

---

Stream: Internet Engineering Task Force (IETF)  
RFC: 9930  
Obsoletes: 7170  
Updates: 9427  
Category: Standards Track  
Published: February 2026  
ISSN: 2070-1721  
Author: A. DeKok, Ed.

# RFC 9930

## 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. This document obsoletes RFC 7170 and updates RFC 9427 by moving all TEAP specifications from those documents to this one.

## 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 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <https://www.rfc-editor.org/info/rfc9930>.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions

with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction	7
1.1. Interoperability Issues	7
1.2. Requirements Language	8
1.3. Terminology	8
2. Protocol Overview	8
2.1. Architectural Model	9
2.2. Protocol-Layering Model	9
2.3. Outer TLVs Versus Inner TLVs	10
3. TEAP Protocol	10
3.1. Version Negotiation	11
3.2. TEAP Authentication Phase 1: Tunnel Establishment	12
3.3. Server Certificate Requirements	13
3.4. Server Certificate Validation	13
3.4.1. Client Certificates Sent During Phase 1	14
3.5. Resumption	14
3.5.1. TLS Session Resumption Using Server State	15
3.5.2. TLS Session Resumption Using Client State	15
3.6. TEAP Authentication Phase 2: Tunneled Authentication	15
3.6.1. Inner Method Ordering	16
3.6.2. Inner EAP Authentication	17
3.6.3. Inner Password Authentication	17
3.6.4. EAP-MSCHAPv2	18
3.6.5. Limitations on Inner Methods	19
3.6.6. Protected Termination and Acknowledged Result Indication	20
3.7. Determining Peer-Id and Server-Id	21
3.8. TEAP Session Identifier	21

3.9. Error Handling	22
3.9.1. Outer-Layer Errors	22
3.9.2. TLS Layer Errors	22
3.9.3. Phase 2 Errors	23
3.10. Fragmentation	23
3.11. Services Requested by the Peer	24
3.11.1. Certificate Provisioning Within the Tunnel	25
3.11.2. Certificate Content and Uses	26
3.11.3. Server Unauthenticated Provisioning Mode	26
3.11.4. Channel Binding	27
4. Message Formats	28
4.1. TEAP Message Format	28
4.2. TEAP TLV Format and Support	30
4.2.1. General TLV Format	30
4.2.2. Authority-ID TLV	32
4.2.3. Identity-Type TLV	32
4.2.4. Result TLV	34
4.2.5. NAK TLV	35
4.2.6. Error TLV	36
4.2.7. Channel-Binding TLV	38
4.2.8. Vendor-Specific TLV	39
4.2.9. Request-Action TLV	40
4.2.10. EAP-Payload TLV	41
4.2.11. Intermediate-Result TLV	42
4.2.12. PAC TLV	43
4.2.13. Crypto-Binding TLV	43
4.2.14. Basic-Password-Auth-Req TLV	46
4.2.15. Basic-Password-Auth-Resp TLV	46
4.2.16. PKCS#7 TLV	47
4.2.17. PKCS#10 TLV	48

4.2.18. Trusted-Server-Root TLV	49
4.2.19. CSR-Attributes TLV	50
4.2.20. Identity-Hint TLV	51
4.3. TLV Rules	52
4.3.1. Outer TLVs	53
4.3.2. Inner TLVs	53
5. Limitations of TEAPv1	54
5.1. Interoperable Inner Methods	55
5.2. Explanation and Background	55
5.3. Next Steps	55
6. Cryptographic Calculations	56
6.1. TEAP Authentication Phase 1: Key Derivations	56
6.2. Intermediate Compound Key Derivations	56
6.2.1. Generating the Inner Method Session Key	57
6.2.2. Generating S-IMCK	59
6.2.3. Choosing Inner Methods Securely	60
6.2.4. Managing and Computing Crypto-Binding	60
6.2.5. Unintended Side Effects	63
6.3. Computing the Compound-MAC	64
6.4. EAP Master Session Key Generation	65
7. IANA Considerations	66
7.1. TEAP TLV Types	66
7.2. TEAP Error TLV (value 5) Error Codes	67
7.3. TLS Exporter Labels	67
7.4. Extended Master Session Key (EMSK) Parameters	67
7.5. Extensible Authentication Protocol (EAP) Registry	68
8. Security Considerations	68
8.1. Mutual Authentication and Integrity Protection	68
8.2. Method Negotiation	68
8.3. Separation of Phase 1 and Phase 2 Servers	69

