Network Working Group
Internet-Draft
Category: Informational
Expires: April 25, 2005

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

October 25, 2004

EAP Flexible Authentication via Secure Tunneling (EAP-FAST)
draft-cam-winget-eap-fast-01.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 client and a server by using the Transport Layer Security (TLS) to establish a mutually authenticated tunnel. However, unlike current existing tunneled authentication protocols, EAP-FAST also enables the establishment of a mutually authenticated tunnel by means of symmetric cryptography. Furthermore, within the

secure tunnel, EAP encapsulated methods can ensue to either facilitate further provision of credentials, authentication or authorization policies by the server to the client.

Table of Contents

1. Introduction

The need to provide user friendly and easily deployable network access solutions has heightened the need to enable strong mutual authentication protocols that internally use weak user credentials. While several such authentication protocols [PEAP] [EAP-TTLS] exist

today, they are encumbered by the use of asymmetric cryptographic operations that often render such protocols prohibitive on very low end peer devices.

Like [TLS-PSK], EAP-FAST employs symmetric cryptography to allay the PKI requirements of [PEAP] or [EAP-TTLS]. Additionally, EAP-FAST employs the TLS client_hello extension [RFC3546] as a further optimization to minimize the state maintained by the server. EAP-FAST's design motivations included:

* Mutual Authentication: an AS must verify the identity and authenticity of the peer, and the peer must verify the authenticity of an AS.

* 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 AS; 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 the conversation between peer and AS.

* 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. This capability is similar to [PEAP].

* Minimize the authentication server's per user authentication state requirements: with large deployments, it is typical to have many servers acting as the AS 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

Some of the following terms are taken from RFC 2284bis:

EAP Server

The entity that terminates the EAP authentication with the peer. In the case where there is no backend authentication server, this term refers to the authenticator. Where the authenticator operates in pass-through, it refers to the backend authentication server.

Authenticator

The end of the link initiating EAP authentication. The term Authenticator is used in [IEEE-802.1X], and authenticator has

the same meaning in this document.

Peer

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

Supplicant

The end of the link that responds to the authenticator in [IEEE-802.1X]. In this document, this end of the link is called the peer.

Backend Authentication Server

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

Master Session Key (MSK)

Keying material exported by an EAP method.

Man in the Middle (MitM)
An adversary that can successfully inject itself between a peer and EAP server. The MitM succeeds by impersonating itself as a valid peer, authenticator or authentication server.

Message Authentication Code (MAC)
A MAC is a function that takes a variable length input and a key to produce a fixed-length output to carry authentication and integrity protection of data.

Message Integrity Check (MIC)
A keyed hash function used for authentication and integrity protection of data. This is usually called a Message Authentication Code (MAC), but IEEE 802 specifications (and this document) use the acronym MIC to avoid confusion with Medium Access Control.

Protected Access Credential (PAC)
Credentials distributed to users for future optimized network authentication, which always consists of a secret part and an opaque part. The secret part is secret key material that can be used in future transactions. The opaque part is presented when the client wishes to obtain access to network resources. It aids the server in validating that the client possesses the secret part.

Silently Discard
This means the implementation discards the packet without further processing. The implementation SHOULD provide the capability of logging the event, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

Successful Authentication
In the context of this document, "successful authentication" is an exchange of EAP messages, as a result of which the authenticator decides to allow access by the peer, and the peer decides to use this access. The authenticator's decision typically involves both authentication and authorization aspects; the peer may successfully authenticate to the authenticator but access may be denied by the authenticator due to policy reasons.

2. Protocol Overview

EAP-FAST is an extensible framework that enables mutual authentication by using a pre-shared secret to establish a protected tunnel. Like [PEAP], the protocol is based on TLS; however, enhancements are made to TLS to enable EAP-FAST to initiate the tunnel establishment exchange using symmetric cryptography while minimizing server state. The tunnel is then used to protect weaker authentication methods, typically based on passwords.

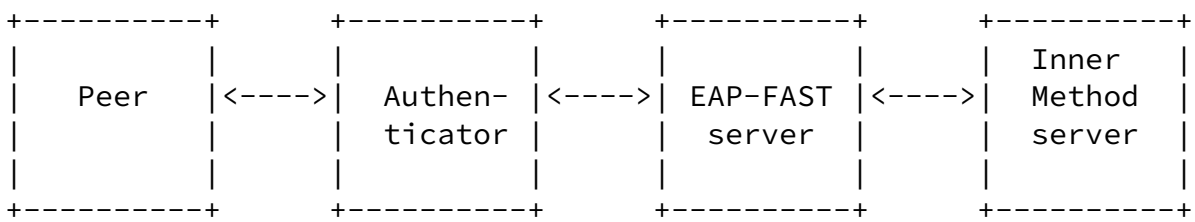
The pre-shared secret used in EAP-FAST is referred to as the Protected Access Credential key (or PAC-Key); the PAC-Key is used to mutually authenticate the client and server when securing a tunnel. Furthermore, the PAC-Key is refreshed and managed as part of the EAP-FAST protocol. EAP-FAST allays server state by the use of a PAC-Opaque, which contains the PAC-Key encrypted by a strong key only known to the server and sent to the server with the TLS ClientHello. With the use of PAC-Opaque, EAP-FAST alleviates the server's need to store per user PAC and state.

The EAP-FAST conversation is used to establish or resume an existing session to typically establish network connectivity between a peer and the network. A peer and AS achieve mutual

authentication by invoking a symmetric authenticated key agreement to protect the communications that further authenticates and authorizes the client to use the network. A successful result is a mutual derivation of strong session keys which can then be provisioned (by the AS) to the network access server (NAS, typically in 802.11 these are the access points or 802.1X authenticators).

3. Architectural Model

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

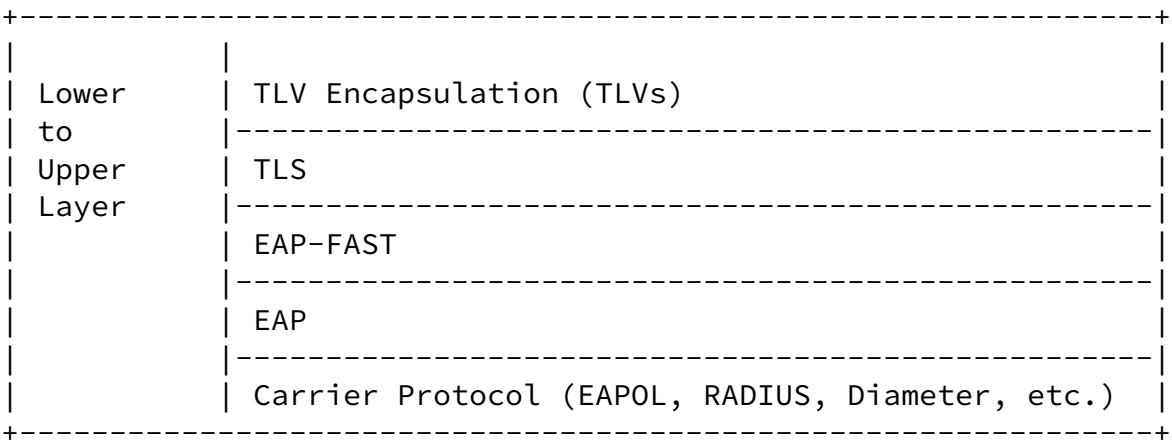


The entities depicted above are logical entities and may or may not correspond to separate network components. For example, the EAP-FAST server and Inner Method server might be a single entity; the authenticator and EAP-FAST server might be a single entity; or, indeed, the functions of the authenticator, 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 (12) provides an additional discussion of the implications of separating EAP-FAST from the inner method.

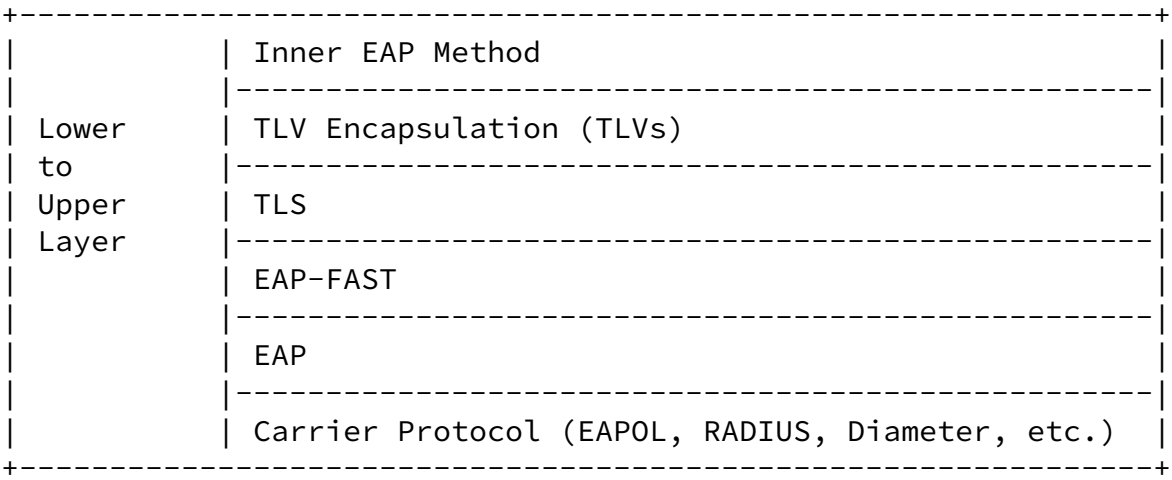
4. Protocol Layering Model

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 method is a payload with standard Type-Length-Value (TLV) objects. 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 method. When the user authentication protocol is itself EAP, the layering is as follows:



Methods for encapsulating EAP within carrier protocols are already

defined. For example, 802.1X EAPOL may be used to transport EAP between client and access point; RADIUS or Diameter are used to transport EAP between authenticator and EAP-FAST server.

