization savings of a full round-trip in low packet loss environments.

7.7 Implementation

Both server and in particular, client implementations must provide a suitably strong PRNG to ensure good entropy challenges. Suitable recommendations for PRNGs can be found in PKCS#5, PKCS#11 and criteria for suitable PRNGS are also defined by NIST Special

Publication 800-22b.

7.8 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 remoted, the EAP server will not reside on the same machine as the authenticator, and therefore the name in the EAP server's certificate cannot be expected to match that of the intended destination. In this case, a more appropriate test might be whether the EAP server's certificate is signed by a CA controlling the intended destination and whether the EAP server exists within a target sub-domain.

7.9 Security Claims

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

Auth. mechanism:	Tunneled authentication as well as pre-shared key.
Ciphersuite negotiation:	Yes
Mutual authentication:	Yes
Integrity protection:	Yes,Only EAP Type Data field and inner EAP methods contained in this field are protected.

Replay protection:	Yes
Confidentiality:	Yes
Key derivation:	Yes
Key strength:	TLS key strength, may be enhanced by binding keys with inner methods
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.

8. Acknowledgements

The EAP-FAST design and protocol specification is based on the ideas and hard efforts of Pad Jakkahalli, Mark Krischer, Doug Smith, Ilan Frenkel, Glen Zorn and Jeremy Steiglitz of Cisco Systems, Inc.

The TLV processing was inspired from work on PEAPv2 with Ashwin Palekar, Dan Smith and Simon Josefsson. Helpful review comments were provided by Russ Housley and Jari Arkko.

9. References

9.1 Normative References

[I-D.cam-winget-eap-fast-provisioning]
Cam-Winget, N., "Dynamic Provisioning using EAP-FAST", draft-cam-winget-eap-fast-provisioning-01 (work in progress), July 2005.

[I-D.salowey-tls-ticket]
Salowey, J., "Transport Layer Security Session Resumption without Server-Side State", draft-salowey-tls-ticket-04 (work in progress), September 2005.

[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.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J.,

and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

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

9.2 Informative References

[RFC2486] Aboba, B. and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

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

[RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

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

Appendix A. Examples

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-Type=EAP-FAST, V=1
                            (EAP-FAST Start, S bit set, A-ID)

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

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

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

       TLS channel established
       (messages sent within the TLS channel)

                               <- 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-Type=EAP-FAST, V=1
                             (EAP-FAST Start, S bit set, A-ID)

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

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

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

     TLS channel established
     (messages sent within the TLS channel)

                            <- 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 Cipher Suite

In the case where an abbreviated TLS handshake is tried and failed and falls back to certificate based full TLS handshake occurs within 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-Type=EAP-FAST, V=1
                           (EAP-FAST Start, S bit set, A-ID)
   EAP-Response/
   EAP-Type=EAP-FAST, V=1
   (TLS client_hello
   [PAC-Opaque extension])->

   // Peer sends PAC-Opaque of Tunnel PAC along with a list of
      ciphersuites supported. If Server rejects the PAC-
```

```
   Opaque, if falls through to the full TLS handshake

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



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

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

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

   // identity protected by TLS.