---

8.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies	69
8.4.1. User Identity Protection and Verification	70
8.5. Dictionary Attack Resistance	71
8.5.1. Protection Against On-Path Attacks	71
8.6. Protecting Against Forged Cleartext EAP Packets	72
8.7. Use of Cleartext Passwords	72
8.8. Accidental or Unintended Behavior	72
8.9. Implicit Challenge	73
8.10. Security Claims	73
9. Changes from RFC 7170	74
10. References	75
10.1. Normative References	75
10.2. Informative References	77
Appendix A. Evaluation Against Tunnel-Based EAP Method Requirements	80
A.1. Requirement 4.1.1: RFC Compliance	80
A.2. Requirement 4.2.1: TLS Requirements	80
A.3. Requirement 4.2.1.1.1: Cipher Suite Negotiation	80
A.4. Requirement 4.2.1.1.2: Tunnel Data Protection Algorithms	81
A.5. Requirement 4.2.1.1.3: Tunnel Authentication and Key Establishment	81
A.6. Requirement 4.2.1.2: Tunnel Replay Protection	81
A.7. Requirement 4.2.1.3: TLS Extensions	81
A.8. Requirement 4.2.1.4: Peer Identity Privacy	81
A.9. Requirement 4.2.1.5: Session Resumption	81
A.10. Requirement 4.2.2: Fragmentation	81
A.11. Requirement 4.2.3: Protection of Data External to Tunnel	81
A.12. Requirement 4.3.1: Extensible Attribute Types	81
A.13. Requirement 4.3.2: Request/Challenge Response Operation	82
A.14. Requirement 4.3.3: Indicating Criticality of Attributes	82
A.15. Requirement 4.3.4: Vendor-Specific Support	82
A.16. Requirement 4.3.5: Result Indication	82

---

A.17. Requirement 4.3.6: Internationalization of Display Strings	82
A.18. Requirement 4.4: EAP Channel-Binding Requirements	82
A.19. Requirement 4.5.1.1: Confidentiality and Integrity	82
A.20. Requirement 4.5.1.2: Authentication of Server	82
A.21. Requirement 4.5.1.3: Server Certificate Revocation Checking	82
A.22. Requirement 4.5.2: Internationalization	83
A.23. Requirement 4.5.3: Metadata	83
A.24. Requirement 4.5.4: Password Change	83
A.25. Requirement 4.6.1: Method Negotiation	83
A.26. Requirement 4.6.2: Chained Methods	83
A.27. Requirement 4.6.3: Cryptographic Binding with the TLS Tunnel	83
A.28. Requirement 4.6.4: Peer-Initiated EAP Authentication	83
A.29. Requirement 4.6.5: Method Metadata	83
Appendix B. Major Differences from EAP-FAST	84
Appendix C. Examples	84
C.1. Successful Authentication	84
C.2. Failed Authentication	85
C.3. Full TLS Handshake Using Certificate-Based Cipher Suite	86
C.4. Client Authentication During Phase 1 with Identity Privacy	88
C.5. Fragmentation and Reassembly	90
C.6. Sequence of EAP Methods	92
C.7. Failed Crypto-Binding	94
C.8. Sequence of EAP Method with Vendor-Specific TLV Exchange	96
C.9. Peer Requests Inner Method After Server Sends Result TLV	98
C.10. Channel Binding	100
C.11. PKCS Exchange	102
C.12. Failure Scenario	103
C.13. Client Certificate in Phase 1	104
Acknowledgments	105
Contributors	106

---

## 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) [RFC8446] 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.

This document describes the Tunnel Extensible Authentication Protocol (TEAP) version 1, which is based on EAP-FAST [RFC4851]. The changes from EAP-FAST to TEAP are largely minor in order to meet the requirements outlined in [RFC6678] for a standard tunnel-based EAP method.

This document also defines cryptographic derivations for use with TLS 1.2. When TLS 1.3 is used, the definitions of cryptographic derivations in [RFC9427] **MUST** be used instead of the ones given here.

Note that while it is technically possible to use TEAPv1 with TLS 1.0 and TLS 1.1, those protocols have been deprecated in [RFC8996]. As such, the definitions given here are only applicable for TLS 1.2 and TLS 1.3.

### 1.1. Interoperability Issues

This document contains substantial changes from [RFC7170]. These changes are largely clarifications and corrections to that specification.

However, there is one major change from [RFC7170] in the specification of the cryptographic-binding information. While there were multiple implementations of [RFC7170], the text in that document was interpreted differently by each implementation. The implementations are interoperable but only for a subset of the functionality described in [RFC7170].

This specification describes how TEAPv1 works in theory but also explains what subset of TEAPv1 is currently interoperable. In order to simplify the description of an already complex specification, all interoperability issues are documented separately from the normal protocol operation.

Please see [Section 5](#) for further discussion of interoperability issues.