5. Protected Access Credential (PAC) for EAP-FAST Authentication

A pre-shared secret mutually and uniquely shared between the peer and AS is used to secure a tunnel during EAP-FAST Authentication. EAP-FAST uses a Protected Access Credential (PAC) to facilitate the use of a single shared secret by the peer and minimize the per user state management on the AS. The PAC is a security credential provided by the AS 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 maps as the TLS pre-master-secret. The PAC-Key is randomly generated by the AS to produce a strong entropy 32-octet key.

2. PAC-Opaque: this is a variable length field that is sent to the AS during the EAP-FAST Phase 1 tunnel establishment. The PAC-Opaque can only be interpreted by the AS 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 issuing PAC server.

3. PAC-Info: this is a variable length field used to provide at minimum, the authority identity or PAC issuer. Other useful

but not mandatory information, such as the PAC-Key lifetime, may also be conveyed by the AS to the peer during PAC provisioning or refreshment.

As the focus of this draft is to define the EAP-FAST protected tunneling and authentication mechanism, it does not address provisioning. That is, provisioning of the PAC may be achieved using the same mechanisms as the provisioning of any other credential such as certificates or username/password credential types.

6. EAP-FAST Authentication

To establish a new session, EAP-FAST employs the PAC to invoke an authenticated key agreement exchange to establish a protected tunnel. Once the tunnel is established, the peer and AS can ensue in further conversations to establish the required authentication and authorization policies. Part of the authorization policy is the generation of the Master Session Keys (MSKs). Portions of the MSKs may be distributed to the NAS using the RADIUS MS-MPPE [RFC2548] attribute. Finally, the server may also update the PAC as part of the EAP-FAST protocol conclusion. This section describes the two phases of EAP-FAST Authentication: Phase 1, the tunnel establishment and Phase 2, the tunneled authentication.

6.1 EAP-FAST Authentication Phase 1: Tunnel Establishment

This conversation is similar to establishing a new EAP-TLS session except it uses new EAP type (EAP-FAST).

The initial conversation begins with the authenticator and the peer negotiating EAP. The authenticator will typically send an EAP-Request/Identity packet to the peer, and the peer will respond with an EAP-Response/Identity packet to the authenticator, containing the username. If the client desires to protect its identity, it may use an anonymous username.

Once the initial Identity Request/Response exchange is completed, while the EAP conversation typically occurs between the authenticator and the peer, the authenticator may act as a pass-through device, with the EAP packets received from the peer being encapsulated for transmission to a backend authentication server. In the discussion that follows, the term "EAP server" or "server" is used to denote the ultimate endpoint conversing with the peer.

Once having received the peer's Identity, and determined that EAP-FAST Authentication is to occur, the EAP server must respond with a EAP-FAST/Start packet, which is an EAP-Request packet with EAP-Type=EAP-FAST and the Start (S) bit set. The EAP-FAST/Start packet shall also include an authority identity (A-ID) TLV to better inform the peer the server's identity. Assuming that the peer

supports EAP-FAST, the EAP-FAST conversation will then begin, with

the peer sending an EAP-Response packet with EAP-Type=EAP-FAST.

The data field of the EAP-Response packet contains an EAP-FAST encapsulated TLS ClientHello handshake message.

The ClientHello message contains the peer's challenge (also called the client_random) and PAC-Opaque. As there may be different EAP-FAST servers a peer may encounter, a peer may be provisioned with unique PACs uniquely identified by the A-ID corresponding to the EAP-FAST server. A peer may choose to cache the different PACs and determine based on the A-ID the corresponding PAC to employ. While EAP-FAST is capable of supporting any ciphersuite, in this version, the ClientHello uses the TLS_RSA_WITH_RC4_128_SHA ciphersuite. As EAP-FAST uses the PAC to establish the keys, the RSA key exchange is not executed, but the specification of RC4 and SHA signals the EAP server that the tunnel must be protected using 128bit RC4 for confidentiality and SHA1 for authenticity.

The EAP server will then respond with an EAP-Request packet with EAP-Type=EAP-FAST. The data field of this packet will encapsulate three TLS records, ServerHello, ChangeCipherSpec and Finished messages. The ServerHello will contain a server_random and ChangeCipherSpec. The TLS Finished message, sent immediately after the ChangeCipherSpec message, contains the first protected message with the negotiated algorithm, keys, and secrets.

The server generates the master_secret prior to composing the EAP-FAST TLS ServerHello message to properly generate the TLS Finished message contents. The server must compute the Tunnel Keys as described in Section 6.2 at this time to properly respond and generate its TLS Finished message.

The peer in turn, must consume the ServerHello to extract the server_random before it can generate the master_secret and Tunnel Keys, as described in Section 6.2.

After verifying the server Finished message, the peer responds back with two TLS records, a ChangeCipherSpec and the peer's TLS Finished message. At this state, the client is ready to receive and transmit protected messages with the server.

Upon verifying the peer's Finished message, the EAP server establishes the tunnel and is ready for the receiving and transmitting protected messages with the peer. The messages are protected using the Tunnel Keys described in Section 6.2.

6.2 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.

An EAP-FAST specific PRF function, T-PRF described in Appendix B (Section 18) is used to generate a fresh master_secret from the specified client_random, server_random and PAC-Key.

The PRF function used to generate keying material is defined by [RFC 2246].

Since a PAC may be used as a credential for other applications beyond EAP-FAST, the PAC-Key is further hashed using T-PRF to generate a fresh TLS master_secret. Additionally, the hash of PAC-Key is required to stretch it to the required 48 octet master_secret:

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

To generate the key material required for EAP-FAST Authentication, the following TLS construction is used:

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

where '+' denotes concatenation.

Since this version of EAP-FAST Authentication employs 128bit RC4 and SHA1, the key_block is partitioned as follows:

```
client_write_MAC_secret[hash_size=20]
server_write_MAC_secret[hash_size=20]
client_write_key[Key_material_length=16]
server_write_key[key_material_length=16]
client_write_IV[IV_size=0]
server_write_IV[IV_size=0]
session_key_seed[seed_size= 40]
```

The client_write_MAC_secret and server_write_MAC_secret are the keys used by the client and server to authenticate subsequent messages respectively. Similarly, the client_write_key and

client_write_IV are used by the client to provide message confidentiality while the server uses the server_write_key and server_write_IV to achieve confidentiality. The session_key_seed is later 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.

6.3 EAP-FAST Authentication Phase 2: Tunneled Authentication

The second portion of the EAP-FAST Authentication conversation consists of at least one complete EAP conversation occurring within

the TLS session negotiated in EAP-FAST Authentication Phase 1; ending with protected termination using the Result-TLV and Crypto-Binding TLV. All EAP messages are encapsulated in the EAP Message TLV.

EAP-FAST Phase 2 will occur only if establishment of a new TLS session in Phase 1 is successful or a TLS session is successfully resumed in Phase 1.

Phase 2 must not occur if the EAP Peer or EAP Server fails authentication during Phase 1. That is, if the tunnel establishment fails and a TLS alert is provided prior to a cleartext EAP failure.

Additionally, Phase 2 must not occur if a protected EAP-Failure has been sent by the EAP Server to the peer, terminating the conversation. Since all packets sent within the EAP-FAST Phase 2 conversation occur after TLS session establishment, they are protected using the negotiated TLS cipher suite. For example, if the cipher suite negotiated is TLS_RSA_WITH_RC4_128_SHA, all EAP-TLV packets of the conversation in Phase 2 including the EAP-TLV header are protected using 128bit RC4 and SHA1 as defined by the TLS protocol [RFC 2246].

6.4 Protected EAP Conversation

Phase 2 of the EAP-FAST Authentication conversation consists of at least one protected EAP authentication, typically using the peer's credentials (typically username and password). This entire EAP

conversation including the user identity and EAP type are protected from eavesdropping and modification by the tunnel encapsulation. A hacker cannot readily determine the EAP method used (except perhaps by traffic analysis) nor can the hacker inject/modify packets to subvert the authentication.

