Internet Engineering Task Force (IETF)
Request for Comments: 7170
Category: Standards Track
ISSN: 2070-1721

H. Zhou
N. Cam-Winget
J. Salowey
Cisco Systems
S. Hanna
Infineon Technologies

May 2014

Tunnel Extensible Authentication Protocol (TEAP) Version 1

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7170.

Table of Contents

1. Introduction

A tunnel-based Extensible Authentication Protocol (EAP) method is an EAP method that establishes a secure tunnel and executes other EAP methods under the protection of that secure tunnel. A tunnel-based EAP method can be used in any lower-layer protocol that supports EAP authentication. There are several existing tunnel-based EAP methods that use Transport Layer Security (TLS) [RFC5246] to establish the secure tunnel. EAP methods supporting this include Protected EAP (PEAP) [PEAP], EAP Tunneled Transport Layer Security (EAP-TTLS) [RFC5281], and EAP Flexible Authentication via Secure Tunneling (EAP-FAST) [RFC4851]. However, they all are either vendor-specific or informational, and the industry calls for a Standards Track tunnel-based EAP method. [RFC6678] outlines the list of requirements for a standard tunnel-based EAP method.

Since its introduction, EAP-FAST [RFC4851] has been widely adopted in a variety of devices and platforms. It has been adopted by the EMU working group as the basis for the standard tunnel-based EAP method. This document describes the Tunnel Extensible Authentication Protocol (TEAP) version 1, based on EAP-FAST [RFC4851] with some minor changes to meet the requirements outlined in [RFC6678] for a standard tunnel-based EAP method.

1.1. Specification Requirements

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

1.2. Terminology

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

Protected Access Credential (PAC)

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

Type-Length-Value (TLV)

> The TEAP protocol utilizes objects in TLV format. The TLV format is defined in Section 4.2.

2. Protocol Overview

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

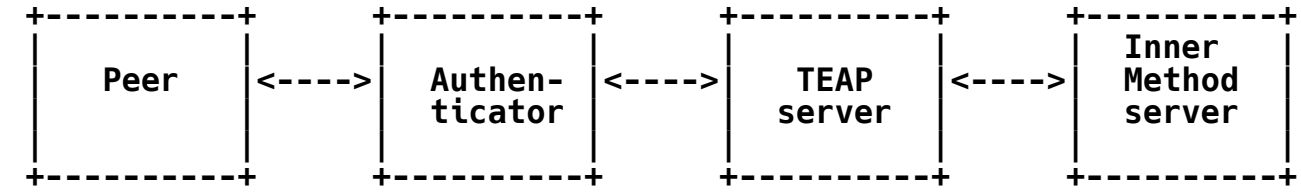
TEAP makes use of the TLS SessionTicket extension [RFC5077], which supports TLS session resumption without requiring session-specific state stored at the server. In this document, the SessionTicket is referred to as the Protected Access Credential opaque data (or PAC-Opaque). The PAC-Opaque may be distributed through the use of the NewSessionTicket message or through a mechanism that uses TLVs within Phase 2 of TEAP. The secret key used to resume the session in TEAP is referred to as the Protected Access Credential key (or PAC-Key). When the NewSessionTicket message is used to distribute the PAC-Opaque, the PAC-Key is the master secret for the session. If TEAP

Phase 2 is used to distribute the PAC-Opaque, then the PAC-Key is distributed along with the PAC-Opaque. TEAP implementations MUST support the [RFC5077] mechanism for distributing a PAC-Opaque, and it is RECOMMENDED that implementations support the capability to distribute the ticket and secret key within the TEAP tunnel.

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

2.1. Architectural Model

The network architectural model for TEAP usage is shown below:

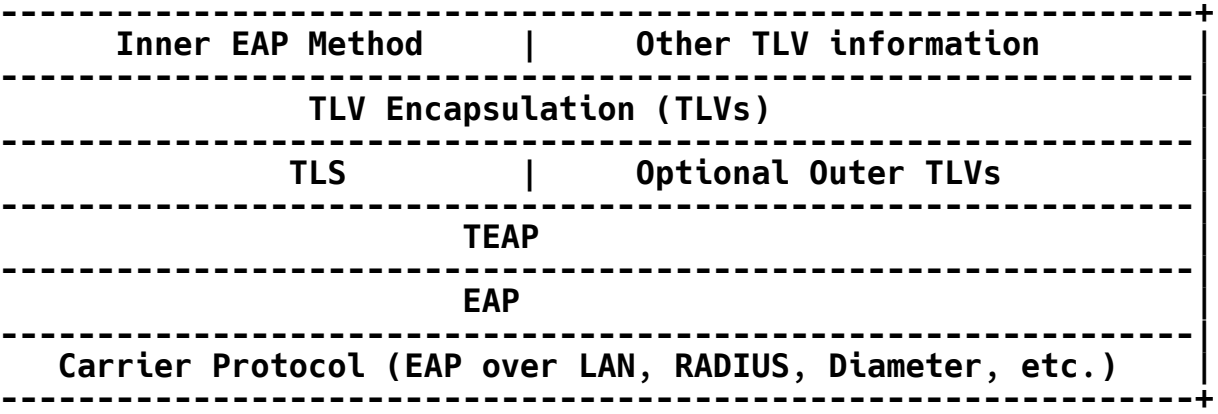


TEAP Architectural Model

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

2.2. Protocol-Layering Model

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



Protocol-Layering Model

The TLV layer is a payload with TLV objects as defined in Section 4.2. The TLV objects are used to carry arbitrary parameters between an EAP peer and an EAP server. All conversations in the TEAP protected tunnel are encapsulated in a TLV layer.

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

Methods for encapsulating EAP within carrier protocols are already defined. For example, IEEE 802.1X [IEEE.802-1X.2013] may be used to transport EAP between the peer and the authenticator; RADIUS [RFC3579] or Diameter [RFC4072] may be used to transport EAP between the authenticator and the EAP server.

3. TEAP Protocol

The operation of the protocol, including Phase 1 and Phase 2, is the topic of this section. The format of TEAP messages is given in Section 4, and the cryptographic calculations are given in Section 5.

3.1. Version Negotiation

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

Version negotiation proceeds as follows:

1. In the first EAP-Request sent with EAP type=TEAP, the EAP server MUST set the Version field to the highest version it supports.

2a. If the EAP peer supports this version of the protocol, it responds with an EAP-Response of EAP type=TEAP, including the version number proposed by the TEAP server.

2b. If the TEAP peer does not support the proposed version but supports a lower version, it responds with an EAP-Response of EAP type=TEAP and sets the Version field to its highest supported version.

2c. If the TEAP peer only supports versions higher than the version proposed by the TEAP server, then use of TEAP will not be possible. In this case, the TEAP peer sends back an EAP-Nak either to negotiate a different EAP type or to indicate no other EAP types are available.

3a. If the TEAP server does not support the version number proposed by the TEAP peer, it MUST either terminate the conversation with an EAP Failure or negotiate a new EAP type.

3b. If the TEAP server does support the version proposed by the TEAP peer, then the conversation continues using the version proposed by the TEAP peer.

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

The TEAP version is not protected by TLS and hence can be modified in transit. In order to detect a modification of the TEAP version, the peers MUST exchange the TEAP version number received during version negotiation using the Crypto-Binding TLV described in Section 4.2.13. The receiver of the Crypto-Binding TLV MUST verify that the version received in the Crypto-Binding TLV matches the version sent by the receiver in the TEAP version negotiation. If the Crypto-Binding TLV fails to be validated, then it is a fatal error and is handled as described in Section 3.6.3.

3.2. TEAP Authentication Phase 1: Tunnel Establishment

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

TLS_RSA_WITH_AES_128_CBC_SHA [RFC5246]

TLS_DHE_RSA_WITH_AES_128_CBC_SHA [RFC5246]

This version of the TEAP implementation SHOULD support the following TLS ciphersuite:

TLS_RSA_WITH_AES_256_CBC_SHA [RFC5246]

Other ciphersuites MAY be supported. It is REQUIRED that anonymous ciphersuites such as TLS_DH_anon_WITH_AES_128_CBC_SHA [RFC5246] only be used in the case when the inner authentication method provides mutual authentication, key generation, and resistance to man-in-the-middle and dictionary attacks. TLS ciphersuites that do not provide confidentiality MUST NOT be used. During the TEAP Phase 1 conversation, the TEAP endpoints MAY negotiate TLS compression. During TLS tunnel establishment, TLS extensions MAY be used. For instance, the Certificate Status Request extension [RFC6066] and the Multiple Certificate Status Request extension [RFC6961] can be used to leverage a certificate-status protocol such as Online Certificate Status Protocol (OCSP) [RFC6960] to check the validity of server certificates. TLS renegotiation indications defined in RFC 5746 [RFC5746] MUST be supported.

The EAP server initiates the TEAP conversation with an EAP request containing a TEAP/Start packet. This packet includes a set Start (S) bit, the TEAP version as specified in Section 3.1, and an authority identity TLV. The TLS payload in the initial packet is empty. The authority identity TLV (Authority-ID TLV) is used to provide the peer a hint of the server's identity that may be useful in helping the

peer select the appropriate credential to use. Assuming that the peer supports TEAP, the conversation continues with the peer sending an EAP-Response packet with EAP type of TEAP with the Start (S) bit clear and the version as specified in Section 3.1. This message encapsulates one or more TLS handshake messages. If the TEAP version negotiation is successful, then the TEAP conversation continues until the EAP server and EAP peer are ready to enter Phase 2. When the full TLS handshake is performed, then the first payload of TEAP Phase 2 MAY be sent along with a server-finished handshake message to reduce the number of round trips.

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

The TEAP tunnel protects peer identity information exchanged during Phase 2 from disclosure outside the tunnel. Implementations that wish to provide identity privacy for the peer identity need to carefully consider what information is disclosed outside the tunnel prior to Phase 2. TEAP implementations SHOULD support the immediate renegotiation of a TLS session to initiate a new handshake message exchange under the protection of the current ciphersuite. This allows support for protection of the peer's identity when using TLS client authentication. An example of the exchanges using TLS renegotiation to protect privacy is shown in Appendix C.

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

3.2.1. TLS Session Resume Using Server State

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

3.2.2. TLS Session Resume Using a PAC

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

1. PAC-Key: this is the key used by the peer as the TLS master secret to establish the TEAP Phase 1 tunnel. The PAC-Key is a strong, high-entropy, at minimum 48-octet key and is typically the master secret from a previous TLS session. The PAC-Key is a secret and MUST be treated accordingly. Otherwise, if leaked, it could lead to user credentials being compromised if sent within the tunnel established using the PAC-Key. In the case that a PAC-Key is provisioned to the peer through another means, it MUST have its confidentiality and integrity protected by a mechanism, such as the TEAP Phase 2 tunnel. The PAC-Key MUST be stored securely by the peer.

2. PAC-Opaque: this is a variable-length field containing the ticket that is sent to the EAP server during the TEAP Phase 1 tunnel establishment based on [RFC5077]. The PAC-Opaque can only be interpreted by the EAP server to recover the required information for the server to validate the peer's identity and authentication. The PAC-Opaque includes the PAC-Key and other TLS session parameters. It may contain the PAC's peer identity. The PAC-Opaque format and contents are specific to the PAC issuing server. The PAC-Opaque may be presented in the clear, so an attacker MUST NOT be able to gain useful information from the PAC-Opaque itself. The server issuing the PAC-Opaque needs to ensure it is protected with strong cryptographic keys and algorithms. The PAC-Opaque may be distributed using the NewSessionTicket message defined in [RFC5077], or it may be distributed through another mechanism such as the Phase 2 TLVs defined in this document.

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

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

3.2.3. Transition between Abbreviated and Full TLS Handshake

If session resumption based on server-side or client-side state fails, the server can gracefully fall back to a full TLS handshake. If the ServerHello received by the peer contains an empty Session ID or a Session ID that is different than in the ClientHello, the server may fall back to a full handshake. The peer can distinguish the server's intent to negotiate a full or abbreviated TLS handshake by checking the next TLS handshake messages in the server response to the ClientHello. If ChangeCipherSpec follows the ServerHello in response to the ClientHello, then the server has accepted the session resumption and intends to negotiate the abbreviated handshake. Otherwise, the server intends to negotiate the full TLS handshake. A peer can request that a new PAC be provisioned after the full TLS handshake and mutual authentication of the peer and the server. A peer SHOULD NOT request that a new PAC be provisioned after the abbreviated handshake, as requesting a new session ticket based on resumed session is not permitted. In order to facilitate the

fallback to a full handshake, the peer SHOULD include ciphersuites that allow for a full handshake and possibly PAC provisioning so the server can select one of these in case session resumption fails. An example of the transition is shown in Appendix C.