## 1.2. Requirements Language

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 BCP 14 [[RFC2119](#)] [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

## 1.3. Terminology

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

### Type-Length-Value (TLV)

The TEAP protocol utilizes objects in TLV format. The TLV format is defined in [Section 4.2](#).

### Inner Method

An authentication method that is sent as application data inside of a TLS exchange that is carried over TEAP. The inner method can be an EAP authentication method, a username/password authentication, or a vendor-specific authentication method. Where the TLS connection is authenticated, the inner method could also be a Public Key Cryptography Standard (PKCS) exchange.

## 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 [[RFC8446](#)] 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.

As discussed in [Section 2.1.7](#) of [[RFC9190](#)] and [Section 3.1](#) of [[RFC9427](#)], the outer EAP Identity **SHOULD** be an anonymous Network Access Identifier (NAI) as described in [Section 2.4](#) of [[RFC7542](#)]. While [Section 5.1](#) of [[RFC3748](#)] places no limits on the contents of the Identity field, [Section 2.6](#) of [[RFC7542](#)] states that Identities that do not follow the NAI format cannot be transported in an Authentication, Authorization, and Accounting (AAA) proxy network. As such, Identities in non-NAI form are likely to not work outside of limited and local networks.

Any inner identities (EAP or otherwise) **SHOULD** also follow the recommendations of [[RFC9427](#)], [Section 3.1](#) about inner identities.

[RFC7170] defined a Protected Access Credential (PAC) to mirror EAP-FAST [RFC4851]. However, implementation experience and analysis determined that the PAC was not necessary. Instead, TEAP performs session resumption using the NewSessionTicket message as defined in Sections 2.1.2 and 2.1.3 of [RFC9190]. As such, the PAC has been deprecated.

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 (Master Session Key [RFC3748]) 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:

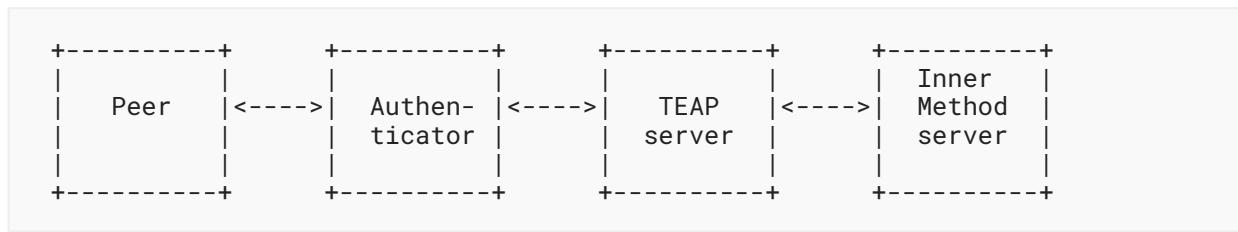


Figure 1: TEAP Architectural Model

The Peer and Authenticator are defined in [RFC3748], Section 1.2. The TEAP server is the "backend authentication server" defined in [RFC3748], Section 1.2. The "Inner Method server" is usually part of the TEAP server and handles the application data (inner methods, EAP, passwords, etc.) inside of the TLS tunnel.

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 8.3 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:

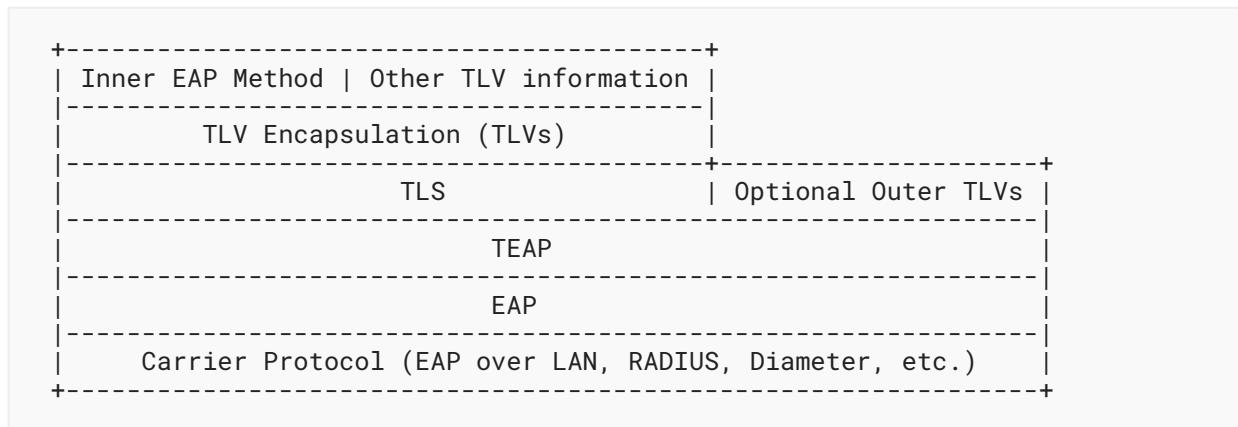


Figure 2: Protocol-Layering Model

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 data exchanges in the TEAP-protected tunnel are encapsulated in a TLV layer.