Phase 2 of the EAP-FAST conversation begins with the EAP server sending an EAP-Request/Identity packet to the peer, protected by the TLS ciphersuite negotiated in EAP-FAST Phase 1. The peer responds with an EAP-Response/Identity packet to the EAP server, containing the peer's userID.

After the protected Identity exchange, the EAP server will send an EAP-Request with the supported EAP type, for example, EAP-Type=EAP-GTC. EAP-FAST enables the use of any (EAP) method to ensue inside the tunnel, the EAP-GTC type is used in this specification as an example.

The EAP conversation within the TLS protected session may involve zero or more EAP authentication methods, including the EAP-TLV method; and completes with protected termination shown in Section 6.5.

After any EAP method, the EAP-FAST server or peer may request the EAP-FAST peer or server respectively, to prove that it has participated in the sequence of authentications successfully completed until that point. The server also concludes the EAP-FAST Phase 2 conversation by invoking a final Result TLV with a Crypto-Binding TLV. The Crypto-Binding TLV is sent in the protected TLS channel. If the EAP-FAST server sends a valid Crypto-Binding TLV to the EAP-FAST peer, the peer MUST respond with a Crypto-Binding TLV in an EAP Response. If the Crypto-Binding TLV is invalid, it MUST be considered failed authentication by EAP-FAST client and a Result TLV with a failure status should follow. If the peer does not respond with an EAP-FAST packet containing the crypto-binding TLV, it MUST be considered failed authentication by the EAP-FAST server. Once the EAP-FAST peer and EAP-FAST server considers them as failed authentications, they are the last packets inside the protected tunnel.

6.5 Protected Termination and Acknowledged Result Indication

The EAP-FAST server and EAP-FAST peer indicate success/failure of a conversation ensued inside the TLS tunnel. Either an Intermediate Result TLV is used if further conversations will occur, or a final Result TLV if it is the concluding success/failure indication. The inclusion of a Crypto-Binding TLV exchange is used to prove that both peers participated in the sequence of authentications (specifically the TLS session and inner authentication methods that generate keys). The Crypto-Binding-TLV exchange is only needed with a Success Result TLV to verify the integrity of the tunnel. If the inner EAP method fails, then no Crypto-Binding-TLV exchange is needed.

If the PAC needs to be updated, the Crypto-Binding TLV must precede the final Result TLV as the final Result TLV exchange also includes the distribution of the PAC in a PAC TLV. Following a successful Intermediate Result TLV and Crypto-Binding TLV exchange, the Result TLV will be the next subsequent EAP-TLV exchange that also includes a PAC TLV to update the PAC.

Both Intermediate and final Result TLVs are sent protected within the TLS channel. The EAP-FAST peer then replies with a corresponding Intermediate or final Result TLV inside protected channel. The conversation concludes with a final Result TLV exchange followed by the EAP-FAST server sending a cleartext EAP-Success/Failure indication.

The only outcome which should be considered as a successful EAP-FAST Authentication is when the final Result TLV of Status=Success and a valid concluding Crypto-Binding TLV, is answered by a final Result TLV of Status=Success and a valid Crypto-Binding-TLV.

All other combinations of the (request, response) Result TLVs such as (Failure, Success), (Failure, Failure), (no Result TLV exchange, no Crypto-Binding TLVs or where the Crypto-Binding TLV validation

is not successful) MUST be considered failed authentications, both by the EAP-FAST peer and EAP-FAST server. Once the EAP-FAST peer and EAP-FAST server considers them as failed authentications, they are the last packets inside the protected tunnel. These are considered failed authentications regardless of whether a cleartext EAP Success or EAP Failure packet is subsequently sent.

In support for session resumption, an EAP-FAST server may send the

success indication and Crypto-Binding TLV, without initiating any EAP conversation in EAP-FAST Phase 2. The EAP-FAST client is allowed to refuse to accept a success message from the EAP-FAST server since the client's policy may require completion of certain authentication methods. If session resume is not invoked and the EAP-FAST server has sent Result-TLV with Status=Success; and the response from the EAP peer is Status=Failure, then the server MUST continue with the EAP-FAST Phase 2 authentication conversation.

6.6 EAP-FAST Authentication Phase 2: Key Derivations

Keying material resulting from all successful conversations ensued in both phases of EAP-FAST Authentication are used to both prove tunnel integrity and generate session keys. A base compound key is the resulting key generated as follows:

EAP-FAST session_key_seed(SKS) is a 40 octet value obtained from the EAP-FAST Authentication Phase 1 described in Section 6.2.

The inner authentication method(s) provide session keys: ISK1..ISKn corresponding to inner methods 1 through n. Only the MSKs from the inner methods are required. If the inner method (i) does not generate an ISK, then ISKi is set to zero (e.g. ISKi = 32 octets of 0x00s). If the inner method generates keying material, EAP-FAST presumes that a minimum of 32 octets are provided. Otherwise, the resulting ISK is padded with zeroes to generate a 32 octet value. Thus, the first 32 octets generated as the encryption keying material by the inner method is used and assigned as the ISK. For example, if EAP-TLS [RFC 2716] is used as an inner method, the resulting first 32 bytes described as the "peer encryption key" in Section 3.5 of [RFC 2716] is assigned as the ISK.

The algorithm uses the EAP-FAST T-PRF as described in Appendix B (Section 18) to generate the following:

```
S-IMCK = SKS           0
For j = 1 to n-1 do
IMCK[j] = T-PRF(S-IMCK[j-1], "Inner Methods Compound Keys", ISK[j],
60);
   Where S-IMCK[j] are the first 40 octets of IMCK[j]
```

ICMK[j] may generate up to 60 octets of keying material. The first 40 octets are used as the key input to the succeeding ICMK[j+1] derivation and the latter 20 octets are used as the key, CMK[j],

used to generate the intermediate Crypto-Binding Compound MAC value.

6.7 Cryptographic Binding: Computing the Compound MAC

For authentication methods that generate keying material, further protection against man-in-the-middle attacks are mitigated through the enforcement of cryptographically binding keying material established by both EAP-FAST Phase 1 and EAP-FAST Phase 2 conversations.

For a successful EAP-FAST Authentication, inner methods are cryptographically combined to generate a compound session key, CMK, used to generate an authentication tag referred to as a Compound MAC and transported in a Crypto-Binding TLV. The Crypto-Binding TLV is used to assure that the same peers invoked all methods in EAP-FAST.

EAP-FAST optionally enables the server or client to invoke Intermediate Result-TLV request/response exchanges with Crypto-Binding TLVs to verify the integrity of the tunnel between methods inside the EAP-FAST Phase 2 conversation.

Similarly, EAP-FAST enforces a mandatory inclusion of a Crypto-Binding TLV after a final method has completed. In both instances, a Crypto-Binding TLV is included when either an Intermediate Result TLV or a final Result TLV is used. The Crypto-Binding TLV includes a 20 octet authentication tag that represents the HMAC-SHA1 hash of the entire Crypto-Binding TLV. The Compound MAC field is zeroed out prior to the computation of the HMAC-SHA1 and subsequently populated with the resulting hash value.

The requesting server shall provide a 32-octet random server_nonce with its last bit set to 0 and compute the Compound MAC field as follows:
HMAC-SHA1(CMK, [Crypto-Binding TLV with Compound MAC field= zeroes])

The responding peer shall respond with the same 32-octet server_nonce value provided by the requestor with its last bit set to 1 and computes the responding Compound MAC field as described above.

6.8 EAP-FAST Authentication: Session Key Generation

EAP-FAST Authentication assures the master session keys are a result of all conversations ensued by generating a compound session key (IMCK). The IMCK is mutually derived by the peer and server

using the T-PRF; the IMCK calculation is defined in Section 6.6. The resulting master session key, MSK, is generated as part of the IMCKn key hierarchy. Where the S-IMCKn is used to generate the session keys as follows:

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

The first version of EAP-FAST generates 64 octets to serve as the successful EAP-FAST authentication master session keys. Interpretation and assignment of these 64 octets of the master session key is specific to each link layer ciphersuite.

EAP-FAST implementations MAY generate EMSK as follows:

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

The Extended Master Session Key (EMSK) is only known to the EAP-FAST peer and server and is not provided to a third party.

6.9 PAC Distribution and Refreshing

The server may distribute or refresh a peer's PAC after a successful EAP-FAST Authentication. A PAC TLV is created to facilitate the distribution and update. A fresh PAC may be distributed after a successful Intermediate Result TLV and Crypto-Binding TLV exchange. A successful EAP-FAST authentication, including a successful Crypto-Binding exchange must ensue before an EAP-FAST server can distribute a fresh PAC. A PAC TLV should not be accepted if it is not TLS tunnel-encapsulated. The fresh PAC is encapsulated in a PAC TLV containing the PAC-Key, PAC-Opaque and PAC-Info TLVs. The PAC-Key is the shared secret key the peer uses to mutually authenticate with the server and establish the tunnel. The PAC-Opaque contains data that is opaque to the recipient, the peer is not the intended consumer of PAC-Opaque and thus should not attempt to interpret it. A peer that has been issued a PAC-Opaque by a server must store that data, and present it back to the server as is, in the TLS ClientHello extension [RFC3546]. PAC-Info provides the peer information about the PAC, at minimum, it provides the information about the authority identity issuing the

PAC.