3.3. TEAP Authentication Phase 2: Tunneled Authentication

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

3.3.1. EAP Sequences

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

EAP method messages are carried within EAP-Payload TLVs defined in Section 4.2.10. If more than one method is going to be executed in the tunnel, then upon method completion, the server MUST send an Intermediate-Result TLV indicating the result. The peer MUST respond to the Intermediate-Result TLV indicating its result. If the result indicates success, the Intermediate-Result TLV MUST be accompanied by

a Crypto-Binding TLV. The Crypto-Binding TLV is further discussed in Sections 4.2.13 and 5.3. The Intermediate-Result TLVs can be included with other TLVs such as EAP-Payload TLVs starting a new EAP conversation or with the Result TLV used in the protected termination exchange.

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

3.3.2. Optional Password Authentication

The use of EAP-FAST-GTC as defined in RFC 5421 [RFC5421] is NOT RECOMMENDED with TEAPv1 because EAP-FAST-GTC is not compliant with EAP-GTC defined in [RFC3748]. Implementations should instead make use of the password authentication TLVs defined in this specification. The authentication server initiates password authentication by sending a Basic-Password-Auth-Req TLV defined in Section 4.2.14. If the peer wishes to participate in password authentication, then it responds with a Basic-Password-Auth-Resp TLV as defined in Section 4.2.15 that contains the username and password. If it does not wish to perform password authentication, then it responds with a NAK TLV indicating the rejection of the Basic-Password-Auth-Req TLV. Upon receiving the response, the server indicates the success or failure of the exchange using an Intermediate-Result TLV. Multiple round trips of password authentication requests and responses MAY be used to support some "housecleaning" functions such as a password or pin change before a user is authenticated.

3.3.3. Protected Termination and Acknowledged Result Indication

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

The Crypto-Binding TLV exchange is used to prove that both the peer and server participated in the tunnel establishment and sequence of authentications. It also provides verification of the TEAP type,

version negotiated, and Outer TLVs exchanged before the TLS tunnel establishment. The Crypto-Binding TLV MUST be exchanged and verified before the final Result TLV exchange, regardless of whether or not there is an inner EAP method authentication. The Crypto-Binding TLV and Intermediate-Result TLV MUST be included to perform cryptographic binding after each successful EAP method in a sequence of one or more EAP methods. The server may send the final Result TLV along with an Intermediate-Result TLV and a Crypto-Binding TLV to indicate its intention to end the conversation. If the peer requires nothing more from the server, it will respond with a Result TLV indicating success accompanied by a Crypto-Binding TLV and Intermediate-Result TLV if necessary. The server then tears down the tunnel and sends a cleartext EAP Success or EAP Failure.

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

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

Error handling for Phase 2 is discussed in Section 3.6.3.

3.4. Determining Peer-Id and Server-Id

The Peer-Id and Server-Id [RFC5247] may be determined based on the types of credentials used during either the TEAP tunnel creation or authentication. In the case of multiple peer authentications, all authenticated peer identities and their corresponding identity types (Section 4.2.3) need to be exported. In the case of multiple server authentications, all authenticated server identities need to be exported.

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

> The subject field identifies the entity associated with the public key stored in the subject public key field. The subject name MAY be carried in the subject field and/or the subjectAltName extension. . . . If subject naming information is present only in the subjectAltName extension (e.g., a key bound only to an email address or URI), then the subject name MUST be an empty sequence and the subjectAltName extension MUST be critical.
>
> Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN).

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

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

3.5. TEAP Session Identifier

The EAP session identifier [RFC5247] is constructed using the tls-unique from the Phase 1 outer tunnel at the beginning of Phase 2 as defined by Section 3.1 of [RFC5929]. The Session-Id is defined as follows:

> Session-Id = teap_type || tls-unique
>
> where teap_type is the EAP Type assigned to TEAP
>
> tls-unique = tls-unique from the Phase 1 outer tunnel at the beginning of Phase 2 as defined by Section 3.1 of [RFC5929]
>
> || means concatenation

3.6. Error Handling

TEAP uses the error-handling rules summarized below:

1. Errors in the outer EAP packet layer are handled as defined in Section 3.6.1.

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

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

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

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

3.6.1. Outer-Layer Errors

Errors on the TEAP outer-packet layer are handled in the following ways:

1. If Outer TLVs are invalid or contain unknown values, they will be ignored.

2. The entire TEAP packet will be ignored if other fields (version, length, flags, etc.) are inconsistent with this specification.

3.6.2. TLS Layer Errors

If the TEAP server detects an error at any point in the TLS handshake or the TLS layer, the server SHOULD send a TEAP request encapsulating a TLS record containing the appropriate TLS alert message rather than immediately terminating the conversation so as to allow the peer to inform the user of the cause of the failure and possibly allow for a restart of the conversation. The peer MUST send a TEAP response to an alert message. The EAP-Response packet sent by the peer may encapsulate a TLS ClientHello handshake message, in which case the TEAP server MAY allow the TEAP conversation to be restarted, or it MAY contain a TEAP response with a zero-length message, in which case the server MUST terminate the conversation with an EAP Failure packet. It is up to the TEAP server whether or not to allow restarts, and, if allowed, how many times the conversation can be restarted. Per TLS [RFC5246], TLS restart is only allowed for non-fatal alerts. A TEAP server implementing restart capability SHOULD impose a limit on the number of restarts, so as to protect against denial-of-service attacks. If the TEAP server does not allow restarts, it MUST terminate the conversation with an EAP Failure packet.

If the TEAP peer detects an error at any point in the TLS layer, the TEAP peer SHOULD send a TEAP response encapsulating a TLS record containing the appropriate TLS alert message. The server may restart the conversation by sending a TEAP request packet encapsulating the

TLS HelloRequest handshake message. The peer may allow the TEAP conversation to be restarted, or it may terminate the conversation by sending a TEAP response with a zero-length message.

3.6.3. Phase 2 Errors

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

For the inner method, retransmission is not needed and SHOULD NOT be attempted, as the Outer TLS tunnel can be considered a reliable transport. If there is a non-fatal error handling the inner method, instead of silently dropping the inner method request or response and not responding, the receiving side SHOULD use an Error TLV with error code Inner Method Error to indicate an error processing the current inner method. The side receiving the Error TLV MAY decide to start a new inner method instead or send back a Result TLV to terminate the TEAP authentication session.

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

3.7. Fragmentation

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

fragmentation and reassembly. Section 3.1 of [RFC3748] discusses determining the MTU usable by EAP, and Section 4.3 discusses retransmissions in EAP.

Since EAP is a lock-step protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field.

TEAP fragmentation support is provided through the addition of flag bits within the EAP-Response and EAP-Request packets, as well as a Message Length field of four octets. Flags include the Length included (L), More fragments (M), and TEAP Start (S) bits. The L flag is set to indicate the presence of the four-octet Message Length field and MUST be set for the first fragment of a fragmented TLS message or set of messages. It MUST NOT be present for any other message. The M flag is set on all but the last fragment. The S flag is set only within the TEAP start message sent from the EAP server to the peer. The Message Length field is four octets and provides the total length of the message that may be fragmented over the data fields of multiple packets; this simplifies buffer allocation.

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

Similarly, when the TEAP server receives an EAP-Response with the M bit set, it responds with an EAP-Request with EAP Type of TEAP and no data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors in the processing of fragments, the EAP server MUST increment the Identifier value for each fragment ACK contained within an EAP-Request, and the peer MUST include this Identifier value in the subsequent fragment contained within an EAP-Response.

3.8. Peer Services

Several TEAP services, including server unauthenticated provisioning, PAC provisioning, certificate provisioning, and channel binding, depend on the peer trusting the TEAP server. Peers MUST authenticate

the server before these peer services are used. TEAP peer implementations MUST have a configuration where authentication fails if server authentication cannot be achieved. In many cases, the server will want to authenticate the peer before providing these services as well.

TEAP peers MUST track whether or not server authentication has taken place. Server authentication results if the peer trusts the provided server certificate. Typically, this involves both validating the certificate to a trust anchor and confirming the entity named by the certificate is the intended server. Server authentication also results when the procedures in Section 3.2 are used to resume a session in which the peer and server were previously mutually authenticated. Alternatively, peer services can be used if an inner EAP method providing mutual authentication and an Extended Master Session Key (EMSK) is executed and cryptographic binding with the EMSK Compound Message Authentication Code (MAC) is correctly validated (Section 4.2.13). This is further described in Section 3.8.3.

An additional complication arises when a tunnel method authenticates multiple parties such as authenticating both the peer machine and the peer user to the EAP server. Depending on how authentication is achieved, only some of these parties may have confidence in it. For example, if a strong shared secret is used to mutually authenticate the user and the EAP server, the machine may not have confidence that the EAP server is the authenticated party if the machine cannot trust the user not to disclose the shared secret to an attacker. In these cases, the parties who participate in the authentication need to be considered when evaluating whether to use peer services.

3.8.1. PAC Provisioning

To request provisioning of a PAC, a peer sends a PAC TLV as defined in Section 4.2.12 containing a PAC Attribute as defined in Section 4.2.12.1 of PAC-Type set to the appropriate value. The peer MUST successfully authenticate the EAP server and validate the Crypto-Binding TLV as defined in Section 4.2.13 before issuing the request. The peer MUST send separate PAC TLVs for each type of PAC it wants to be provisioned. Multiple PAC TLVs can be sent in the same packet or in different packets. The EAP server will send the PACs after its internal policy has been satisfied, or it MAY ignore the request or request additional authentications if its policy dictates. The server MAY cache the request and provision the PACs requested after all of its internal policies have been satisfied. If a peer receives a PAC with an unknown type, it MUST ignore it.

A PAC TLV containing a PAC-Acknowledge attribute MUST be sent by the peer to acknowledge the receipt of the Tunnel PAC. A PAC TLV containing a PAC-Acknowledge attribute MUST NOT be used by the peer to acknowledge the receipt of other types of PACs. If the peer receives a PAC TLV with an unknown attribute, it SHOULD ignore the unknown attribute.

3.8.2. Certificate Provisioning within the Tunnel

Provisioning of a peer's certificate is supported in TEAP by performing the Simple PKI Request/Response from [RFC5272] using PKCS#10 and PKCS#7 TLVs, respectively. A peer sends the Simple PKI Request using a PKCS#10 CertificateRequest [RFC2986] encoded into the body of a PKCS#10 TLV (see Section 4.2.17). The TEAP server issues a Simple PKI Response using a PKCS#7 [RFC2315] degenerate "Certificates Only" message encoded into the body of a PKCS#7 TLV (see Section 4.2.16), only after an authentication method has run and provided an identity proof on the peer prior to a certificate is being issued.

In order to provide linking identity and proof-of-possession by including information specific to the current authenticated TLS session within the signed certification request, the peer generating the request SHOULD obtain the tls-unique value from the TLS subsystem as defined in "Channel Bindings for TLS" [RFC5929]. The TEAP peer operations between obtaining the tls_unique value through generation of the Certification Signing Request (CSR) that contains the current tls_unique value and the subsequent verification of this value by the TEAP server are the "phases of the application protocol during which application-layer authentication occurs" that are protected by the synchronization interoperability mechanism described in the interoperability note in "Channel Bindings for TLS" ([RFC5929], Section 3.1). When performing renegotiation, TLS "secure_renegotiation" [RFC5746] MUST be used.

The tls-unique value is base-64-encoded as specified in Section 4 of [RFC4648], and the resulting string is placed in the certification request challengePassword field ([RFC2985], Section 5.4.1). The challengePassword field is limited to 255 octets (Section 7.4.9 of [RFC5246] indicates that no existing ciphersuite would result in an issue with this limitation). If tls-unique information is not embedded within the certification request, the challengePassword field MUST be empty to indicate that the peer did not include the optional channel-binding information (any value submitted is verified by the server as tls-unique information).

The server SHOULD verify the tls-unique information. This ensures that the authenticated TEAP peer is in possession of the private key used to sign the certification request.