Methods for encapsulating EAP within carrier protocols are already defined. For example, IEEE 802.1X [[IEEE.802-1X.2020](#)] 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.

### 2.3. Outer TLVs Versus Inner TLVs

TEAP packets may include TLVs both inside and outside the TLS tunnel defined as follows:

#### Outer TLVs

This term 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 fragments is counted as one message.

#### Inner TLVs

This term 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 TEAP, TLS records may encapsulate zero or more Inner TLVs, but no Outer TLVs are used.

## 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 6](#).

### 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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 bid-down attack on 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.

Intermediate results are signaled via the Intermediate-Result TLV ([Section 4.2.11](#)). However, the Crypto-Binding TLV **MUST** be validated before any Intermediate-Result TLV or Result TLV is examined. If the Crypto-Binding TLV fails to be validated for any reason, then it is a fatal error and is handled as described in [Section 3.9.3](#).

The true success or failure of TEAP is conveyed by the Result TLV with value Success or Failure. 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.

### 3.2. TEAP Authentication Phase 1: Tunnel Establishment

TEAP relies on the TLS handshake [[RFC8446](#)] 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 cipher suites:

- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256

Other cipher suites **MAY** be supported. Implementations **MUST** implement the recommended cipher suites in [[RFC9325](#)], [Section 4.2](#) for TLS 1.2 and in [[RFC9325](#)], [Section 4.3](#) for TLS 1.3.

It is **REQUIRED** that anonymous cipher suites such as TLS\_DH\_anon\_WITH\_AES\_128\_CBC\_SHA [[RFC5246](#)] only be used in the case when the inner method provides mutual authentication, key generation, and resistance to on-path and dictionary attacks. TLS cipher suites that do not provide confidentiality **MUST NOT** be used. During the TEAP Phase 1, 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 the Online Certificate Status Protocol (OCSP) [[RFC6960](#)] to check the validity of server certificates. TLS renegotiation indications defined in [[RFC5746](#)] **MUST** be supported.

Use of TLS-PSK is **NOT RECOMMENDED**. TEAP has not been designed to work with TLS-PSK, and no use cases, security analyses, or implementations have been done. TLS-PSK may work (or not) with TEAP, depending on the status of a particular implementation, and it is therefore not useful to deploy it.

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 cipher suites 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 cipher suite. 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](#).

### 3.3. Server Certificate Requirements

Server certificates **MUST** include a subjectAltName extension, with the dnsName attribute containing a Fully Qualified Domain Name (FQDN) string. Server certificates **MAY** also include a SubjectDN containing a single element, "CN=", which contains the FQDN of the server. However, this use of SubjectDN is deprecated for TEAP and is forbidden in [\[RFC9525\], Section 2](#).

The KeyUsage extensions **MAY** be included but are not required.

The ExtendedKeyUsage extensions defined in [\[RFC5280\]](#) **MAY** also be included, but their use is discouraged. Systems **SHOULD** use a private Certification Authority (CA) for EAP in preference to public CAs. The most commonly used public CAs are focused on the web, and those certificates are not always suitable for use with EAP. In contrast, private CAs can be designed for any purposes and can be restricted to an enterprise or an other organization.

### 3.4. Server Certificate Validation

As part of the TLS negotiation, the server usually presents a certificate to the peer. In most cases, the certificate needs to be validated, but there are a number of situations where the EAP peer does not need to do certificate validation:

- when the peer has the server's End Entity (EE) certificate pinned or loaded directly into its trusted anchor information [\[RFC4949\]](#);
- when the peer is requesting server unauthenticated provisioning;
- when the peer is certain that it will be using an authenticated inner method.

In some cases, such as onboarding (or "bootstrapping"), the certificate validation may be delayed. However, once the onboarding has taken place, the validation steps described below **MUST** still be performed.

In all other cases, the EAP peer **MUST** validate the server certificate. This validation is done in the same manner as is done for EAP-TLS, which is discussed in [\[RFC9190\], Section 5.3](#) and in [\[RFC5216\], Section 5.3](#). Further guidance on server identity validation can be found in [\[RFC9525\], Section 6](#).

Where the EAP peer has an NAI, EAP peers **MUST** use the realm to perform the DNS-ID validation as per [\[RFC9525\], Section 6](#). The realm is used both to construct the list of reference identifiers as defined in [\[RFC9525\], Section 6.1.2](#), and as the "source domain" field of that same section.