Once the EAP-FAST peer receives a PAC TLV, it needs to securely save the new PAC-Key, PAC-Opaque and optionally, the PAC-Info. Additionally, upon receipt of a new PAC, the peer must respond with a successful PAC-Acknowledgement TLV. If the peer responds with a PAC-Acknowledgement failure, the EAP-FAST server may invoke another Result TLV failure resulting in a failed EAP-FAST authentication.

The server may refresh a PAC only after a successful exchange of the concluding Intermediate Result TLV and Crypto-Binding TLV. The peer must use the new PAC-Key and PAC-Info in subsequent EAP-FAST Authentication sessions.

N.B. In-band PAC refreshing is enforced by server policy. The server, based on the PAC-Opaque information, may determine not to refresh a peer's PAC through the PAC TLV mechanism even if the PAC-Key has expired.

7. 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.

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 client 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 client 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 client, it terminates the conversation.

The version negotiation procedure guarantees that the EAP-FAST client 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 a EAP-FAST version negotiated, the peers MUST exchange information on the EAP-FAST version negotiated using the Crypto-Binding TLV. The concluding Intermediate or final Result TLV comes with a mandatory Crypto-Binding TLV that includes the EAP-FAST version which MUST be consistent to that specified in the EAP-FAST Start message.

8. Error Handling

The EAP-FAST protocol uses TLS alert messages to communicate and handle error conditions in all phases of EAP-FAST. Errors during the tunnel establishment or protection in EAP-FAST Authentication are handled via TLS alert messages, while errors during the protected tunnel are expected to be handled by the individual EAP methods. Intermediate Result TLVs are also used as status indications of the individual EAP methods in EAP-FAST Phase 2.

If the EAP-FAST server detects an error at any point in the EAP-FAST conversation, the EAP-FAST server should send an EAP-Request packet with EAP-Type=EAP-FAST, encapsulating a TLS record containing the appropriate TLS alert message.

The EAP-FAST server should send a 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. To ensure that the peer receives the TLS alert message, the EAP server must wait for the peer to reply with an EAP-Response packet before terminating the connection.

The EAP-Response packet sent by the peer may encapsulate a TLS client_hello handshake message, in which case the EAP-FAST server

MAY allow the EAP-FAST conversation to be restarted, or it MAY contain an EAP-Response packet with EAP-Type=EAP-FAST and Flags and Version fields without any additional data , in which case the EAP-FAST Server MUST send an EAP-Failure packet, and terminate the conversation.

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 client detects an error at any point in the EAP-FAST conversation, the EAP-FAST client should send an EAP-Response packet with EAP-Type=EAP-FAST, encapsulating a TLS record containing the appropriate TLS alert message. The EAP-FAST server may restart the conversation by sending an EAP-Request packet encapsulating the TLS server_hello handshake message, in which case the EAP-FAST client may allow the EAP-FAST conversation to be restarted; or terminate the conversation.

If during the EAP-FAST Authentication Phase 1 session establishment, EAP-FAST servers cannot obtain or verify the PAC, the server should send an EAP-Request packet with EAP-Type=EAP-FAST, encapsulating a TLS record containing the appropriate TLS alert message, before terminating the conversation. The EAP-FAST peer should inform the use of the mismatching PAC and terminating the conversation.

8.1 Error Alerts

EAP-FAST uses TLS-Alert to handle errors in the EAP-TLS handshake. EAP-FAST employs the standard TLS error alerts described in TLS Protocol Specification [RFC 2246]. In addition, it reuses the following TLS alert to support EAP-FAST specific error conditions:

bad certificate

Server cannot find an acceptable PAC-Opaque in the ClientHello message. This can be a result of either the peer not sending the PAC-Opaque or the PAC-Opaque provided cannot be decrypted by the

server or expired. This message is always a fatal error.

decrypt_error

If PAC-Key doesn't match between the peer and server and either peer or server fails to validate Finished message, decrypt_error Alert should be used. This message is always a fatal error.

9. Session Resume

EAP-FAST offers a means to bypass further conversations such as inner EAP authentication methods when a peer has an established session identified by Session ID. This enables a peer to optimally generate fresh master session keys without having to re-invoke the inner EAP authentication method in EAP-FAST Authentication Phase 2. Applications that require user intervention for the inner authentication method (e.g. OTP) can benefit from this feature when service has been established but believes it must refresh its master session keys.

EAP-FAST session resumption is achieved in the same manner TLS achieves session resume. Session Resume is achieved by the peer responding with a known Session ID in its ClientHello record. The EAP-FAST Authentication Phase 1 conversation proceeds in a similar fashion as described in Section 6 with the exception of the use of the PAC-Opaque in the ClientHello. That is, a session resumption is distinguished by the client's indication of a valid (e.g. non-zero) SessionID and omission of the PAC-Opaque in its ClientHello message. To support session resumption, the server must minimally cache the client's SessionID, master_secret and CipherSpec. If the server finds a match for the SessionID and is willing to establish a new connection using the specified session state, the server will respond with the same SessionID and proceed with the EAP-FAST Authentication Phase 1 tunnel establishment described in Section 6.1. The key derivations used in the EAP-FAST Authentication Phase 1 employ the corresponding SessionID's master_secret in accordance to the TLS [RFC 2246] session resumption specification.

After a successful conclusion of the EAP-FAST Authentication Phase 1 conversation, the server then decides whether to honor session resumption based on the Session ID value. It may reject and initiate the inner EAP authentication method to signal the start of a full EAP-FAST Authentication Phase 2 conversation. The server may accept a session resumption based on the Session ID specified by the peer as well as the time elapsed since the previous authentication.

The server may accept a session resumption and bypass the inner EAP authentication method by immediately requesting a final Result TLV without a Crypto-Binding TLV. The concluding Result TLV exchange is the same as that described in Section 6.5. The EAP-FAST master session keys are generated as described in Section 6.8, with the exception that S-IMCK[n] is SKS without going through the compound key derivation, as in this case no inner EAP method has run.

Even if the session is successfully resumed, the peer and EAP-FAST server must not assume that either will skip inner EAP methods. The peer may have roamed to a network which may use the same EAP server, but may require conformance with a different authentication policy. After a session is successfully resumed, the EAP-Server may start a full Phase 2 of the EAP-FAST Authentication conversation.

10. Fragmentation

A single TLS record may be up to 16384 octets in length, but a TLS message may span multiple TLS records, and a TLS certificate message may in principle be as long as 16MB. The group of EAP-FAST messages sent in a single round may thus be larger than the PPP MTU size, the maximum RADIUS packet size of 4096 octets, or even the Multilink Maximum Received Reconstructed Unit (MRRU). As described in [2], the multilink MRRU is negotiated via the Multilink MRRU LCP option, which includes an MRRU length field of two octets, and thus can support MRRUs as large as 64 KB.

However, note that in order to protect against reassembly lockup and denial of service attacks, it may be desirable for an implementation to set a maximum size for one such group of TLS messages. Since a typical certificate chain is rarely longer than a few thousand octets, and no other field is likely to be anywhere near as long, a reasonable choice of maximum acceptable message length might be 64 KB.

If this value is chosen, then fragmentation can be handled via the multilink PPP fragmentation mechanisms described in [RFC1990]. While this is desirable, EAP methods are used in other applications such as [IEEE80211] and there may be cases in which multilink or the MRRU LCP option cannot be negotiated. As a result, an EAP-FAST implementation MUST provide its own support for fragmentation and reassembly.

Since EAP is an ACK-NAK protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or

damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field as is provided in IPv4.

EAP-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=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.

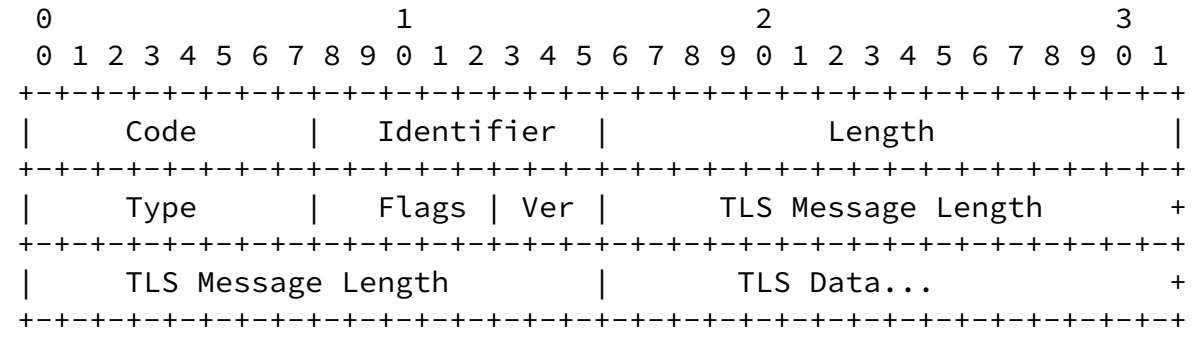
Similarly, when the EAP-FAST server receives an EAP-Response with the M bit set, it must respond with an EAP-Request with EAP-Type=EAP-FAST and no data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors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

11. EAP-FAST Detailed Description

11.1 EAP-FAST Packet Format

A summary of the EAP-FAST Request/Response packet format is shown below.