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

3.8.3. Server Unauthenticated Provisioning Mode

In Server Unauthenticated Provisioning Mode, an unauthenticated tunnel is established in Phase 1, and the peer and server negotiate an EAP method in Phase 2 that supports mutual authentication and key derivation that is resistant to attacks such as man-in-the-middle and dictionary attacks. This provisioning mode enables the bootstrapping of peers when the peer lacks the ability to authenticate the server during Phase 1. This includes both cases in which the ciphersuite negotiated does not provide authentication and in which the ciphersuite negotiated provides the authentication but the peer is unable to validate the identity of the server for some reason.

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

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

3.8.4. Channel Binding

[RFC6677] defines EAP channel bindings to solve the "lying NAS" and the "lying provider" problems, using a process in which the EAP peer gives information about the characteristics of the service provided by the authenticator to the Authentication, Authorization, and Accounting (AAA) server protected within the EAP method. This allows the server to verify the authenticator is providing information to

the peer that is consistent with the information received from this authenticator as well as the information stored about this authenticator.

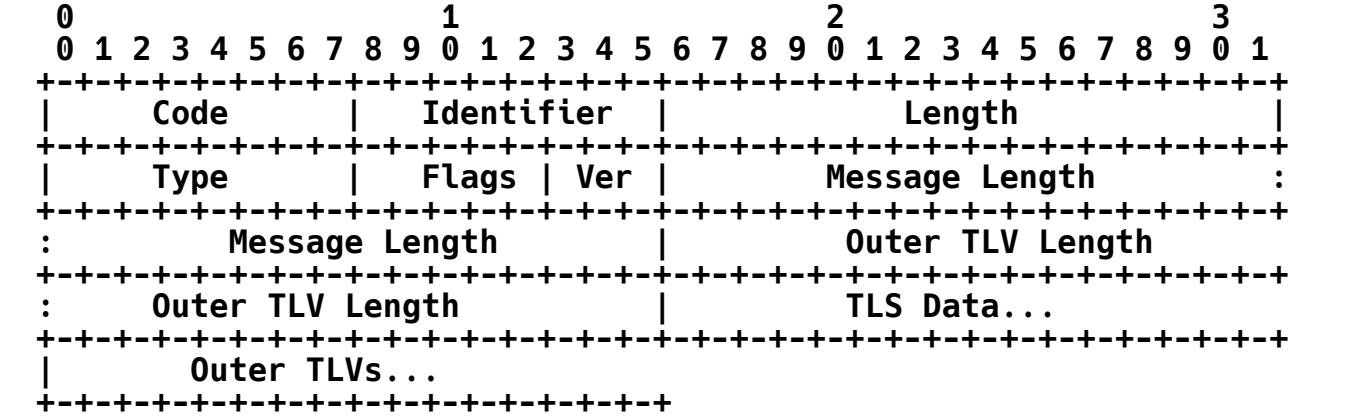
TEAP supports EAP channel binding using the Channel-Binding TLV defined in Section 4.2.7. If the TEAP server wants to request the channel-binding information from the peer, it sends an empty Channel-Binding TLV to indicate the request. The peer responds to the request by sending a Channel-Binding TLV containing a channel-binding message as defined in [RFC6677]. The server validates the channel-binding message and sends back a Channel-Binding TLV with a result code. If the server didn't initiate the channel-binding request and the peer still wants to send the channel-binding information to the server, it can do that by using the Request-Action TLV along with the Channel-Binding TLV. The peer MUST only send channel-binding information after it has successfully authenticated the server and established the protected tunnel.

4. Message Formats

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

4.1. TEAP Message Format

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



Code

The Code field is one octet in length and is defined as follows:

1 Request

2 Response

Identifier

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

Length

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

Type

55 for TEAP

Flags

L Length included; set to indicate the presence of the four-octet Message Length field. It MUST be present for the first fragment of a fragmented message. It MUST NOT be present for any other message.

M More fragments; set on all but the last fragment.

S TEAP start; set in a TEAP Start message sent from the server to the peer.

O Outer TLV length included; set to indicate the presence of the four-octet Outer TLV Length field. It MUST be present only in the initial request and response messages. If the initial message is fragmented, then it MUST be present only on the first fragment.

R Reserved (MUST be zero and ignored upon receipt)

Ver

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

Message Length

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

Outer TLV Length

The Outer TLV Length field is four octets and is present only if the O bit is set. This field provides the total length of the Outer TLVs if present.

TLS Data

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

Outer TLVs

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

4.2. TEAP TLV Format and Support

The TLVs defined here are TLV objects. The TLV objects could be used to carry arbitrary parameters between an EAP peer and EAP server within the protected TLS tunnel.

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

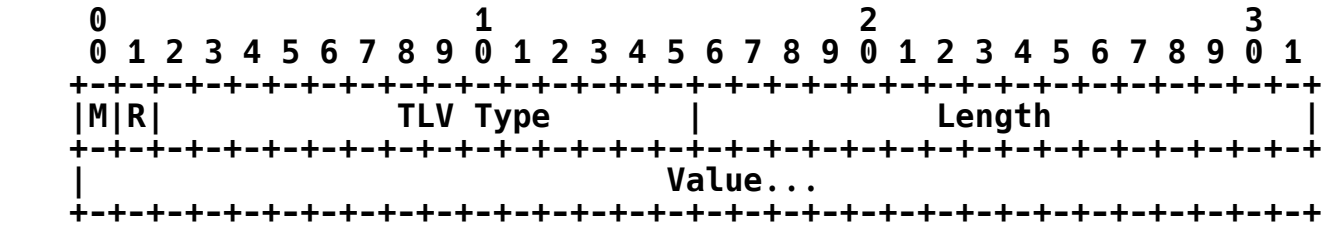
Note that a peer or server may support a TLV with the mandatory bit set but may not understand the contents. The appropriate response to a supported TLV with content that is not understood is defined by the individual TLV specification.

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

- Authority-ID TLV
- Identity-Type TLV
- Result TLV
- NAK TLV
- Error TLV
- Request-Action TLV
- EAP-Payload TLV
- Intermediate-Result TLV
- Crypto-Binding TLV
- Basic-Password-Auth-Req TLV
- Basic-Password-Auth-Resp TLV

4.2.1. General TLV Format

TLVs are defined as described below. The fields are transmitted from left to right.



M

0 Optional TLV

1 Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

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

0 Unassigned

1 Authority-ID TLV (Section 4.2.2)

2 Identity-Type TLV (Section 4.2.3)

3 Result TLV (Section 4.2.4)

4 NAK TLV (Section 4.2.5)

5 Error TLV (Section 4.2.6)

6 Channel-Binding TLV (Section 4.2.7)

7 Vendor-Specific TLV (Section 4.2.8)

8 Request-Action TLV (Section 4.2.9)

9 EAP-Payload TLV (Section 4.2.10)

10 Intermediate-Result TLV (Section 4.2.11)

11 PAC TLV (Section 4.2.12)

12 Crypto-Binding TLV (Section 4.2.13)

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

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

15 PKCS#7 TLV (Section 4.2.16)

16 PKCS#10 TLV (Section 4.2.17)

17 Trusted-Server-Root TLV (Section 4.2.18)

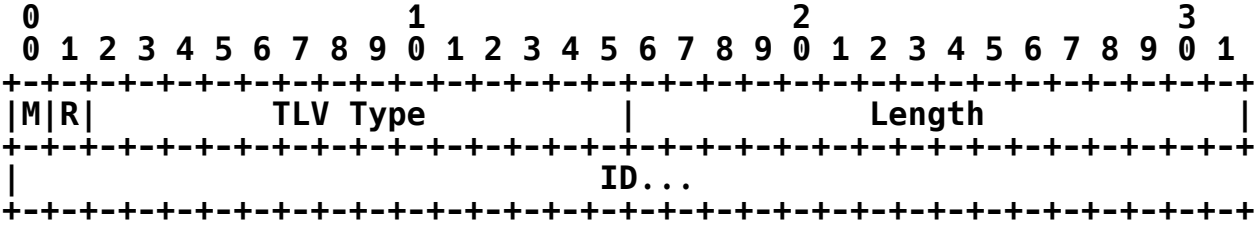
Length

The length of the Value field in octets.

Value

The value of the TLV.

4.2.2. Authority-ID TLV



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

1 - Authority-ID

Length

The Length field is two octets and contains the length of the ID field in octets.

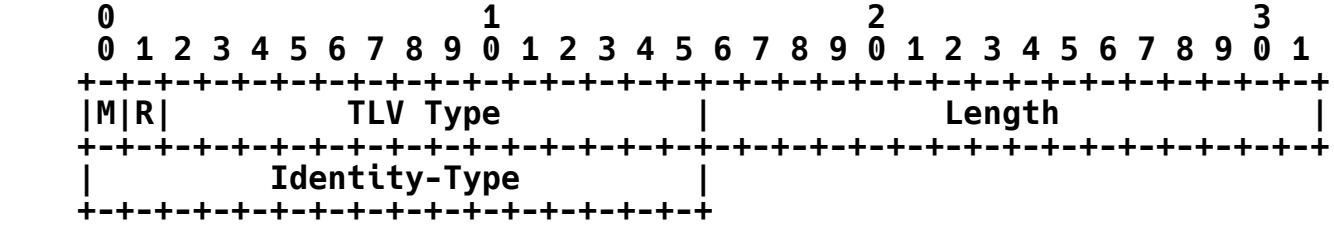
ID

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

4.2.3. Identity-Type TLV

The Identity-Type TLV allows an EAP server to send a hint to help the EAP peer select the right type of identity, for example, user or machine. TEAPv1 implementations MUST support this TLV. Only one Identity-Type TLV SHOULD be present in the TEAP request or response packet. The Identity-Type TLV request MUST come with an EAP-Payload TLV or Basic-Password-Auth-Req TLV. If the EAP peer does have an identity corresponding to the identity type requested, then the peer SHOULD respond with an Identity-Type TLV with the requested type. If the Identity-Type field does not contain one of the known values or if the EAP peer does not have an identity corresponding to the identity type requested, then the peer SHOULD respond with an Identity-Type TLV with the one of available identity types. If the server receives an identity type in the response that does not match the requested type, then the peer does not possess the requested credential type, and the server SHOULD proceed with authentication for the credential type proposed by the peer, proceed with requesting another credential type, or simply apply the network policy based on the configured policy, e.g., sending Result TLV with Failure.

The Identity-Type TLV is defined as follows:



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

2 - Identity-Type TLV

Length

2

Identity-Type

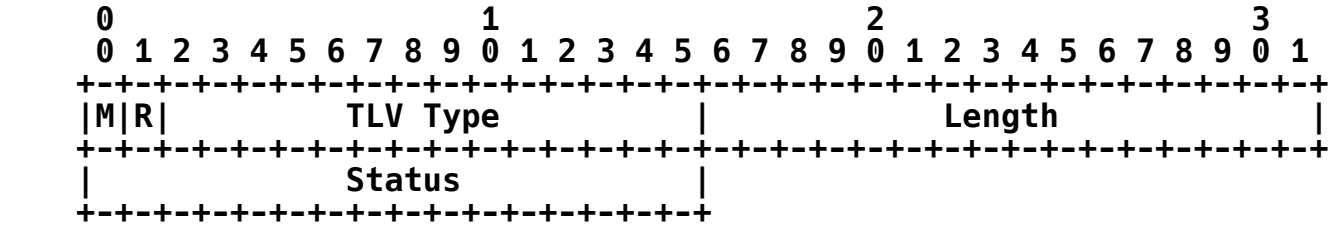
The Identity-Type field is two octets. Values include:

1 User

2 Machine

4.2.4. Result TLV

The Result TLV provides support for acknowledged success and failure messages for protected termination within TEAP. If the Status field does not contain one of the known values, then the peer or EAP server MUST treat this as a fatal error of Unexpected TLVs Exchanged. The behavior of the Result TLV is further discussed in Sections 3.3.3 and 3.6.3. A Result TLV indicating failure MUST NOT be accompanied by the following TLVs: NAK, EAP-Payload TLV, or Crypto-Binding TLV. The Result TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

3 - Result TLV

Length

2

Status

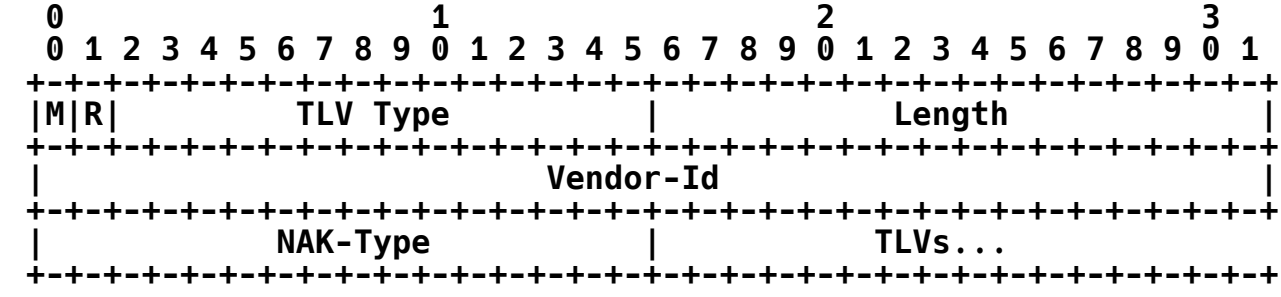
The Status field is two octets. Values include:

1 Success

2 Failure

4.2.5. NAK TLV

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



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

4 - NAK TLV

Length

>=6

Vendor-Id

The Vendor-Id field is four octets and contains the Vendor-Id of the TLV that was not supported. The high-order octet is 0, and the low-order three octets are the Structure of Management

Information (SMI) Network Management Private Enterprise Number of the Vendor in network byte order. The Vendor-Id field MUST be zero for TLVs that are not Vendor-Specific TLVs.