When performing server certificate validation, implementations **MUST** also support 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 comparison might not be appropriate or enabled.

In most situations, the EAP peer will have no network access during the authentication process. It will therefore have no way of correlating the server identity given in the certificate presented by the EAP server with a hostname, as is done with other protocols such as HTTPS. Therefore, if the EAP peer has no preconfigured trust anchor, it will have few, if any, ways of validating the server's certificate.

### 3.4.1. Client Certificates Sent During Phase 1

Note that since TLS client certificates are sent in the clear with TLS 1.2, 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 sending 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 EAP (for example, EAP-TLS) within the TLS session in TEAP Phase 2 instead of using TLS handshake renegotiation.

When TLS 1.3 or later is used, it is **RECOMMENDED** that client certificates are sent in Phase 1 instead of via Phase 2 and EAP-TLS. Doing so will reduce the number of round trips. Further discussion of this issue is given below in [Section 3.6.5](#)

## 3.5. Resumption

For resumption, [\[RFC9190\]](#), [Section 5.7](#) discusses EAP-TLS resumption for all versions of TLS and is incorporated herein by reference. [\[RFC9427\]](#), [Section 4](#) is also incorporated by reference, as it provides generic discussion of resumption for TLS-based EAP methods when TLS 1.3 is used.

This document only describes TEAP issues when resumption is used for TLS versions of 1.2 and earlier. It also describes resumption issues that are specific to TEAP for TLS 1.3.

If the server agrees to resume the session, Phase 2 is bypassed entirely. If the server does not agree to resume the session, then the server rejects the resumption as per [\[RFC9190\]](#), [Section 5.7](#). It then continues with a full handshake. After the full TLS handshake has completed, both EAP server and peer **MUST** proceed with Phase 2.

All TEAP implementations **MUST** support resumption. Using resumption can significantly improve the scalability and stability of authentication systems. For example, some environments such as universities may have users re-authenticating multiple times a day, if not hourly. Failure to implement resumption would increase the load on the user database by orders of magnitude, leading to unnecessary cost.

The following sections describe how a TEAP session can be resumed based on server-side or client-side state.

### 3.5.1. TLS Session Resumption Using Server State

TEAP session resumption is achieved in the same manner TLS achieves session resumption. To support session resumption, the server and peer cache the Session ID, master secret, and cipher suite. 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.

### 3.5.2. TLS Session Resumption Using Client State

TEAP supports the resumption of sessions based on the state being stored on the client side using the TLS SessionTicket extension techniques described in [\[RFC5077\]](#) and [\[RFC9190\]](#).

## 3.6. 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 3.6.6](#).

If the peer is not authenticated in Phase 1, the TEAP peer **SHOULD** send one or more Identity-Hint TLVs ([Section 4.2.20](#)) as soon as the TLS connection has been established. This information lets the TEAP server choose an authentication type that is appropriate for that identity. When the TEAP peer does not provide an Identity-Hint TLV, the TEAP server does not know which inner method is supported by the peer. It must choose an inner method and propose it to the peer, which may reject that inner method. As a result, the peer fails to authenticate and fails to obtain network access.

The TLV exchange includes the execution of zero or more inner methods within the protected tunnel as described in [Sections 3.6.2](#) and [3.6.3](#). A server **MAY** proceed directly to the protected termination exchange without performing any inner authentication if it does not wish to request further authentication from the peer. A server **MAY** request one or more authentications within the protected tunnel. After completion of each inner method, the server decides whether or not to begin another inner method or to send a Result TLV.

Implementations **MUST** support at least two sequential inner methods, which allows both machine and user authentication to be performed. Implementations **SHOULD** also limit the number of sequential inner authentications, as there is no reason to perform a large number of inner authentications in one TEAP conversation.

Implementations wishing to use their own proprietary authentication method may substitute the EAP-Payload or Basic-Password-Auth-Req TLV for the Vendor-Specific TLV, which carries another authentication method. Any vendor-specific authentication method **MUST** support calculation of the Crypto-Binding TLV and **MUST** use Intermediate-Result TLV and Result TLV as is done with other authentication methods.

### 3.6.1. Inner Method Ordering

Due to issues noted in [Section 5](#), the order of inner methods has implications for both security and interoperability.

Where the authentication is expected to use multiple inner methods, implementations **SHOULD** order the methods so that a method that derives an Extended Master Session Key (EMSK) is used first before any other method. This ordering helps to securely tie the inner method to the TLS session and therefore can prevent attacks.

Implementations **SHOULD** support both EAP and basic password for inner methods. Implementations that support multiple types of inner methods (User and Machine) **MUST** support all of those methods in any order or combination. That is, it is explicitly permitted to "mix and match" inner methods.