The fields are transmitted from left to right.



Code

1 - Request
2 - Response

Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field MUST be changed on each Request packet.

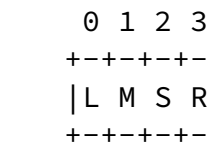
Length

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

Type

43 - EAP-FAST

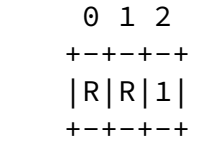
Flags



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

The L bit (length included) is set to indicate the presence of the four octet TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS message or set of messages. The M bit (more fragments) is set on all but the last fragment. The S bit (EAP-FAST Start) is set in an EAP-FAST Start message. This differentiates the EAP-FAST Start message from a fragment acknowledgement.

Version



R = Reserved (must be zero)

TLS Message Length

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

TLS data

The TLS data consists of the encapsulated packet in TLS record format. An EAP-FAST packet with Flags and Version fields but with empty data field to used to indicate EAP-FAST acknowledgement for either TLS Alert or TLS Finished.

11.2 EAP-FAST TLV Format

The EAP-FAST TLV is a payload with standard Type-Length-Value (TLV) objects similar to those defined by [PEAP]. The TLV objects could be used to carry arbitrary parameters between EAP peer and EAP server. Possible uses for TLV objects include: language and character set for Notification messages; cryptographic binding; IPv6 Binding Update.

The EAP peer may not necessarily implement all the TLVs supported by the EAP server; and hence to allow for interoperability, the TLV method allows 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 that the peer must understand the TLV. A peer can determine that a TLV is unknown when it does not support the TLV; or when the TLV is corrupted. The mandatory bit does not indicate that the peer successfully applied the value of the TLV. The specification of a TLV could define additional conditions under which the TLV can be determined to be unknown.

If an EAP peer finds an unknown TLV which is marked as mandatory; it must indicate a failure to the EAP server using the NAK TLV; and all the other TLVs in the message MUST be ignored.

If an EAP peer finds an unknown TLV which is marked as optional; then it must ignore the TLV. The EAP peer is not required to inform the EAP server of unknown TLVs which are marked as optional. If the EAP peer finds that the packet has no TLVs, then it must send a response with EAP-TLV Response Packet. The Response packet may contain no TLVs.

If an EAP server finds an unknown TLV which is marked as mandatory; the other TLVs in the message MUST be ignored. The EAP server can drop the connection or send an EAP-TLV request packet with NAK-TLV to the EAP client.

An EAP-FAST TLV packet can be sequenced before or after any other EAP method. The packet does not have to contain any mandatory TLVs.

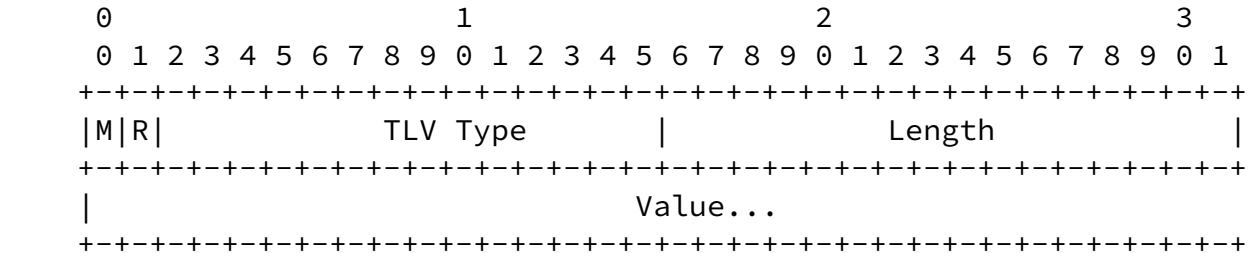
Compliant EAP-FAST implementations must support the EAP-FAST TLV

exchange, including processing of mandatory/optional settings on the TLV, the NAK TLV, the Crypto-Binding TLV, EAP Payload TLV, PAC TLV, Intermediate Result TLV and the Result TLV.

The EAP-TLV Request and Response packets shown below are included in this specification to serve as information only. The actual EAP-FAST inner method packets inside the TLS tunnel are all encapsulated using the EAP-TLV TLV format, instead of the EAP-TLV format. The EAP-TLV header are not needed and thus omitted, since all inner method packets are encapsulated in EAP-TLV.

11.3 TLV format

EAP-FAST TLVs are defined as follows:



M

0 - Non-mandatory 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 - Reserved
6 - Reserved
7 - Reserved
8 - Reserved
9 - EAP Payload TLV
10 - Intermediate Result TLV
11 - PAC TLV
12 - Crypto-Binding TLV

Length

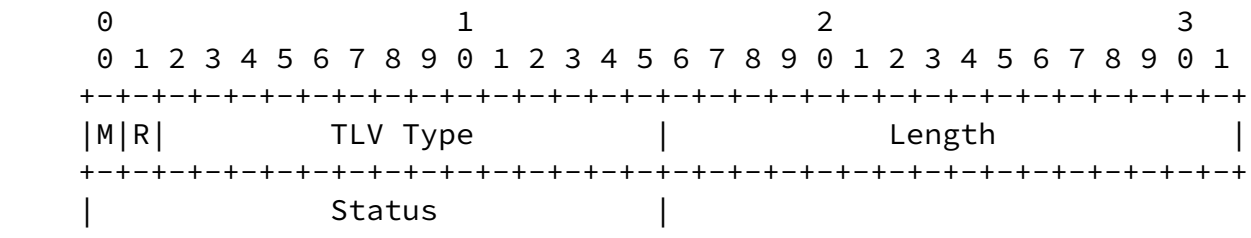
The length of the Value field in octets.

Value

The value of the TLV.

11.4 Result TLV

The Result TLV provides support for acknowledged Success and Failure messages within EAP-FAST. It is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

3 - Result TLV

Length

2

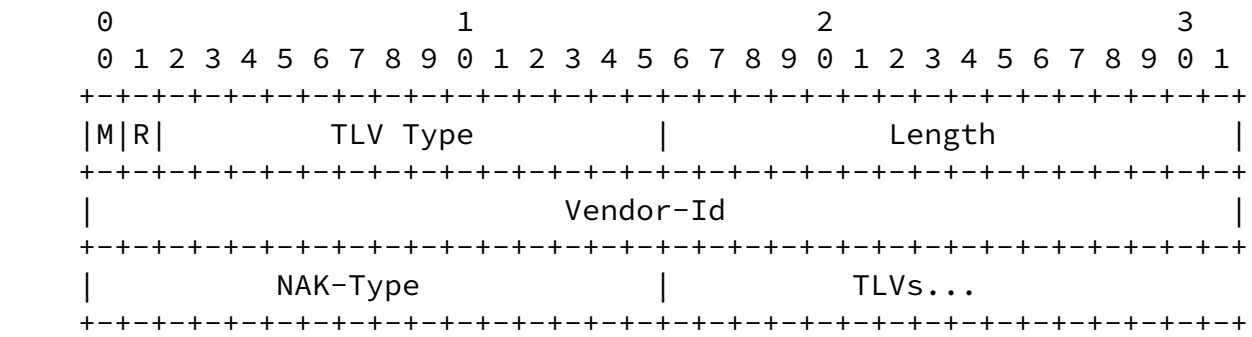
Status

The status field is two octets. Values include:

1 - Success
2 - Failure

11.5 NAK TLV

The NAK TLV allows a peer to detect when TLVs that are not supported by the other peer. It is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

4 - NAK TLV

Length

Variable

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. For Vendor-Specific TLVs, the Vendor-ID MUST be set to the SMI code.

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 can be used in the

future to communicate why the offending TLV was determined to be unsupported.