NAK-Type

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

TLVs

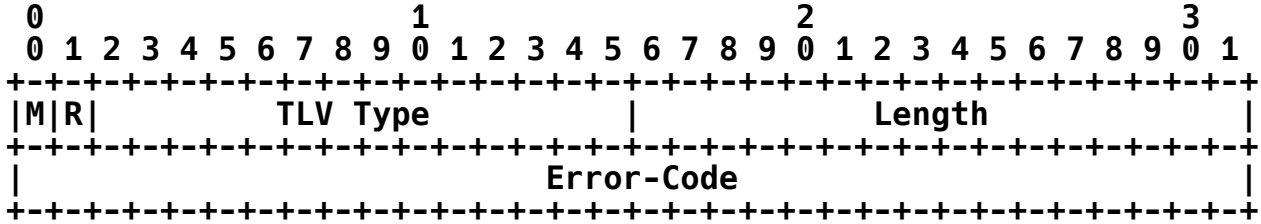
This field contains a list of zero or more TLVs, each of which MUST NOT have the mandatory bit set. These optional TLVs are for future extensibility to communicate why the offending TLV was determined to be unsupported.

4.2.6. Error TLV

The Error TLV allows an EAP peer or server to indicate errors to the other party. A TEAP packet can contain 0 or more Error TLVs. The Error-Code field describes the type of error. Error codes 1-999 represent successful outcomes (informative messages), 1000-1999 represent warnings, and 2000-2999 represent fatal errors. A fatal Error TLV MUST be accompanied by a Result TLV indicating failure, and the conversation is terminated as described in Section 3.6.3.

Many of the error codes below refer to errors in inner method processing that may be retrieved if made available by the inner method. Implementations MUST take care that error messages do not reveal too much information to an attacker. For example, the usage of error message 1031 (User account credentials incorrect) is NOT RECOMMENDED, because it allows an attacker to determine valid usernames by differentiating this response from other responses. It should only be used for troubleshooting purposes.

The Error TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

5 - Error TLV

Length

4

Error-Code

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

1	User account expires soon
2	User account credential expires soon
3	User account authorizations change soon
4	Clock skew detected
5	Contact administrator
6	User account credentials change required
1001	Inner Method Error
1002	Unspecified authentication infrastructure problem
1003	Unspecified authentication failure
1004	Unspecified authorization failure
1005	User account credentials unavailable
1006	User account expired
1007	User account locked: try again later
1008	User account locked: admin intervention required

1009 Authentication infrastructure unavailable

1010 Authentication infrastructure not trusted

1011 Clock skew too great

1012 Invalid inner realm

1013 Token out of sync: administrator intervention required

1014 Token out of sync: PIN change required

1015 Token revoked

1016 Tokens exhausted

1017 Challenge expired

1018 Challenge algorithm mismatch

1019 Client certificate not supplied

1020 Client certificate rejected

1021 Realm mismatch between inner and outer identity

1022 Unsupported Algorithm In Certificate Signing Request

1023 Unsupported Extension In Certificate Signing Request

1024 Bad Identity In Certificate Signing Request

1025 Bad Certificate Signing Request

1026 Internal CA Error

1027 General PKI Error

1028 Inner method's channel-binding data required but not supplied

1029 Inner method's channel-binding data did not include required information

1030 Inner method's channel binding failed

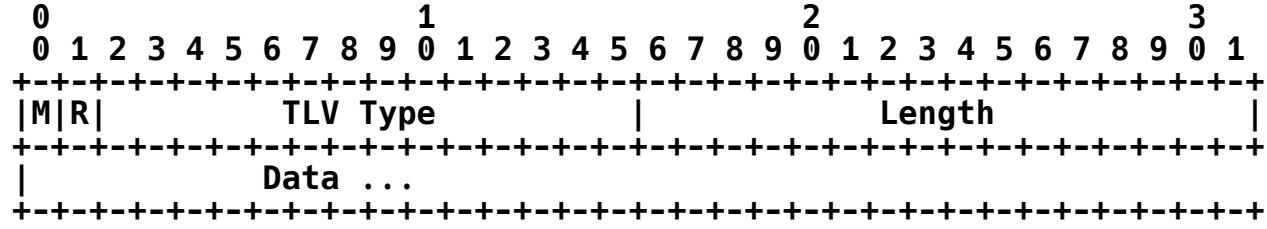
1031 User account credentials incorrect [USAGE NOT RECOMMENDED]

2001 Tunnel Compromise Error

2002 Unexpected TLVs Exchanged

4.2.7. Channel-Binding TLV

The Channel-Binding TLV provides a mechanism for carrying channel-binding data from the peer to the EAP server and a channel-binding response from the EAP server to the peer as described in [RFC6677]. TEAPv1 implementations MAY support this TLV, which cannot be responded to with a NAK TLV. If the Channel-Binding data field does not contain one of the known values or if the EAP server does not support this TLV, then the server MUST ignore the value. The Channel-Binding TLV is defined as follows:



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

6 - Channel-Binding TLV

Length

variable

Data

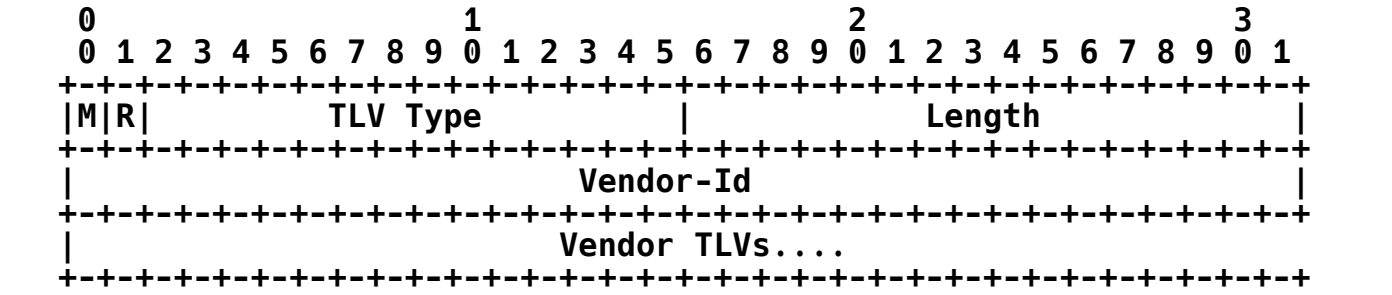
The data field contains a channel-binding message as defined in Section 5.3 of [RFC6677].

4.2.8. Vendor-Specific TLV

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

Vendor TLVs may be optional or mandatory. Vendor TLVs sent with Result TLVs MUST be marked as optional. If the Vendor-Specific TLV is marked as mandatory, then it is expected that the receiving side needs to recognize the vendor ID, parse all Vendor TLVs within, and deal with error handling within the Vendor-Specific TLV as defined by the vendor.

The Vendor-Specific TLV is defined as follows:



M

0 or 1

R

Reserved, set to zero (0)

TLV Type

7 - Vendor-Specific TLV

Length

4 + cumulative length of all included Vendor TLVs

Vendor-Id

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

Vendor TLVs

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

4.2.9. Request-Action TLV

The Request-Action TLV MAY be sent by both the peer and the server in response to a successful or failed Result TLV. It allows the peer or server to request the other side to negotiate additional EAP methods or process TLVs specified in the response packet. The receiving side MUST process this TLV. The processing for the TLV is as follows:

The receiving entity MAY choose to process any of the TLVs that are included in the message.

If the receiving entity chooses NOT to process any TLV in the list, then it sends back a Result TLV with the same code in the Status field of the Request-Action TLV.

If multiple Request-Action TLVs are in the request, the session can continue if any of the TLVs in any Request-Action TLV are processed.

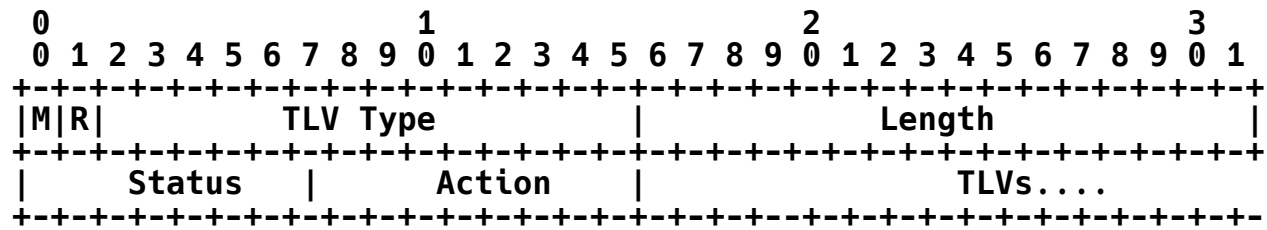
If multiple Request-Action TLVs are in the request and none of them is processed, then the most fatal status should be used in the Result TLV returned. If a status code in the Request-Action TLV is not understood by the receiving entity, then it should be treated as a fatal error.

After processing the TLVs or EAP method in the request, another round of Result TLV exchange would occur to synchronize the final status on both sides.

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

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

The Request-Action TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

8 - Request-Action TLV

Length

2 + cumulative length of all included TLVs

Status

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

1 Success

2 Failure

Action

The Action field is one octet. Values include:

1 Process-TLV

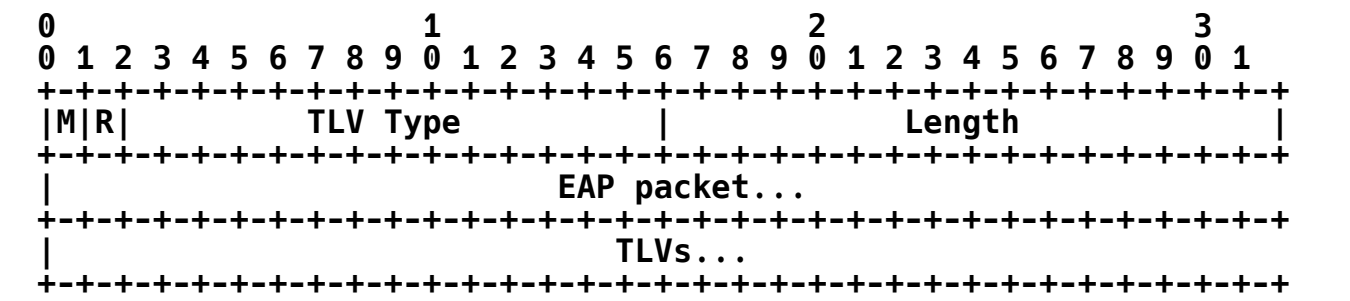
2 Negotiate-EAP

TLVs

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

4.2.10. EAP-Payload TLV

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



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

9 - EAP-Payload TLV

Length

length of embedded EAP packet + cumulative length of additional TLVs

EAP packet

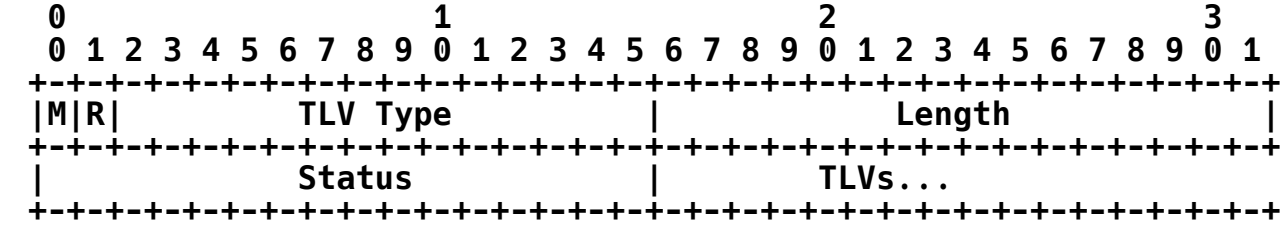
This field contains a complete EAP packet, including the EAP header (Code, Identifier, Length, Type) fields. The length of this field is determined by the Length field of the encapsulated EAP packet.