For example, a server can request user authentication from the peer and have the peer return machine authentication (or vice versa). If the server is configured to accept machine authentication, it **MUST** accept that response. If that authentication succeeds, then depending on local policy, the server **SHOULD** proceed with requesting user authentication again.

Similarly, a peer that is configured to support multiple types of inner methods (User and Machine) can return a method other than what the server requested. This action is usually taken by the peer so that it orders inner methods for increased security. See [Section 6.2.3](#) for further discussion of this issue.

However, the peer and server **MUST NOT** assume that either will skip inner 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](#).

The completion of each inner method is signaled by an Intermediate-Result TLV. Where the Intermediate-Result TLV indicates failure, an Error TLV **SHOULD** also be included using the most descriptive error code possible. The Intermediate-Result TLV may be accompanied by another TLV indicating that the server wishes to perform a subsequent authentication. When all inner methods have completed, the server **MUST** send a Result TLV indicating success or failure instead of a TLV that carries an inner method.

### 3.6.2. Inner EAP Authentication

EAP [[RFC3748](#)] prohibits use of multiple authentication methods within a single EAP conversation in order to limit vulnerabilities to on-path attacks. TEAP addresses on-path attacks through support for cryptographic protection of the inner EAP exchange and cryptographic binding of the inner EAP method(s) to the protected tunnel. Inner methods are executed serially in a sequence. This version of TEAP does not support initiating multiple inner methods simultaneously in parallel. The inner methods need not be distinct. For example, EAP-TLS ([[RFC5216](#)] and [[RFC9190](#)]) could be run twice as an inner method, first using machine credentials, followed by a second instance using user credentials.

When EAP is used as an inner method, the EAP messages are carried within EAP-Payload TLVs defined in [Section 4.2.10](#). Note that in this use case, TEAP is simply a carrier for EAP, much as RADIUS is a carrier for EAP. The full EAP state machine runs as normal and is carried over the EAP-Payload TLV. Each distinct EAP authentication **MUST** be managed as a separate EAP state machine.

A TEAP server therefore **MUST** begin an EAP authentication with an EAP-Request/Identity (carried in an EAP-Payload TLV). However, a TEAP server **MUST NOT** finish the EAP conversation with an EAP Success or EAP Failure packet; the Intermediate-Result TLV is used instead.

Upon completion of each EAP authentication in the tunnel, the server **MUST** send an Intermediate-Result TLV indicating the result of that authentication. When the result indicates success, it **MUST** be accompanied by a Crypto-Binding TLV. The peer **MUST** respond to the Intermediate-Result TLV indicating its own result and similarly on success **MUST** accompany the TLV with its own Crypto-Binding TLV. The Crypto-Binding TLV is further discussed in [Sections 4.2.13](#) and [6.3](#). The Intermediate-Result TLVs can be included with other TLVs that indicate a subsequent authentication or with the Result TLV used in the protected termination exchange.

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

### 3.6.3. Inner Password Authentication

The authentication server (AS) 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 that contains the username and password as defined in [Section 4.2.15](#). If it does not wish to perform password authentication, then it responds with a Negative Acknowledgment (NAK) TLV indicating the rejection of the Basic-Password-Auth-Req TLV.

The basic password authentication defined here is similar in functionality to that used by EAP-TTLS [[RFC5281](#)] with inner password authentication. It shares a similar security and risk analysis.

Multiple round trips of password authentication requests and responses **MAY** be used to support some "housekeeping" functions such as a password or pin change before a user is considered to be authenticated. Multiple rounds **MAY** also be used to communicate a user's password and, separately, a one-time password such as Time-Based One-Time Passwords (TOTPs) [RFC6238].

Implementations **MUST** limit the number of round trips for password authentication. It is reasonable to use one or two round trips. Further round trips are likely to be problematic and **SHOULD** be avoided.

The first Basic-Password-Auth-Req TLV received in a session **MUST** include a prompt, which the peer displays to the user. Subsequent authentication rounds **SHOULD** include a prompt, but it is not always necessary.

When the peer first receives a Basic-Password-Auth-Req TLV, it should allow the user to enter both a username and a password, which are then placed in the Basic-Password-Auth-Resp TLV. If the peer receives subsequent Basic-Password-Auth-Req TLVs in the same authentication session, it **MUST NOT** prompt for a username and instead allow the user to enter only a password. The peer **MUST** copy the username used in the first Basic-Password-Auth-Resp TLV into all subsequent Basic-Password-Auth-Resp TLVs.

Servers **MUST** track the username across multiple password rounds and reject authentication if the identity changes from one Basic-Password-Auth-Resp TLV to the next. There is no reason for a user (or machine) to change identities in the middle of authentication.

Upon reception of a Basic-Password-Auth-Resp TLV in the tunnel, 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 6.3.