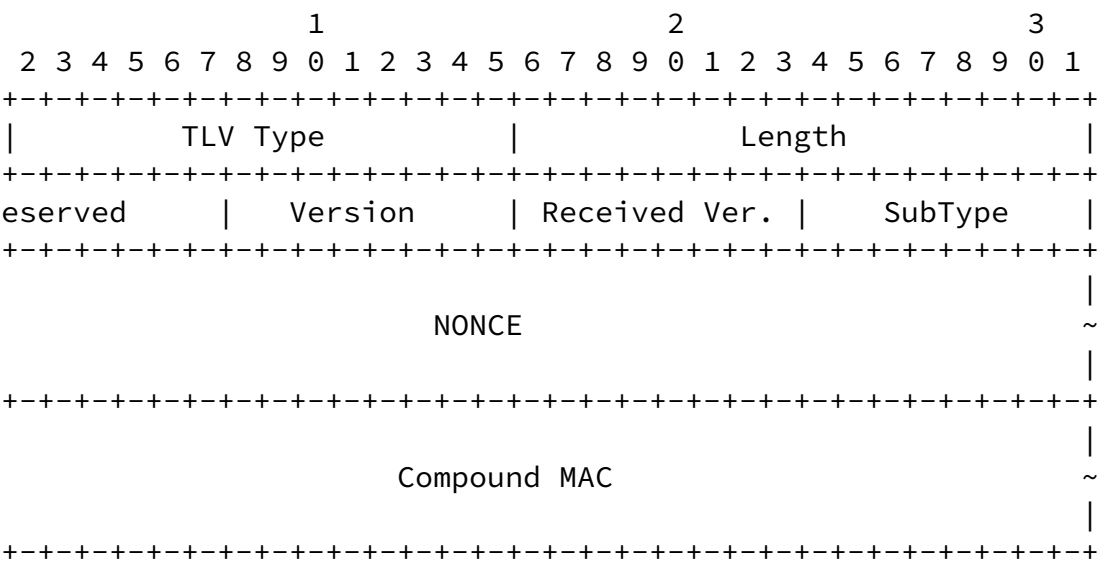
11.6 Crypto-Binding TLV

The Crypto-Binding TLV is used to prove that both peers participated in the sequence of authentications (specifically the TLS session and inner EAP methods that generate keys).

Both the Binding Request and Binding Response use the same packet format; with the SubType field indicating whether it is a request or response.

The Crypto-Binding TLV can be used to perform Cryptographic Binding after each EAP method in a sequence of EAP methods completes within the EAP-FAST Authentication Phase 2. The Crypto-Binding TLV MUST be used once during or before a Protected Termination along with a Result or Intermediate TLV.

This message format is used for the Binding Request and also the Binding Response. This uses TLV type CRYPTO_BINDING_TLV. The format is as given below, with fields transmitted from left to right:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

12 - Crypto-Binding TLV.

Length

56

Reserved

The Reserved field is a single octet and must be set to all zeros.

Version

The Version field is a single octet, which is set to the version of Crypto Binding TLV. For the crypto-binding TLV defined in this specification, it is set to 1.

Received Version

The Received Version field is a single octet and MUST be set to the EAP-FAST version number received during version negotiation.

SubType

The SubType field is 1 octet.
0 - Binding Request
1 - Binding Response

Nonce

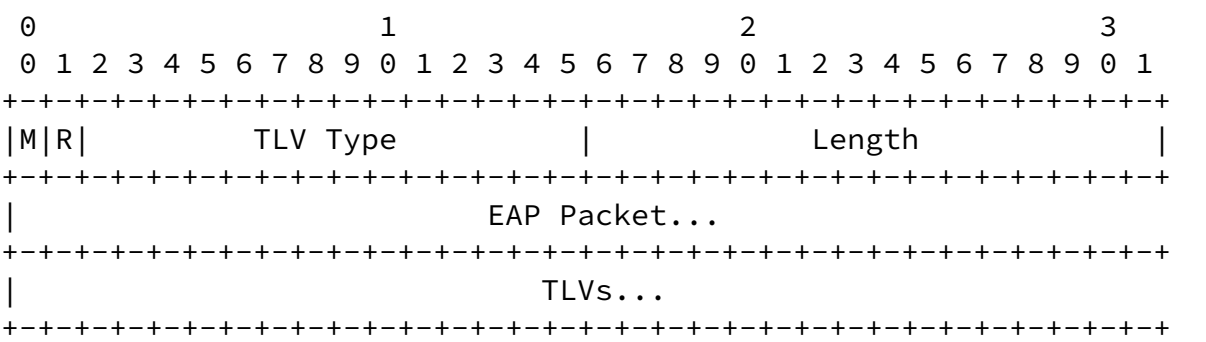
The Nonce field is 32 octets. It contains a 256 bit random number generated by the server on request. The peer responds with the server nonce incremented by 1.

Compound MAC

The Compound MAC field is 20 octets. It contains an authentication tag for this TLV. It is calculated over entire Crypto-binding TLV with Compound MAC field filled with zero.

11.7 EAP Payload TLV

EAP Payload TLV is used to encapsulate all the EAP messages. It is defined as follows:



M

  1 - Mandatory TLV

R

  Reserved, set to zero (0)

TLV Type

  9

Length

  >=0

EAP Packet

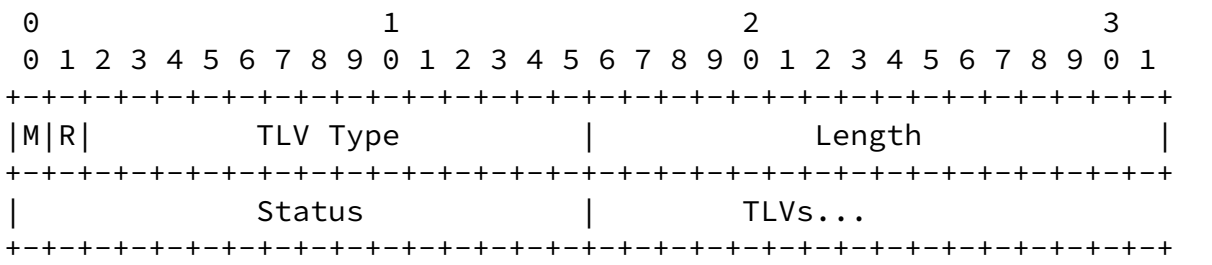
  This field contains a complete EAP packet, including the EAP header (Code, Identifier, Length, Type) fields. The length of this field is determined by the Length field of the encapsulated EAP packet.

TLVs...

  This (optional) field contains a list of TLVs associated with the EAP packet field. The TLVs utilize the same format described Section 4.3, and 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.

11.8 Intermediate Result TLV

The Intermediate Result TLV provides support for acknowledged intermediate Success and Failure messages within EAP-FAST. EAP-FAST implementations MUST support this TLV; and this TLV cannot be responded to with a NAK TLV. It is defined as follows:



M

1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

10

Length

>=2

Status

The Status field is two octets. Values include:

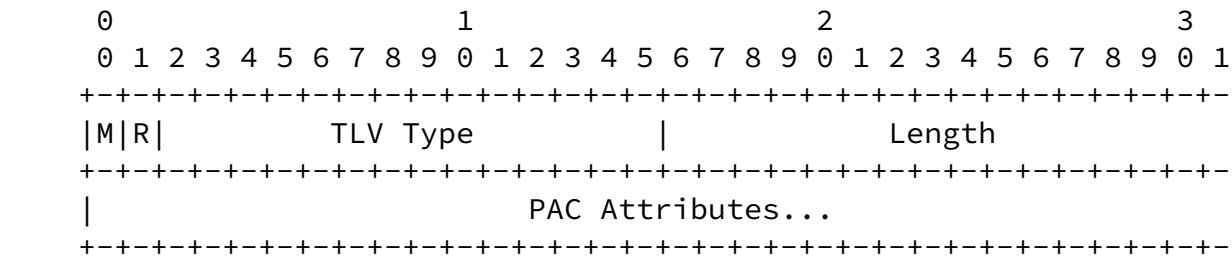
1 - Success
2 - Failure

TLVs

This (optional) field is of indeterminate length, and contains the TLVs associated with the Intermediate Result TLV, in the same format as described in Section 4.3. The TLVs in this field MUST NOT have the mandatory bit set.

11.9 PAC TLV

The PAC TLV provides support for acknowledged refreshing from the server side within EAP-FAST. A consistent PAC format will allow it to be used by multiple applications beyond EAP-FAST. A general PAC TLV format is defined as follows:



M

0 - Non-mandatory TLV
1 - Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

11- PAC TLV:

Length

The length of the PAC Attributes field in octets.

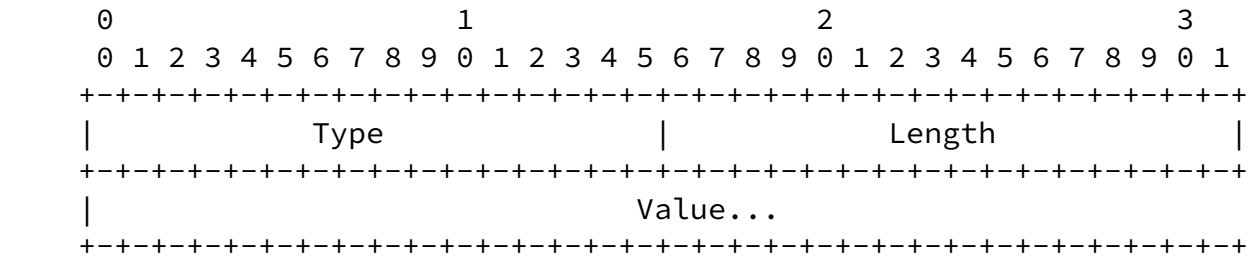
PAC Attributes

A list of PAC attributes in the TLV format.