TLVs

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

4.2.11. Intermediate-Result TLV

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



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

10 - Intermediate-Result TLV

Length

2 + cumulative length of the embedded associated TLVs

Status

The Status field is two octets. Values include:

1 Success

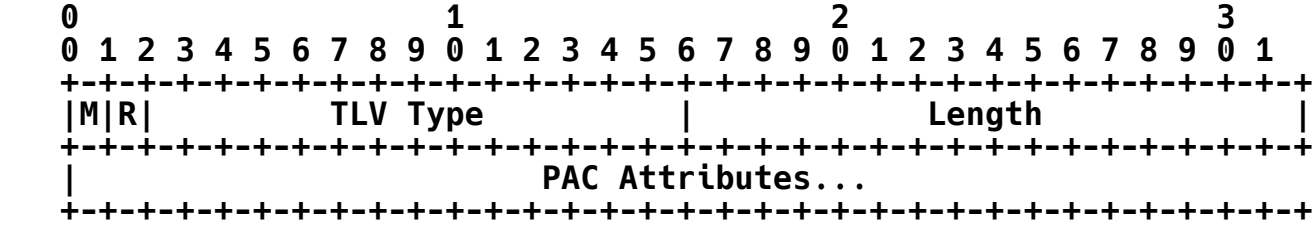
2 Failure

TLVs

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

4.2.12. PAC TLV Format

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



M

0 or 1

R

Reserved, set to zero (0)

TLV Type

11 - PAC TLV

Length

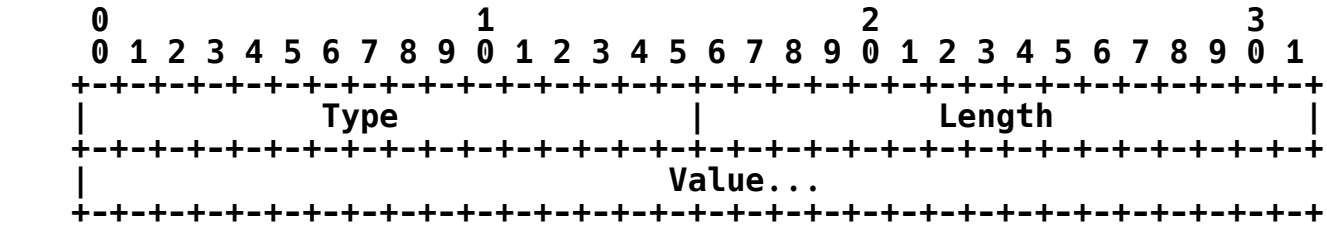
Two octets containing the length of the PAC Attributes field in octets.

PAC Attributes

A list of PAC attributes in the TLV format.

4.2.12.1. Formats for PAC Attributes

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



Type

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

1 - PAC-Key

2 - PAC-Opaque

3 - PAC-Lifetime

4 - A-ID

5 - I-ID

6 - Reserved

7 - A-ID-Info

8 - PAC-Acknowledgement

9 - PAC-Info

10 - PAC-Type

Length

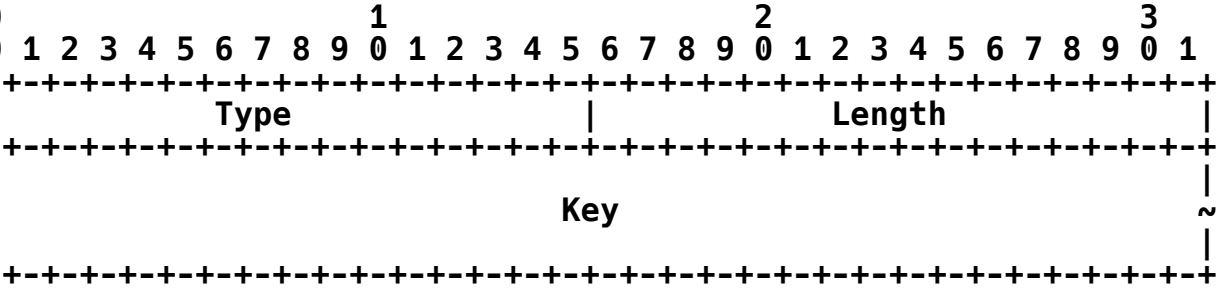
Two octets containing the length of the Value field in octets.

Value

The value of the PAC attribute.

4.2.12.2. PAC-Key

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



Type

1 - PAC-Key

Length

2-octet length indicating the length of the key.

Key

The value of the PAC-Key.

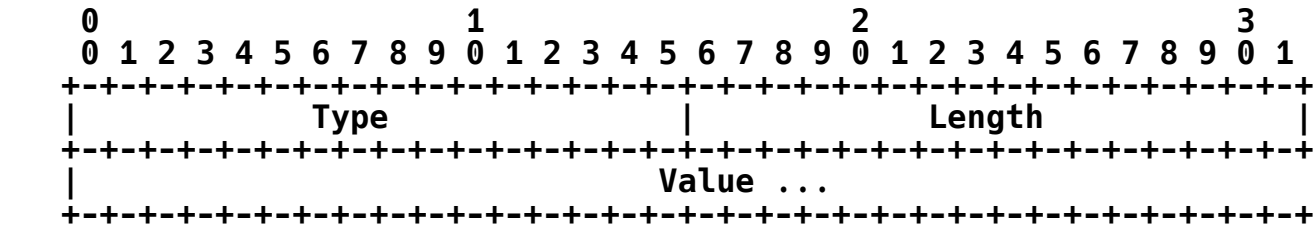
4.2.12.3. PAC-Opaque

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

The PAC-Opaque is opaque to the peer, and thus the peer MUST NOT attempt to interpret it. A peer that has been issued a PAC-Opaque by a server stores that data and presents it back to the server according to its PAC-Type. The Tunnel PAC is used in the ClientHello SessionTicket extension field defined in [RFC5077]. If a peer has opaque data issued to it by multiple servers, then it stores the data issued by each server separately according to the A-ID. This requirement allows the peer to maintain and use each opaque datum as an independent PAC pairing, with a PAC-Key mapping to a PAC-Opaque identified by the A-ID. As there is a one-to-one correspondence between the PAC-Key and PAC-Opaque, the peer determines the PAC-Key

and corresponding PAC-Opaque based on the A-ID provided in the TEAP/Start message and the A-ID provided in the PAC-Info when it was provisioned with a PAC-Opaque.

The PAC-Opaque attribute format is summarized as follows:



Type

2 - PAC-Opaque

Length

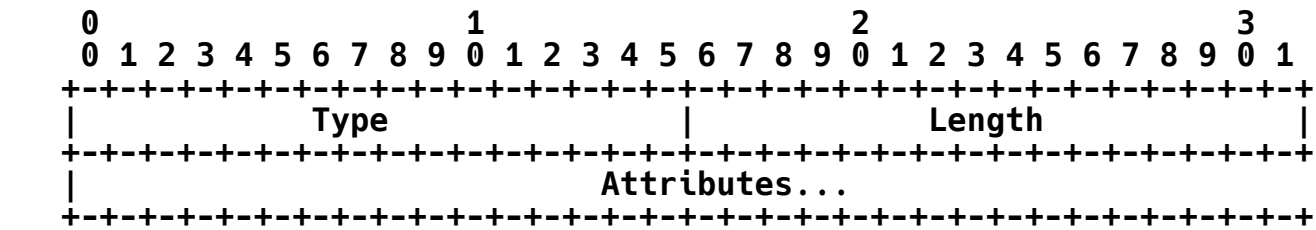
The Length field is two octets, which contains the length of the Value field in octets.

Value

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

4.2.12.4. PAC-Info

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



Type

9 - PAC-Info

Length

2-octet field containing the length of the Attributes field in octets.

Attributes

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

3 - PAC-Lifetime

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

4 - A-ID

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

5 - I-ID

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

is assumed that the Tunnel PAC can be used for multiple users and the peer will not enforce the unique-Tunnel-PAC-per-user policy.

7 - A-ID-Info

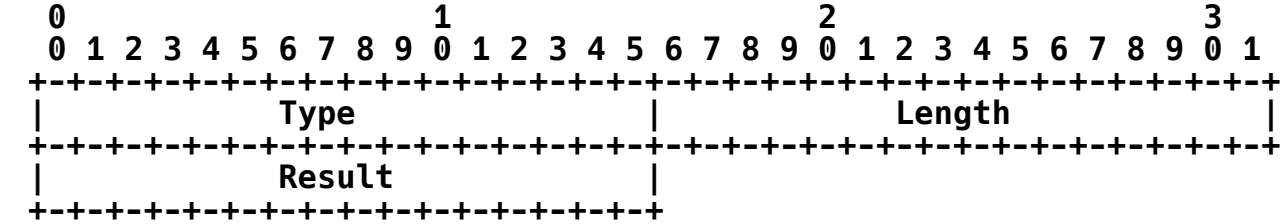
Authority Identifier Information is intended to provide a user-friendly name for the A-ID. It may contain the enterprise name and server name in a human-readable format. This TLV serves as an aid to the peer to better inform the end user about the A-ID. The name is encoded in UTF-8 [RFC3629] format.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

10 - PAC-Type

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

4.2.12.5. PAC-Acknowledgement TLV

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



Type

8 - PAC-Acknowledgement

Length

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

Result

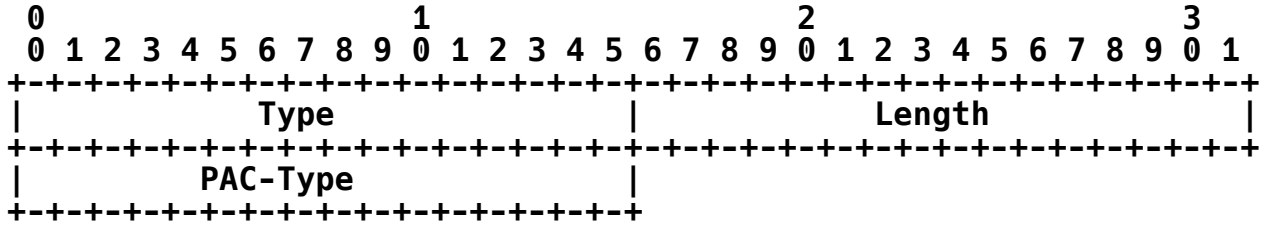
The resulting value MUST be one of the following:

1 - Success

2 - Failure

4.2.12.6. PAC-Type TLV

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



Type

10 - PAC-Type

Length

2-octet field with a value of 2.