The Intermediate-Result TLVs can be included with other TLVs that indicate a subsequent authentication or with the Result TLV used in the protected termination exchange.

The use of EAP-FAST-GTC as defined in [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.

### 3.6.4. EAP-MSCHAPv2

If using EAP-MSCHAPv2 [KAMATH] as an inner EAP method, the EAP-FAST-MSCHAPv2 variant defined in [RFC5422], Section 3.2.3 **MUST** be used instead of the derivation defined in [MSCHAP].

The difference between EAP-MSCHAPv2 and EAP-FAST-MSCHAPv2 is that the first and the second 16 octets of the EAP-MSCHAPv2 Master Session Key (MSK) are swapped when it is used as the Inner Method Session Keys (IMSKs) for TEAP.

### 3.6.5. Limitations on Inner Methods

Implementations **SHOULD** limit the permitted inner EAP methods to a small set such as EAP-TLS and the EAP-FAST-MSCHAPv2 variant of EAP-MSCHAPv2. These EAP methods are the most commonly supported inner methods in TEAP and are known to be interoperable among multiple implementations.

Other EAP methods such as EAP-pwd, EAP-SIM, EAP-AKA, or EAP-AKA' can be used within a TEAP tunnel. Any EAP method that derives both MSK and EMSK is likely to work as an inner method for TEAP, because EAP-TLS has that behavior and it works. EAP methods that derive only MSK should work, as EAP-FAST-MSCHAPv2 has that behavior, and it works. Other EAP methods are untested and may or may not work.

Tunneled EAP methods such as PEAP [[PEAP](#)], EAP-TTLS [[RFC5281](#)], and EAP-FAST [[RFC4851](#)] **MUST NOT** be used for inner EAP authentication. There is no reason to have multiple layers of TLS in order to protect a password exchange.

The EAP methods defined in [[RFC3748](#)], [Section 5](#), such as MD5-Challenge, One-Time Password (OTP), and Generic Token Card (GTC), do not derive an MSK or an EMSK and are vulnerable to on-path attacks. The construction of the OTP and GTC methods makes this attack less relevant, as the information being sent is generally a one-time token. However, EAP-OTP and EAP-GTC offer no benefit over the basic password authentication defined in [Section 3.6.3](#), which also does not perform crypto-binding of the inner method to the TLS session. These EAP methods are therefore not useful as Phase 2 methods within TEAP.

Other EAP methods are less widely used and highly likely to not work as the inner EAP method for TEAP.

In order to protect from on-path attacks, TEAP implementations **MUST NOT** permit the use of inner EAP methods that fail to perform crypto-binding of the inner method to the TLS session.

Implementations **MUST NOT** permit resumption for the inner EAP methods such as EAP-TLS. If the user or machine needs to be authenticated, it should use a method that provides full authentication. If the user or machine needs to do resumption, it can perform a full authentication once and then rely on the outer TLS session for resumption. This restriction applies also to all TLS-based EAP methods that can tunnel other EAP methods. As a result, this document updates [[RFC9427](#)].

In general, the reason to use a non-TLS-based EAP method inside of a TLS-based EAP method such as TEAP is for privacy. Many previous EAP methods may leak information about user identity, and those leaks are prevented by running the method inside of a protected TLS tunnel.

EAP-TLS is permitted in Phase 2 for two use cases. The first use case is when TLS 1.2 is used, as the client certificate is not protected as with TLS 1.3. It is therefore **RECOMMENDED** that when TLS 1.3 is used for the outer TEAP exchange, the client certificate is sent in Phase 1 instead of doing EAP-TLS in Phase 2. This behavior will simplify the authentication exchange and use fewer round trips than doing EAP-TLS inside of TEAP.

The second use case for EAP-TLS in Phase 2 is where both the user and machine use client certificates for authentication. Since TLS permits only one client certificate to be presented, only one certificate can be used in Phase 1. The second certificate is then presented via EAP-TLS in Phase 2.

For basic password authentication, it is **RECOMMENDED** that this method be only used for the exchange of one-time passwords. The existence of password-based EAP methods such as EAP-pwd ([RFC5931] and [RFC8146]) makes most cleartext password exchanges unnecessary. The updates to EAP-pwd in [RFC8146] permit it to be used with databases that store passwords in "salted" form, which greatly increases security.

Where no inner method provides an EMSK, the Crypto-Binding TLV offers little protection, as it cannot tie the inner EMSK to the TLS session via the TLS-PRF. As a result, the TEAP session will be vulnerable to on-path active attacks. Implementations and deployments **SHOULD** adopt various mitigation strategies described in [RFC7029], [Section 3.2](#). Implementations also need to implement the inner method ordering described in [Section 6.1](#) in order to fully prevent on-path attacks.