A PAC attribute is comprised of three general PAC fields encapsulated in a common format. The contents of these fields are described in succeeding sections. The PAC TLV contains all of the required information to appropriately distribute the client with a PAC.

11.9.1 Formats for PAC TLV attributes

A common encapsulating format is used to convey the different fields that comprise a PAC attribute. The common encapsulation is defined as follows:



Type

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

1 - PAC-Key
2 - PAC-Opaque
3 - CRED_LIFETIME
4 - A-ID
5 - I-ID
6 - SERVER_PROTECTED_DATA
7 - A-ID-Info
8 - PAC-Acknowledgement
9 - PAC-Info

Length

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

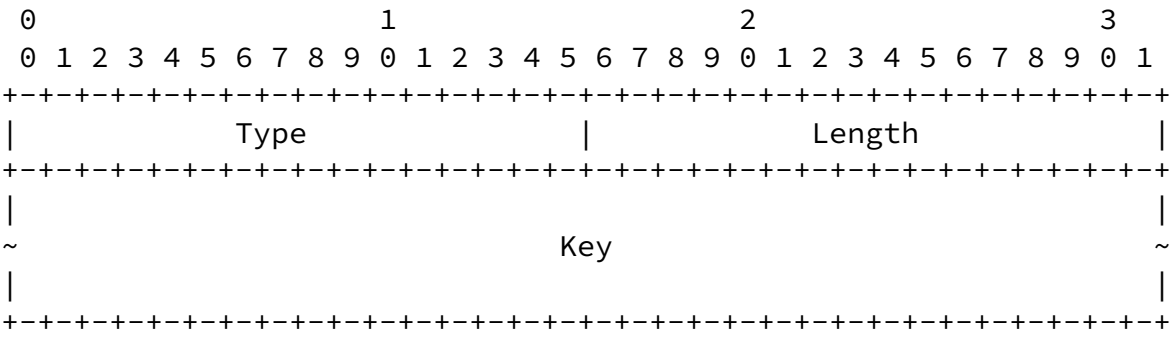
Value

The value of the PAC Attribute.

11.9.2 PAC-Key

The PAC-Key is distributed as an attribute of type PAC-Key (Type=1). The key is a randomly generated octet string. The key is represented as an octet string whose length is determined by the

length field. The generator of this key is the issuer of the credential, identified by the A-ID.



Type

1 - PAC-Key

Length

The Length filed is two octets. For this version of EAP-FAST, PAC-Key is 32 octets.

Key

The Key field contains the PAC-Key.

11.9.3 PAC-Opaque

The PAC-Opaque section contains data that is opaque to the recipient, the peer is not the intended consumer of PAC-Opaque and thus should not attempt to interpret it. A peer that has been issued a PAC-Opaque by a server MUST store that data, and present it back to the server as is, in the ClientHello extension field

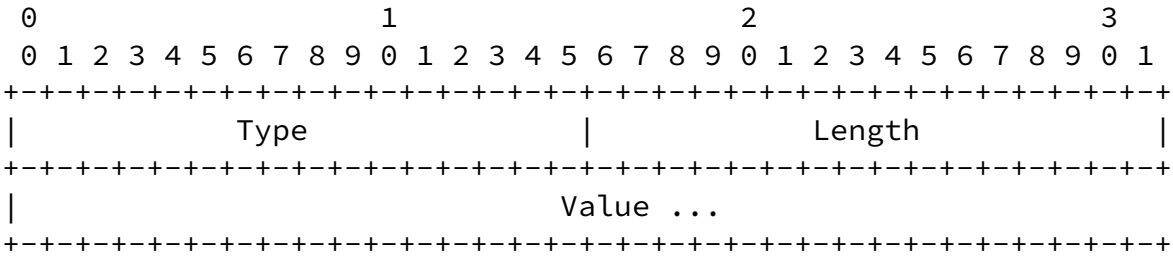
[RFC3546]. If a client has opaque data issued to it by multiple servers, then it MUST store the data issued by each server separately. This requirement allows the client to maintain and use each opaque data as an independent PAC pairing, with a PAC-Key mapping to a PAC-Opaque identified by the A-ID. As there is a one

to one correspondence between PAC-Key and PAC-Opaque, a peer must determine the PAC-Key and corresponding PAC-Opaque based on the A-ID provided in the EAP-FAST/Start message and the A-ID provided in the PAC-Info when it was provisioned with a PAC-Opaque. Each client must not parse any PAC-Opaque data given to it.

As the PAC-Opaque is server specific, its contents and definition are specific to the issuer of the PAC, e.g. the PAC server.

The PAC-Opaque field is embedded as part of the PAC TLV when the server has determined that the PAC must be refreshed and sends a new PAC.

The PAC-Opaque field format is summarized as follows:



Type

2 - PAC-Opaque

Length

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

Value

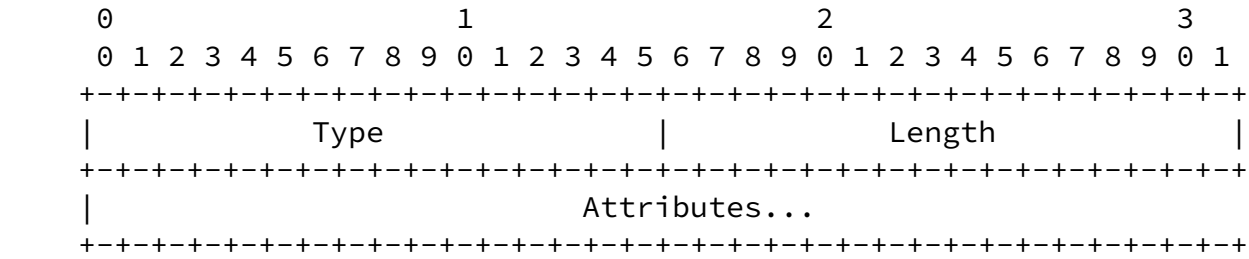
The Value field contains the actual data for PAC-Opaque

The PAC-Opaque field is also passed from the peer to the server during the EAP-FAST Authentication Phase 1 conversation to enable the server to validate and recreate the PAC-Key. When it is passed from the peer, it is encapsulated as defined above in the ClientHello.

11.9.4 PAC-Info

PAC-Info is comprised of a set of PAC attributes. At minimum, the A-ID TLV is required to convey the issuing identity to the peer. Other optional fields may be included in the PAC to provide more

information to the peer. It is a container attribute for various types of information each of which is encoded in conformance to the PAC TLV field format as defined in Section 11.3.



Type

9 - PAC-Info

Length

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

Attributes

The Attributes field contains a list of PAC Attributes.

Each mandatory and optional field type is defined as follows:

CRED_LIFETIME (type 3)
This is a 4 octet quantity representing the expiration time of the credential in UNIX UTC time. This is a mandatory field contained in the PAC-Opaque field to enable the server to validate the PAC. This field may also be optionally provided to the peer as part of PAC-Info.

A-ID (type 4)
Authority identifier is the name of the authority that issued the token. The A-ID is intended to be unique across all issuing servers to avoid namespace collisions. Server implementations should use measures to ensure the A-ID used is globally unique to avoid name collisions. The A-ID is used by the peer to determine which PAC to employ. Similarly, the server uses the A-ID to both authenticate the PAC-Opaque and determine which master key was used to issue the PAC. This field is mandatory and included in both the PAC-Opaque and as the first TLV comprising PAC-Info.

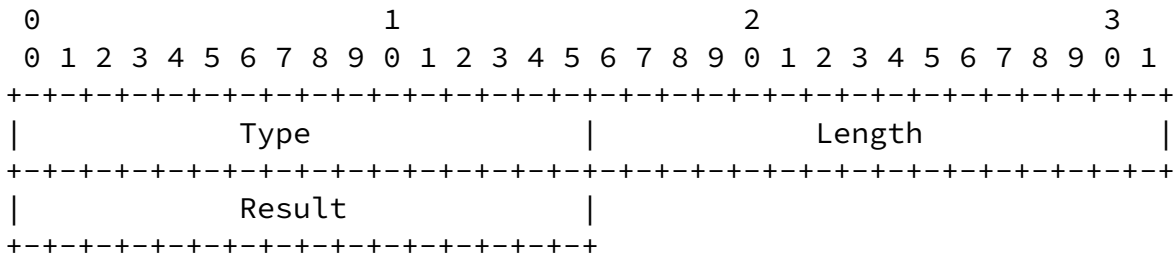
I-ID (type 5)
Initiator identifier is the peer identity associated with the credential. The server employs the I-ID in the EAP-FAST Phase 2 conversation to validate that the same peer identity used to execute EAP-FAST Phase 1 is also used in at minimum one inner EAP method in EAP-FAST Phase 2. This field is a mandatory field in

PAC-Opaque and may optionally be included in the PAC-Info. If the AS is enforcing the I-ID validation on inner EAP method, then I-ID is mandatory in PAC-Info, to enable the client to also enforce a unique PAC for each unique user. If I-ID is missing from the PAC-Info, it is assumed that the PAC can be used for multiple users and client will not enforce the unique PAC per user policy.