PAC-Type

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

1 - Tunnel PAC

4.2.13. Crypto-Binding TLV

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

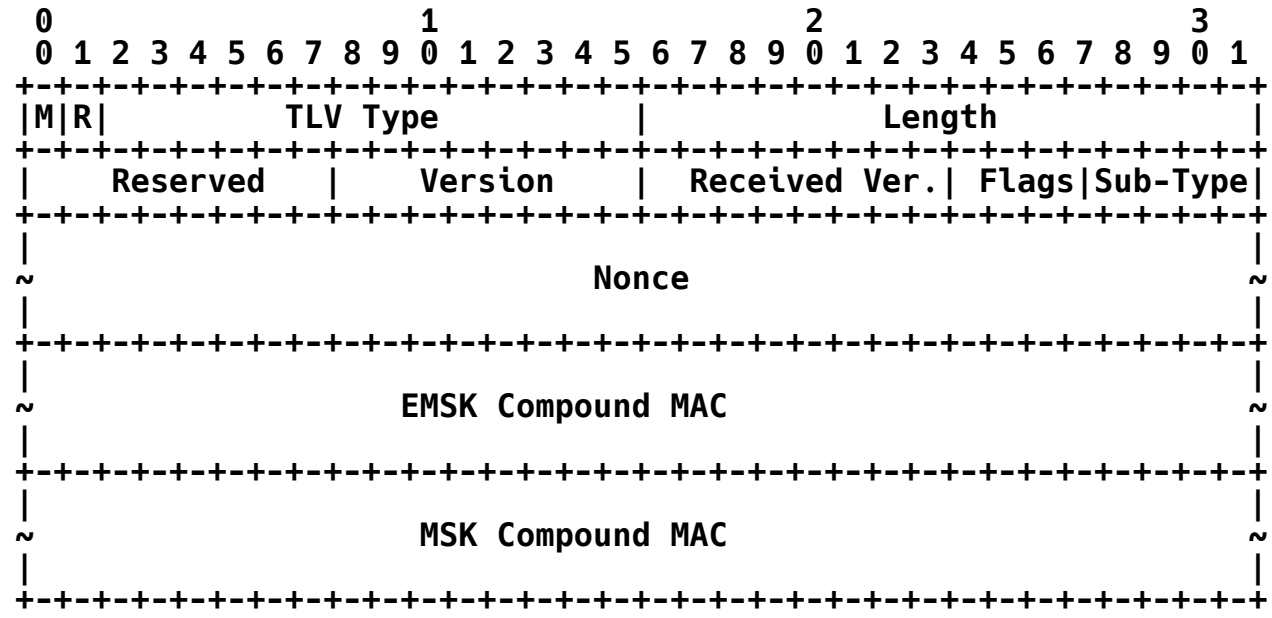
The Crypto-Binding TLV MUST be exchanged and verified before the final Result TLV exchange, regardless of whether there is an inner EAP method authentication or not. It MUST be included with the Intermediate-Result TLV to perform cryptographic binding after each successful EAP method in a sequence of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

- The Crypto-Binding TLV version is supported.

- The MAC verifies correctly.
- The received version in the Crypto-Binding TLV matches the version sent by the receiver during the EAP version negotiation.
- The subtype is set to the correct value.

If any of the above checks fails, then the TLV is invalid. An invalid Crypto-Binding TLV is a fatal error and is handled as described in Section 3.6.3

The Crypto-Binding TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

12 - Crypto-Binding TLV

Length

76

Reserved

Reserved, set to zero (0)

Version

The Version field is a single octet, which is set to the version of Crypto-Binding TLV the TEAP method is using. For an implementation compliant with this version of TEAP, the version number MUST be set to one (1).

Received Ver

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

Flags

The Flags field is four bits. Defined values include

1 EMSK Compound MAC is present

2 MSK Compound MAC is present

3 Both EMSK and MSK Compound MAC are present

Sub-Type

The Sub-Type field is four bits. Defined values include

0 Binding Request

1 Binding Response

Nonce

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

EMSK Compound MAC

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

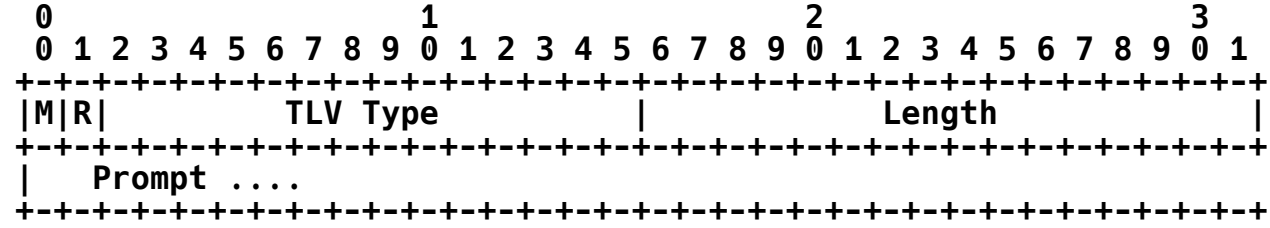
MSK Compound MAC

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

4.2.14. Basic-Password-Auth-Req TLV

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

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



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

13 - Basic-Password-Auth-Req TLV

Length

variable

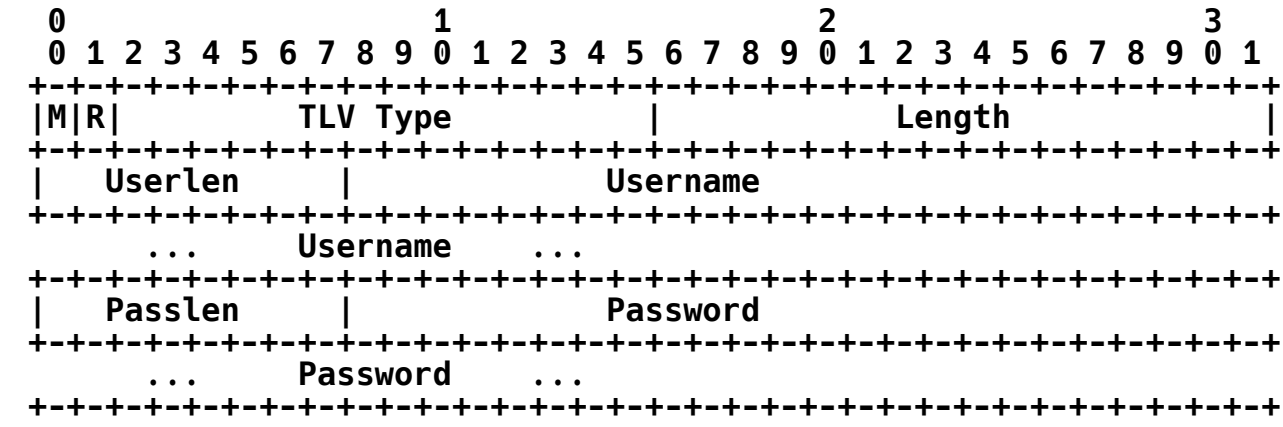
Prompt

optional user prompt message in UTF-8 [RFC3629] format

4.2.15. Basic-Password-Auth-Resp TLV

The Basic-Password-Auth-Resp TLV is used by the peer to respond to a Basic-Password-Auth-Req TLV with a username and password. The TLV contains a username and password. The username and password are in UTF-8 [RFC3629] format.

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



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

14 - Basic-Password-Auth-Resp TLV

Length

variable

Userlen

Length of Username field in octets

Username

Username in UTF-8 [RFC3629] format

Passlen

Length of Password field in octets

Password

Password in UTF-8 [RFC3629] format

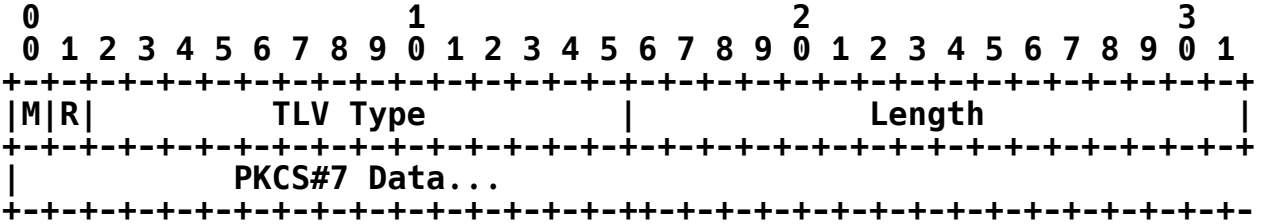
4.2.16. PKCS#7 TLV

The PKCS#7 TLV is used by the EAP server to deliver certificate(s) to the peer. The format consists of a certificate or certificate chain in binary DER encoding [X.690] in a degenerate Certificates Only PKCS#7 SignedData Content as defined in [RFC5652].

When used in response to a Trusted-Server-Root TLV request from the peer, the EAP server MUST send the PKCS#7 TLV inside a Trusted-Server-Root TLV. When used in response to a PKCS#10 certificate enrollment request from the peer, the EAP server MUST send the PKCS#7 TLV without a Trusted-Server-Root TLV. The PKCS#7 TLV is always marked as optional, which cannot be responded to with a NAK TLV. TEAP implementations that support the Trusted-Server-Root TLV or the PKCS#10 TLV MUST support this TLV. Peers MUST NOT assume that the certificates in a PKCS#7 TLV are in any order.

TEAP servers MAY return self-signed certificates. Peers that handle self-signed certificates or trust anchors MUST NOT implicitly trust these certificates merely due to their presence in the certificate bag. Note: Peers are advised to take great care in deciding whether to use a received certificate as a trust anchor. The authenticated nature of the tunnel in which a PKCS#7 bag is received can provide a level of authenticity to the certificates contained therein. Peers are advised to take into account the implied authority of the EAP server and to constrain the trust it can achieve through the trust anchor received in a PKCS#7 TLV.

The PKCS#7 TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

15 - PKCS#7 TLV

Length

The length of the PKCS#7 Data field.

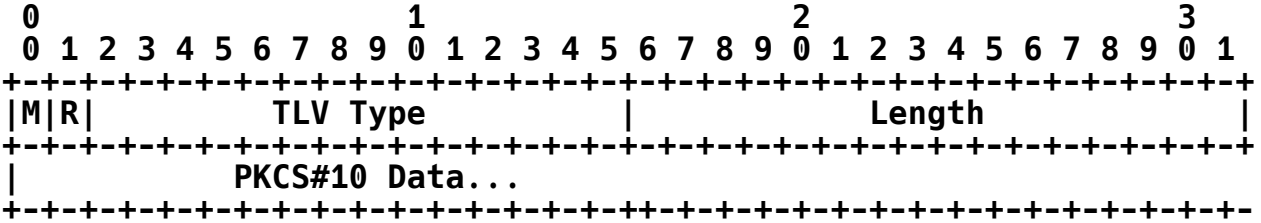
PKCS#7 Data

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

4.2.17. PKCS#10 TLV

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

The PKCS#10 TLV is defined as follows:



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

16 - PKCS#10 TLV

Length

The length of the PKCS#10 Data field.

PKCS#10 Data

This field contains the DER-encoded PKCS#10 certificate request.

4.2.18. Trusted-Server-Root TLV

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

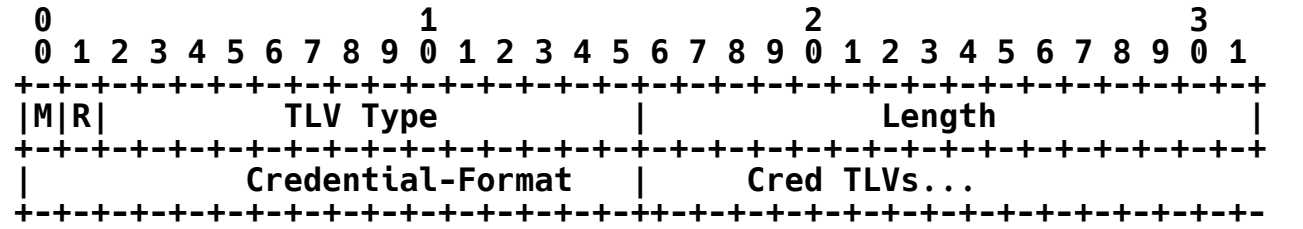
After the peer has determined that it has successfully authenticated the EAP server and validated the Crypto-Binding TLV, it MAY send one or more Trusted-Server-Root TLVs (marked as optional) to request the trusted server root certificates from the EAP server. The EAP server MAY send one or more root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

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

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

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

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



M

0 - Optional TLV

R

Reserved, set to zero (0)

TLV Type

17 - Trusted-Server-Root TLV

Length

>=2 octets

Credential-Format

The Credential-Format field is two octets. Values include:

1 - PKCS#7-Server-Certificate-Root

Cred TLVs

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

4.3. TLV Rules

To save round trips, multiple TLVs can be sent in a single TEAP packet. However, multiple EAP Payload TLVs, multiple Basic Password Authentication TLVs, or an EAP Payload TLV with a Basic Password Authentication TLV within one single TEAP packet is not supported in this version and MUST NOT be sent. If the peer or EAP server

receives multiple EAP Payload TLVs, then it MUST terminate the connection with the Result TLV. The order of TLVs in TEAP does not matter, except one should always process the Identity-Type TLV before processing the EAP TLV or Basic Password Authentication TLV as the Identity-Type TLV is a hint to the type of identity that is to be authenticated.