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

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

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

```
                              <- 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-Type=EAP-FAST, V=1
                              (EAP-FAST Start, S bit set, A-ID)
      EAP-Response/
      EAP-Type=EAP-FAST, V=1
      (TLS client_hello)->
                              <- EAP-Request/
                              EAP-Type=EAP-FAST, V=1
                              (TLS server_hello,
                               TLS certificate,
```

```
                            [TLS server_key_exchange,]
                            [TLS certificate_request,]
                             TLS server_hello_done)
      EAP-Response/
      EAP-Type=EAP-FAST, V=1
      (TLS client_key_exchange,
       TLS change_cipher_spec,
       TLS finished) ->
                            <- EAP-Request/
                            EAP-Type=EAP-FAST, V=1
                            (TLS change_cipher_spec,
                             TLS finished,TLS Hello-Request)

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

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

      TLS client_hello ->


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

                            <- TLS change_cipher_spec,
                               TLS finished,
                               Result TLV (Success)
```



```
      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-Type=EAP-FAST, V=1
                           (EAP-FAST Start, S bit set, A-ID)

   EAP-Response/
   EAP-Type=EAP-FAST, V=1
   (TLS client_hello)->
                           <- EAP-Request/
                           EAP-Type=EAP-FAST, V=1
                           (TLS server_hello,
                            TLS certificate,
                           [TLS server_key_exchange,]
                           [TLS certificate_request,]
                            TLS server_hello_done)
                           (Fragment 1: L, M bits set)

   EAP-Response/
   EAP-Type=EAP-FAST, V=1 ->

                           <- EAP-Request/
                              EAP-Type=EAP-FAST, V=1
                           (Fragment 2: M bit set)
   EAP-Response/
   EAP-Type=EAP-FAST, V=1 ->
                           <- EAP-Request/
                           EAP-Type=EAP-FAST, V=1
                           (Fragment 3)
   EAP-Response/
```

```
EAP-Type=EAP-FAST, V=1
([TLS certificate,]
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished)
 (Fragment 1: L, M bits set)->

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

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

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

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

// identity protected by TLS.

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

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

// Method X exchanges followed by Protected Termination

                          <- 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-Type=EAP-FAST, V=1
                           (EAP-FAST Start, S bit set, A-ID)

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

```
                              TLS finished,
                             EAP-Payload-TLV[
                             EAP-Request/Identity])

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



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

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

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

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

             // Optional additional X Method exchanges...

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

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

                              <- Intermediate Result TLV (Success),
                               Crypto-Binding TLV (Version=1
                               EAP-FAST Version=1, Nonce,
                               CompoundMAC),
                               EAP Payload TLV [EAP-Type=Y],

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

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

```
      Intermediate Result TLV (Success),
      Crypto-Binding TLV (Version=1,
      EAP-FAST Version=1, Nonce,
      CompoundMAC),
      EAP-Payload-TLV [EAP-Type=Y] ->

             // Optional additional Y Method exchanges...

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

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



```
                              <- Intermediate-Result-TLV (Success),
                                Crypto-Binding TLV (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-Type=EAP-FAST, V=1
                           (EAP-FAST Start, S bit set, A-ID)

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



```
    TLS change_cipher_spec,
    TLS finished)

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

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

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

   EAP-Payload TLV/
   EAP Identity Response ->

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

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

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

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

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

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

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

                            <- EAP-Failure
```


A.8 Stateless Session Resume Using Authorization PAC

The following exchanges show a successful server stateless EAP-FAST session resume using Tunnel PAC with User Authorization PAC. The conversation will appear as follows:

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

```
                          EAP-Type=EAP-FAST, V=1
                          (EAP-FAST Start, S bit set, A-ID)

   EAP-Response/
   EAP-Type=EAP-FAST, V=1
   (TLS client_hello with
   PAC-Opaque extension)->
                          <- EAP-Request/
                          EAP-Type=EAP-FAST, V=1
                          (TLS server_hello,
                          (TLS change_cipher_spec,
                           TLS finished)
   EAP-Response/
   EAP-Type=EAP-FAST, V=1
   (TLS change_cipher_spec,
    TLS finished)
   (PAC-TLV with User Authorization
    PAC) ->

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

      // User Authorization PAC is piggybacked to the TLS Finished as
         Application Data and protected by the TLS tunnel

                          <- Result TLV (Success)

      // User Authorization PAC is valid and inner methods are bypassed


   Result TLV (Success) ->

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



```
                          <- EAP-Success
```

A.9 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-Type=EAP-FAST, V=1
                           (EAP-FAST Start, S bit set, A-ID)

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

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

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

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

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

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

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

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

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

                        <- Intermediate Result TLV (Success),
                         Crypto-Binding TLV (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 next packet to minimize round-trips.

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

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

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
```

Intellectual Property Statement

ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

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