A-ID-Info (type 7)
Authority Identifier Information is a mandatory TLV intended to provide a user-friendly name for the A-ID. It may contain the enterprise name and server name in a more human-readable format. This TLV serves as an aid to the peer to better inform the end-user about the A-ID. This field is a mandatory field in the PAC-Info.

11.9.5 PAC-Acknowledgement TLV

The PAC-Acknowledgement TLV is used to acknowledge the receipt of the PAC TLV by the peer. Peer sends this TLV in response to the PAC TLV to indicate the result of the retrieving and storing of the new PAC.



Type

8 - PAC-Acknowledgement

Length

2

Result

1 - Success
2 - Failure

12. 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 RFC 2284bis [EAP].

12.1 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. Further, the Result TLV is enforced to be run after any EAP method that supports (mutual) authentication ensuring that it was the same peer and AS 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 cleartext EAP Success or Failure, a peer will also receive a cleartext EAP success or failure. The received cleartext EAP

success or failure must match that received in the Result TLV; the peer may silently discard those cleartext EAP success or failure messages that do not coincide with the status sent in the protected Result TLV.

12.2 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.

12.3 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.

12.4 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 cleartext 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 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].

12.5 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
* EAP negotiation
* Sequencing of EAP methods
* Strong mutually derived master session keys
* Support for fragmentation and reassembly
* Acknowledged success/failure indication
* Faster re-authentications through session resumption
* Mitigation of dictionary attacks
* Mitigation of man-in-the-middle attacks
* 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 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 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.

12.5.1 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 userID 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 backend 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.

12.5.2 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.

12.5.3 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 way:

* 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,

* 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.

12.5.4 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 cleartext, 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 cleartext 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.

12.6 PAC Storage Considerations

The main premise behind EAP-FAST is to protect the authentication stream over the media link. However, physical security is still an issue. Some care should be taken to protect the PAC on both the peer and server. The peer must store securely both the PAC-Key and PAC-Opaque, while the server must secure storage of its security association context used to consume the PAC-Opaque. Additionally, if manual provisioning is employed, the transportation mechanism used to distribute the PAC must also be secured.

Most of the attacks described here would require some level of effort to execute; conceivably greater than their value. The main focus therefore, should be to ensure that proper protections are used on both the client and server. There are a number of potential attacks which can be considered against secure key storage such as:

* weak passphrases
  On the client side, keys are usually protected by a passphrase.

On some environments, this passphrase may be associated with the user's password. In either case, if an attacker can obtain the encrypted key for a range of users, he may be able to successfully attack a weak passphrase. The tools are already in place today to allow an attacker to easily attack all Outlook or Outlook Express users in an enterprise environment. Most viruses or worms of this sort attract attention to themselves by their action, but that need not be the case. A simple, genuine appearing email could surreptitiously access keys from known locations and email them directly to the attacker, attracting little notice.

* key finding attacks
  Key finding attacks are usually mentioned in reference to web servers, where the private SSL key may be stored securely, but at some point it must be decrypted and stored in system memory. An attacker with access to system memory can actually find the key by identifying their mathematical properties. To date, this attack appears to be purely theoretical and primarily acts to argue strongly for secure access controls on the server itself to prevent such unauthorized code from executing.

* key duplication , key substitution, key modification
  Once keys are accessible to an attacker on either the client or server, they fall under three forms of attack: key duplication, key substitution and key modification. The first option would be the most common, allowing the attacker to masquerade as the user in question. The second option could have some use if an attacker could implement it on the server.

Another consideration is the use of secure mechanisms afforded by the particular device. For instance, some laptops enable secure key storage through a special chip. It would be worthwhile for implementations to explore the use of such a mechanism.

12.7 Protecting against Forged Clear Text EAP Packets

As described earlier, EAP Success and EAP Failure packets are in general sent in cleartext 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 cleartext 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 2284, the AS must send a cleartext 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.

12.8 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.

12.9 Security Claims

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

Auth. mechanism:	Tunneled authentication as well as pre-shared key.

```
        Ciphersuite negotiation:Yes. See [RFC2246].
        Mutual authentication:  Yes. See Section 12.1.
        Integrity protection:   Yes. See Section 12.1. Only EAP Type
                                Data field and inner EAP methods
```



```
                                contained in this field are
                                protected.
        Replay protection:      Yes. See [RFC2246].
        Confidentiality:        Yes. See [RFC2246].
        Key derivation:         Yes. See Section 6.6.
        Key strength:           TLS key strength, may be enhanced by
                                binding keys with inner methods
        Dictionary attack prot.:Yes. See Section 12.5.2.
        Key hierarchy:          Yes. See Section 6.6.
        Fast reconnect:         Yes. See Section 9.
        Cryptographic binding:  Yes. See Section 6.7.
        Session independence:   Yes. See [RFC2716].
        Fragmentation:          Yes. See Section 10.
        Channel binding:        No.
```

13. 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 namespace in EAP-FAST that requires registration: TLV Types. These numbers may be assigned by First Come First Served [RFC2434].

14. References

14.1 Normative

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

[EAP]
Blunk, L., et. al., "Extensible Authentication Protocol (EAP)", RFC3748, June 2004.

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

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

[RFC 3546]
Blake-Wilson, S., et al, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

14.2 Informative

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

[RFC2279]
Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2631]
Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, January 1999.

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

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

[SHARED KEYS IN TLS]

Gutmann, P., "Use of Shared Keys in the TLS Protocol", draft-ietf-tls-sharedkeys-02.txt (expired), October 2003.

[IKEv2]
Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", draft-ietf-ipsec-ikev2-17 (work in progress), September 2004.

[PEAP]
Palekar, et. al., "Protected EAP Protocol (PEAP) Version 2", draft-josefsson-pppext-eap-tls-eap-10 (work in progress), October 2004

[RFC 3526]
Kivinen, T., "More Modular Exponential (MODP) DIffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003

[MITM] Puthenkulam, J., "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04 (expired), October 2003.

[RFC2486BIS]
Aboba, et. al., "The Network Access Identifier", draft-arkko-roamops-rfc2486bis-02.txt (work in progress), July, 2004.

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

[EAP-TTLS]
Funk & Blake-Wilson, "EAP Tunneled TLS Authentication Protocol", draft-ietf-pppext-eap-ttls-05.txt (work in progress), July 2004

[TLS-PSK]
Eronen & Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", draft-ietf-tls-psk-02, Septempber 2004

15. Acknowledgments

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

16. Author's Addresses

Nancy Cam-Winget
Cisco Systems
3625 Cisco Way
San Jose, CA 95134
US
Phone: +1 408 853 0532
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
Phone: +1 206 256 3380
Email: jsalowey@cisco.com

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

Phone : +1 330 523 2132
Email: hzhou@cisco.com

17. Appendix A: Examples

17.1 Successful Authentication

The following exchanges show a successful EAP-FAST authentication with 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)->

                        <- 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)
                        Binding-TLV(Version=0,SNonce,
                        CompoundMAC)
```

```
   Intermediate Result TLV (Success)
   Binding-TLV(Version=0,
   CNonce, CompoundMAC) ->

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

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

   TLS channel torn down
   (messages sent in cleartext)

                            <- EAP-Success
```

17.2 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)->

                           <- 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

    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 cleartext)

                               <- EAP-Failure
```

18. Appendix B: EAP-FAST PRF (T-PRF)

EAP-FAST employs a simpler PRF than the TLS PRF where possible. For instance, when generating the master_secret, master session keys and cryptographic binding keys and computations, EAP-FAST employs the following PRF construction:

PRF(key, label, seed) = HMAC-SHA1(key, label + "\0" + seed)

Where '+' indicates concatenation and "\0" is a NULL character. Label is intended to be a unique label for each different use of the T-PRF.

To generate the desired OutputLength octet length of key material, the T-PRF is iterated as follows:

```
T-PRF (Key, S, OutputLength) = T1 + T2 + T3 + T4 + ...
      Where S = label + 0x00 + seed;  and

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)
```

OutputLength is a two octet value that is represented in big endian order. The NULL character, 0x00 shall be present when a label string is provided. Also note that the seed value may be optional and may be omitted as in the case of the MSK derivation described in Section 6.8.

Where each Ti generates 20-octets of keying material, the last Tn may be truncated to accommodate the desired length specified by OutputLength.

19. Appendix C: Test Vectors

19.1 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
```

19.2 Crypto-Bind 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
```


20. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any

Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

21. Disclaimer of Validity

22. Copyright Statement

23. Expiration Date

This memo is filed as <draft-cam-winget-eap-fast-01.txt>, and expires April 25, 2005.