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

0 This TLV MUST NOT be present in the message.

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

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

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

4.3.1. Outer TLVs

The following table provides a guide to which TLVs may be included in the TEAP packet outside the TLS channel, which kind of packets, and in what quantity:

Request	Response	Success	Failure	TLVs
0-1	0	0	0	Authority-ID
0-1	0-1	0	0	Identity-Type
0+	0+	0	0	Vendor-Specific

Outer TLVs MUST be marked as optional. Vendor-TLVs inside Vendor-Specific TLV MUST be marked as optional when included in Outer TLVs. Outer TLVs MUST NOT be included in messages after the first two TEAP messages sent by peer and EAP-server respectively. That is the first EAP-server-to-peer message and first peer-to-EAP-server message. If the message is fragmented, the whole set of messages is counted as one message. If Outer TLVs are included in messages after the first two TEAP messages, they MUST be ignored.

4.3.2. Inner TLVs

The following table provides a guide to which Inner TLVs may be encapsulated in TLS in TEAP Phase 2, in which kind of packets, and in what quantity. The messages are as follows: Request is a TEAP Request, Response is a TEAP Response, Success is a message containing a successful Result TLV, and Failure is a message containing a failed Result TLV.

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

NOTE: Vendor TLVs (included in Vendor-Specific TLVs) sent with a Result TLV MUST be marked as optional.

5. Cryptographic Calculations

For key derivation and crypto-binding, TEAP uses the Pseudorandom Function (PRF) and MAC algorithms negotiated in the underlying TLS session. Since these algorithms depend on the TLS version and ciphersuite, TEAP implementations need a mechanism to determine the version and ciphersuite in use for a particular session. The implementation can then use this information to determine which PRF and MAC algorithm to use.

5.1. TEAP Authentication Phase 1: Key Derivations

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

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

The session_key_seed is used by the TEAP authentication Phase 2 conversation to both cryptographically bind the inner method(s) to the tunnel as well as generate the resulting TEAP session keys. The other TLS keying materials are derived and used as defined in [RFC5246].

5.2. Intermediate Compound Key Derivations

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

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

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

> IMSK = First 32 octets of TLS-PRF(EMSK, "TEAPbindkey@ietf.org" | "\0" | 64)
>
> where "|" denotes concatenation, EMSK is the EMSK from the inner method, "TEAPbindkey@ietf.org" consists the ASCII value for the label "TEAPbindkey@ietf.org" (without quotes), "\0" = is a NULL octet (0x00 in hex), length is the 2-octet unsigned integer in network byte order, and TLS-PRF is the PRF negotiated as part of TLS handshake [RFC5246].

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

However, it's possible that the peer and server sides might not have the same capability to export EMSK. In order to maintain maximum flexibility while prevent downgrading attack, the following mechanism is in place.

On the sender of the Crypto-Binding TLV side:

> If the EMSK is not available, then the sender computes the Compound MAC using the MSK of the inner method.

If the EMSK is available and the sender's policy accepts MSK-based MAC, then the sender computes two Compound MAC values. The first is computed with the EMSK. The second one is computed using the MSK. Both MACs are then sent to the other side.

If the EMSK is available but the sender's policy does not allow downgrading to MSK-generated MAC, then the sender SHOULD only send EMSK-based MAC.

On the receiver of the Crypto-Binding TLV side:

If the EMSK is not available and an MSK-based Compound MAC was sent, then the receiver validates the Compound MAC and sends back an MSK-based Compound MAC response.

If the EMSK is not available and no MSK-based Compound MAC was sent, then the receiver handles like an invalid Crypto-Binding TLV with a fatal error.

If the EMSK is available and an EMSK-based Compound MAC was sent, then the receiver validates it and creates a response Compound MAC using the EMSK.

If the EMSK is available but no EMSK-based Compound MAC was sent and its policy accepts MSK-based MAC, then the receiver validates it using the MSK and, if successful, generates and returns an MSK-based Compound MAC.

If the EMSK is available but no EMSK Compound MAC was sent and its policy does not accept MSK-based MAC, then the receiver handles like an invalid Crypto-Binding TLV with a fatal error.

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

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

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

5.3. Computing the Compound MAC

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

The Compound MAC computation is as follows:

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

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

1. The entire Crypto-Binding TLV attribute with both the EMSK and MSK Compound MAC fields zeroed out.

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

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

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

5.4. EAP Master Session Key Generation

TEAP authentication assures the Master Session Key (MSK) and Extended Master Session Key (EMSK) output from the EAP method are the result of all authentication conversations by generating an Intermediate Compound Key (IMCK). The IMCK is mutually derived by the peer and the server as described in Section 5.2 by combining the MSKs from

inner EAP methods with key material from TEAP Phase 1. The resulting MSK and EMSK are generated as part of the IMCKn key hierarchy as follows:

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

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

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

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

6. IANA Considerations

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

The EAP Method Type number 55 has been assigned for TEAP.

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

0 Unassigned

1 Authority-ID TLV

2 Identity-Type TLV

3 Result TLV

4 NAK TLV

5 Error TLV

6 Channel-Binding TLV

7 Vendor-Specific TLV

8 Request-Action TLV

9 EAP-Payload TLV

10 Intermediate-Result TLV

11 PAC TLV

12 Crypto-Binding TLV

13 Basic-Password-Auth-Req TLV

14 Basic-Password-Auth-Resp TLV

15 PKCS#7 TLV

16 PKCS#10 TLV

17 Trusted-Server-Root TLV

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

1 User

2 Machine

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

1 Success

2 Failure

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

1 User account expires soon

2 User account credential expires soon

3	User account authorizations change soon
4	Clock skew detected
5	Contact administrator
6	User account credentials change required
1001	Inner Method Error
1002	Unspecified authentication infrastructure problem
1003	Unspecified authentication failure
1004	Unspecified authorization failure
1005	User account credentials unavailable
1006	User account expired
1007	User account locked: try again later
1008	User account locked: admin intervention required
1009	Authentication infrastructure unavailable
1010	Authentication infrastructure not trusted
1011	Clock skew too great
1012	Invalid inner realm
1013	Token out of sync: administrator intervention required
1014	Token out of sync: PIN change required
1015	Token revoked
1016	Tokens exhausted
1017	Challenge expired
1018	Challenge algorithm mismatch
1019	Client certificate not supplied
1020	Client certificate rejected

1021 Realm mismatch between inner and outer identity

1022 Unsupported Algorithm In Certificate Signing Request

1023 Unsupported Extension In Certificate Signing Request

1024 Bad Identity In Certificate Signing Request

1025 Bad Certificate Signing Request

1026 Internal CA Error

1027 General PKI Error

1028 Inner method's channel-binding data required but not supplied

1029 Inner method's channel-binding data did not include required information

1030 Inner method's channel binding failed

1031 User account credentials incorrect [USAGE NOT RECOMMENDED]

2001 Tunnel Compromise Error

2002 Unexpected TLVs Exchanged

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

1 Process-TLV

2 Negotiate-EAP

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

1 PAC-Key

2 PAC-Opaque

3 PAC-Lifetime

4 A-ID

5 I-ID

6 Reserved

7 A-ID-Info

8 PAC-Acknowledgement

9 PAC-Info

10 PAC-Type

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

1 Tunnel PAC

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

1 PKCS#7-Server-Certificate-Root

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

TEAP registers the label "EXPORTER: teap session key seed" in the TLS Exporter Label Registry [RFC5705]. This label is used in derivation as defined in Section 5.1.

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

7. Security Considerations

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

7.1. Mutual Authentication and Integrity Protection

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

The Result TLV is protected and conveys the true Success or Failure of TEAP, and it should be used as the indicator of its success or failure respectively. However, as EAP terminates with either 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 SHOULD silently discard those cleartext EAP Success or Failure messages that do not coincide with the status sent in the protected Result TLV.

7.2. Method Negotiation

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

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

7.3. Separation of Phase 1 and Phase 2 Servers

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

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

- Loss of identity protection
- Offline dictionary attacks
- Lack of policy enforcement
- Man-in-the-middle attacks (as described in [RFC7029])

There may be cases where a trust relationship exists between the Phase 1 and Phase 2 servers, such as on a campus or between two offices within the same company, where there is no danger in revealing the inner identity and credentials of the peer to entities between the two servers. In these cases, using a proxy solution without end-to-end protection of TEAP MAY be used. The TEAP encrypting/decrypting gateway MUST, at a minimum, provide support for IPsec, TLS, or similar protection in order to provide confidentiality for the portion of the conversation between the gateway and the EAP server. In addition, separation of the inner and outer method servers allows for crypto-binding based on the inner method MSK to be thwarted as described in [RFC7029]. Implementation and deployment SHOULD adopt various mitigation strategies described in [RFC7029]. If the inner method is deriving EMSK, then this threat is mitigated as TEAP utilizes the mutual crypto-binding based on EMSK as described in [RFC7029].

7.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies

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

- Per-packet confidentiality and integrity protection
- User identity protection
- Better support for notification messages
- Protected EAP inner method negotiation
- Sequencing of EAP methods
- Strong mutually derived MSKs
- Acknowledged success/failure indication

- Faster re-authentications through session resumption
- Mitigation of dictionary attacks
- Mitigation of man-in-the-middle attacks
- Mitigation of some denial-of-service attacks

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

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

7.4.1. User Identity Protection and Verification

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

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

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

7.4.2. Dictionary Attack Resistance

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

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

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

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

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

TEAP crypto binding does not guarantee man-in-the-middle protection if the client allows a connection to an untrusted server, such as in the case where the client does not properly validate the server's certificate. If the TLS ciphersuite derives the master secret solely from the contribution of secret data from one side of the conversation (such as ciphersuites based on RSA key transport), then an attacker who can convince the client to connect and engage in authentication can impersonate the client to another server even if a strong inner method is executed within the tunnel. If the TLS ciphersuite derives the master secret from the contribution of secrets from both sides of the conversation (such as in ciphersuites based on Diffie-Hellman), then crypto binding can detect an attacker in the conversation if a strong inner method is used.

7.4.4. PAC Binding to User Identity

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

7.5. Protecting against Forged Cleartext EAP Packets

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

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

The success/failure decisions within the TEAP tunnel indicate the final decision of the TEAP authentication conversation. After a success/failure result has been indicated by a protected mechanism, the TEAP peer can process unprotected EAP Success and EAP Failure messages; however, the peer MUST ignore any unprotected EAP Success

or Failure messages where the result does not match the result of the protected mechanism.

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

7.6. Server Certificate Validation

As part of the TLS negotiation, the server presents a certificate to the peer. The peer SHOULD verify the validity of the EAP server certificate and SHOULD also examine the EAP server name presented in the certificate in order to determine whether the EAP server can be trusted. When performing server certificate validation, implementations MUST provide support for the rules in [RFC5280] for validating certificates against a known trust anchor. In addition, implementations MUST support matching the realm portion of the peer's NAI against a SubjectAltName of type dNSName within the server certificate. However, in certain deployments, this might not be turned on. Please note that in the case where the EAP authentication is remote, the EAP server will not reside on the same machine as the authenticator, and therefore, the name in the EAP server's certificate cannot be expected to match that of the intended destination. In this case, a more appropriate test might be whether the EAP server's certificate is signed by a certification authority (CA) controlling the intended domain and whether the authenticator can be authorized by a server in that domain.

7.7. Tunnel PAC Considerations

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

also apply to the PAC-Opaque. The PAC-Info may contain information about the Tunnel PAC such as the identity of the PAC issuer and the Tunnel PAC lifetime for use in the management of the Tunnel PAC. The PAC-Info should be securely stored by the peer to protect it from disclosure and modification.

7.8. Security Claims

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

Auth. mechanism:	Certificate-based, shared-secret-based, and various tunneled authentication mechanisms.
Ciphersuite negotiation:	Yes
Mutual authentication:	Yes
Integrity protection:	Yes. Any method executed within the TEAP tunnel is integrity protected. The cleartext EAP headers outside the tunnel are not integrity protected.
Replay protection:	Yes
Confidentiality:	Yes
Key derivation:	Yes
Key strength:	See Note 1 below.
Dictionary attack prot.:	Yes
Fast reconnect:	Yes
Cryptographic binding:	Yes
Session independence:	Yes
Fragmentation:	Yes
Key Hierarchy:	Yes
Channel binding:	Yes

Notes

1. BCP 86 [RFC3766] offers advice on appropriate key sizes. The National Institute for Standards and Technology (NIST) also offers advice on appropriate key sizes in [NIST-SP-800-57]. [RFC3766], Section 5 advises use of the following required RSA or DH (Diffie-Hellman) module and DSA (Digital Signature Algorithm) subgroup size in bits for a given level of attack resistance in bits. Based on the table below, a 2048-bit RSA key is required to provide 112-bit equivalent key strength:

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

8. Acknowledgements

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

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

The method for linking identity and proof-of-possession by placing the tls-unique value in the challengePassword field of the CSR as described in Section 3.8.2 was inspired by the technique described in "Enrollment over Secure Transport" [RFC7030].

Helpful review comments were provided by Russ Housley, Jari Arkko, Ilan Frenkel, Jeremy Steiglitz, Dan Harkins, Sam Hartman, Jim Schaad, Barry Leiba, Stephen Farrell, Chris Lonvick, and Josh Howlett.

9. References

9.1. Normative References

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

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

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

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

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

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

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

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

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

[RFC6677] Hartman, S., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, July 2012.

9.2. Informative References

[IEEE.802-1X.2013]
IEEE, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X, December 2013.

[NIST-SP-800-57]
National Institute of Standards and Technology, "Recommendation for Key Management", NIST Special Publication 800-57, July 2012.

[PEAP] Microsoft Corporation, "[MS-PEAP]: Protected Extensible Authentication Protocol (PEAP)", February 2014.

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

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, November 2000.

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

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

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

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

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

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

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

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4851] Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

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

[RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.

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

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

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

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

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

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

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

[RFC6124] Sheffer, Y., Zorn, G., Tschofenig, H., and S. Fluhrer, "An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol", RFC 6124, February 2011.

[RFC6678] Hoeper, K., Hanna, S., Zhou, H., and J. Salowey, "Requirements for a Tunnel-Based Extensible Authentication Protocol (EAP) Method", RFC 6678, July 2012.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, June 2013.

[RFC7029] Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, October 2013.

[RFC7030] Pritikin, M., Yee, P., and D. Harkins, "Enrollment over Secure Transport", RFC 7030, October 2013.

[X.690] ITU-T, "ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, November 2008.

Appendix A. Evaluation against Tunnel-Based EAP Method Requirements

This section evaluates all tunnel-based EAP method requirements described in [RFC6678] against TEAP version 1.

A.1. Requirement 4.1.1: RFC Compliance

TEAPv1 meets this requirement by being compliant with RFC 3748 [RFC3748], RFC 4017 [RFC4017], RFC 5247 [RFC5247], and RFC 4962 [RFC4962]. It is also compliant with the "cryptographic algorithm agility" requirement by leveraging TLS 1.2 for all cryptographic algorithm negotiation.

A.2. Requirement 4.2.1: TLS Requirements

TEAPv1 meets this requirement by mandating TLS version 1.2 support as defined in Section 3.2.

A.3. Requirement 4.2.1.1.1: Ciphersuite Negotiation

TEAPv1 meets this requirement by using TLS to provide protected ciphersuite negotiation.

A.4. Requirement 4.2.1.1.2: Tunnel Data Protection Algorithms

TEAPv1 meets this requirement by mandating TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory-to-implement ciphersuite as defined in Section 3.2.

A.5. Requirement 4.2.1.1.3: Tunnel Authentication and Key Establishment

TEAPv1 meets this requirement by mandating TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory-to-implement ciphersuite that provides certificate-based authentication of the server and is approved by NIST. The mandatory-to-implement ciphersuites only include ciphersuites that use strong cryptographic algorithms. They do not include ciphersuites providing mutually anonymous authentication or static Diffie-Hellman ciphersuites as defined in Section 3.2.

A.6. Requirement 4.2.1.2: Tunnel Replay Protection

TEAPv1 meets this requirement by using TLS to provide sufficient replay protection.

A.7. Requirement 4.2.1.3: TLS Extensions

TEAPv1 meets this requirement by allowing TLS extensions, such as TLS Certificate Status Request extension [RFC6066] and SessionTicket extension [RFC5077], to be used during TLS tunnel establishment.

A.8. Requirement 4.2.1.4: Peer Identity Privacy

TEAPv1 meets this requirement by establishment of the TLS tunnel and protection identities specific to the inner method. In addition, the peer certificate can be sent confidentially (i.e., encrypted).

A.9. Requirement 4.2.1.5: Session Resumption

TEAPv1 meets this requirement by mandating support of TLS session resumption as defined in Section 3.2.1 and TLS session resume using a PAC as defined in Section 3.2.2 .

A.10. Requirement 4.2.2: Fragmentation

TEAPv1 meets this requirement by leveraging fragmentation support provided by TLS as defined in Section 3.7.

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

TEAPv1 meets this requirement by including the TEAP version number received in the computation of the Crypto-Binding TLV as defined in Section 4.2.13.

A.12. Requirement 4.3.1: Extensible Attribute Types

TEAPv1 meets this requirement by using an extensible TLV data layer inside the tunnel as defined in Section 4.2.

A.13. Requirement 4.3.2: Request/Challenge Response Operation

TEAPv1 meets this requirement by allowing multiple TLVs to be sent in a single EAP request or response packet, while maintaining the half-duplex operation typical of EAP.

A.14. Requirement 4.3.3: Indicating Criticality of Attributes

TEAPv1 meets this requirement by having a mandatory bit in each TLV to indicate whether it is mandatory to support or not as defined in Section 4.2.

A.15. Requirement 4.3.4: Vendor-Specific Support

TEAPv1 meets this requirement by having a Vendor-Specific TLV to allow vendors to define their own attributes as defined in Section 4.2.8.

A.16. Requirement 4.3.5: Result Indication

TEAPv1 meets this requirement by having a Result TLV to exchange the final result of the EAP authentication so both the peer and server have a synchronized state as defined in Section 4.2.4.

A.17. Requirement 4.3.6: Internationalization of Display Strings

TEAPv1 meets this requirement by supporting UTF-8 format in the Basic-Password-Auth-Req TLV as defined in Section 4.2.14 and the Basic-Password-Auth-Resp TLV as defined in Section 4.2.15.

A.18. Requirement 4.4: EAP Channel-Binding Requirements

TEAPv1 meets this requirement by having a Channel-Binding TLV to exchange the EAP channel-binding data as defined in Section 4.2.7.

A.19. Requirement 4.5.1.1: Confidentiality and Integrity

TEAPv1 meets this requirement by running the password authentication inside a protected TLS tunnel.

A.20. Requirement 4.5.1.2: Authentication of Server

TEAPv1 meets this requirement by mandating authentication of the server before establishment of the protected TLS and then running inner password authentication as defined in Section 3.2.

A.21. Requirement 4.5.1.3: Server Certificate Revocation Checking

TEAPv1 meets this requirement by supporting TLS Certificate Status Request extension [RFC6066] during tunnel establishment.

A.22. Requirement 4.5.2: Internationalization

TEAPv1 meets this requirement by supporting UTF-8 format in Basic-Password-Auth-Req TLV as defined in Section 4.2.14 and Basic-Password-Auth-Resp TLV as defined in Section 4.2.15.

A.23. Requirement 4.5.3: Metadata

TEAPv1 meets this requirement by supporting Identity-Type TLV as defined in Section 4.2.3 to indicate whether the authentication is for a user or a machine.

A.24. Requirement 4.5.4: Password Change

TEAPv1 meets this requirement by supporting multiple Basic-Password-Auth-Req TLV and Basic-Password-Auth-Resp TLV exchanges within a single EAP authentication, which allows "housekeeping"" functions such as password change.

A.25. Requirement 4.6.1: Method Negotiation

TEAPv1 meets this requirement by supporting inner EAP method negotiation within the protected TLS tunnel.

A.26. Requirement 4.6.2: Chained Methods

TEAPv1 meets this requirement by supporting inner EAP method chaining within protected TLS tunnels as defined in Section 3.3.1.

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

TEAPv1 meets this requirement by supporting cryptographic binding of the inner EAP method keys with the keys derived from the TLS tunnel as defined in Section 4.2.13.

A.28. Requirement 4.6.4: Peer-Initiated EAP Authentication

TEAPv1 meets this requirement by supporting the Request-Action TLV as defined in Section 4.2.9 to allow a peer to initiate another inner EAP method.

A.29. Requirement 4.6.5: Method Metadata

TEAPv1 meets this requirement by supporting the Identity-Type TLV as defined in Section 4.2.3 to indicate whether the authentication is for a user or a machine.

Appendix B. Major Differences from EAP-FAST

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

1. The EAP method name has been changed from EAP-FAST to TEAP; this change thus requires that a new EAP Type be assigned.

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

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

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

5. Support is provided for passing optional Outer TLVs in the first two message exchanges, in addition to the Authority-ID TLV data in EAP-FAST.

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

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

Appendix C. Examples

C.1. Successful Authentication

The following exchanges show a successful TEAP authentication with basic password authentication and optional PAC refreshment. The conversation will appear as follows:

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

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

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

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

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

      TLS channel established
      (messages sent within the TLS channel)

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

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

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

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

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

      TLS channel torn down
      (messages sent in cleartext)

                              <- EAP-Success
```

C.2. Failed Authentication

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

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

    EAP-Response/
    Identity (MyID1) ->


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

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

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

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

    TLS channel established
    (messages sent within the TLS channel)

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

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

                            <- Result TLV (Failure)
```

```
       Result TLV (Failure) ->

       TLS channel torn down
       (messages sent in cleartext)

                                <- EAP-Failure
```

C.3. Full TLS Handshake Using Certificate-Based Ciphersuite

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

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

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

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

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

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

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

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

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

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

      // identity protected by TLS.

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

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

      // Method X exchanges followed by Protected Termination

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

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

      // TLS channel torn down
      (messages sent in cleartext)

                               <- EAP-Success
```

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

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

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

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

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

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

   // peer sends TLS client_hello to request TLS renegotiation
```

```
      TLS client_hello ->

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

                               <- TLS change_cipher_spec,
                                  TLS finished,
                                  Crypto-Binding TLV (Request),
                                  Result TLV (Success)

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

      //TLS channel torn down
      (messages sent in cleartext)

                               <- EAP-Success
```

C.5. Fragmentation and Reassembly

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

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

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

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

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

                              <- EAP-Request/
                                 EAP-Type=TEAP, V=1
                              (Fragment 2: M bit set)
      EAP-Response/
      EAP-Type=TEAP, V=1 ->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (Fragment 3)
      EAP-Response/
      EAP-Type=TEAP, V=1
      ([TLS certificate,]
       TLS client_key_exchange,
      [TLS certificate_verify,]
       TLS change_cipher_spec,
       TLS finished)
       (Fragment 1: L, M bits set)->

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

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

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

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

      // identity protected by TLS.

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

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

      // Method X exchanges followed by Protected Termination

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

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

      // TLS channel torn down
      (messages sent in cleartext)

                              <- EAP-Success
```

C.6. Sequence of EAP Methods

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

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

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

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

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

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

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

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

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

       // Optional additional X Method exchanges...

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

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

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

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

   // Compound MAC calculated using keys generated from
      EAP method X and the TLS tunnel.

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

          // Optional additional Y Method exchanges...

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

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

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

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

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

   // TLS channel torn down (messages sent in cleartext)

                          <- EAP-Success
```

C.7. Failed Crypto-Binding

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

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

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

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

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

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

EAP-Payload TLV/
EAP Identity Response ->

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

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

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

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

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

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

   // TLS channel torn down
      (messages sent in cleartext)

                              <- EAP-Failure
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

      // Compound MAC calculated using keys generated from
         EAP method X and the TLS tunnel.
```

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

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

                               <- Vendor-Specific TLV

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

      Result-TLV (Success) ->

      // TLS channel torn down (messages sent in cleartext)

                                <- EAP-Success
```

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

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

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

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

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

```
      EAP-Response/
      EAP-Type=TEAP, V=1
      [TLS certificate,]
       TLS client_key_exchange,
      [TLS certificate_verify,]
       TLS change_cipher_spec,
       TLS finished ->
                              <- EAP-Request/
                              EAP-Type=TEAP, V=1
                              (TLS change_cipher_spec,
                               TLS finished,
                               Crypto-Binding TLV (Request),
                                Result TLV (Success))

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

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

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

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

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

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

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

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

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

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

```
      // TLS channel torn down
      (messages sent in cleartext)

                              <- EAP-Success
```

C.10. Channel Binding

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

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

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

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

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

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

      TLS channel established
      (messages sent within the TLS channel)

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

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

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

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

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

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

      Result-TLV (Success) ->

      TLS channel torn down
      (messages sent in cleartext)

                             <- EAP-Success
```

Authors' Addresses

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

EMail: hzhou@cisco.com

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

EMail: ncamwing@cisco.com

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

EMail: jsalowey@cisco.com

Stephen Hanna
Infineon Technologies
79 Parsons Street
Brighton, MA 02135
US

EMail: steve.hanna@infineon.com