### 3.6.6. 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 inner methods 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. Except as noted below, the Crypto-Binding TLV **MUST** be exchanged and verified before the final Result TLV exchange, regardless of whether or not there is an inner method. The Crypto-Binding TLV and Intermediate-Result TLV **MUST** be included to perform cryptographic binding after each successful authentication in a sequence of one or more inner 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 to be run), 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 3.9.3](#).

### 3.7. 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 authentication method is run, then the Peer-Id is obtained from that inner EAP authentication 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.

### 3.8. TEAP Session Identifier

For TLS 1.2 and earlier, 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:

- | denotes concatenation,
- teap\_type is the EAP Type assigned to TEAP, and
- tls-unique = 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 derivation for TLS 1.3 is given in [\[RFC9427\]](#), [Section 2.1](#)

### 3.9. Error Handling

TEAP uses the error-handling rules summarized below:

1. Errors in the outer EAP packet layer are handled as defined in [Section 3.9.1](#).
2. Errors in the TLS layer are communicated via TLS alert messages in both phases of TEAP.
3. The Intermediate-Result TLVs carry success or failure indications of the individual inner methods in TEAP Phase 2. Errors within an EAP conversation in Phase 2 are expected to be handled by the individual EAP authentication 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.

#### 3.9.1. 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.

#### 3.9.2. 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 TEAP exchange so as to allow the peer to inform the user of the cause of the failure. The TEAP peer **MUST** send a TEAP response to an alert message. The EAP-Response packet sent by the peer **SHOULD** contain a TEAP response with a zero-length message. The server **MUST** terminate the TEAP exchange with an EAP Failure packet no matter what the client says.

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, which contains a zero-length message. The server then **MUST** terminate the conversation with an EAP failure as discussed in the previous paragraph.

While TLS 1.3 [RFC8446] allows for the TLS conversation to be restarted, it is not clear when that would be useful (or used) for TEAP. Fatal TLS errors will cause the TLS conversation to fail. Non-fatal TLS errors can likely be ignored entirely. As a result, TEAP implementations **MUST NOT** permit TLS restarts.

### 3.9.3. Phase 2 Errors

There are a large number of situations where errors can occur during Phase 2 processing. This section describes both errors and the recommended processing of them.

When the server receives a Result TLV with a fatal Error TLV from the peer, it **MUST** terminate the TLS tunnel and reply with an EAP Failure.

When the peer receives a Result TLV with a fatal Error TLV from the server, it **MUST** respond with a Result TLV indicating failure. The server **MUST** discard any data it receives from the peer and reply with an EAP Failure. The final message from the peer is required by the EAP state machine and serves only to allow the server to reply to the peer with the EAP Failure.

The following items describe specific errors and processing in more detail.

#### Fatal Error processing a TLV:

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 using the most descriptive error code possible.

#### Fatal Error during TLV Exchanges:

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.

#### Fatal Error due to tunnel compromise:

For errors involving a tunnel compromise, such as when the Crypto-Binding TLV fails validation, the error code is Tunnel Compromise Error.

#### Non-Fatal Error due to inner method:

If there is a non-fatal error while running the inner method, the receiving side **SHOULD NOT** silently drop the inner method exchange. Instead, it **SHOULD** reply with an Error TLV using the most descriptive error code possible.

If there is no error code that matches the particular issue, then the value Inner Method Error (1001) **SHOULD** be used. This response is a positive indication that there was an error processing the current inner method. The side receiving a non-fatal Error TLV **MAY** decide to start a new and different inner method instead or send back a Result TLV to terminate the TEAP authentication session.

## 3.10. Fragmentation

Fragmentation of EAP packets is discussed in [RFC5216], Section 2.1.5. There is no special handling of fragments for TEAP.

---

### 3.11. Services Requested by the Peer

Several TEAP operations, including server unauthenticated provisioning, certificate provisioning, and channel binding, depend on the peer trusting the TEAP server. If the peer trusts the provided server certificate, then the server is authenticated.

Typically, this authentication process involves the peer validating the certificate to a trust anchor by verifying that the server presenting the certificate holds the private key and confirming that the entity named by the certificate is the intended server. Server authentication also occurs when the procedures in [Section 3.2](#) are used to resume a session where the peer and server were previously mutually authenticated. Alternatively, the server is deemed to be authenticated if an inner EAP method provides mutual authentication along with an MSK and/or EMSK. The inner method **MUST** also provide for cryptographic binding via the Compound Message Authentication Code (MAC), as discussed in [Section 4.2.13](#). This issue is further described in [Section 3.11.3](#).

TEAP peers **MUST** track whether or not server authentication has taken place. When the server cannot be authenticated, the peer **MUST NOT** request any services such as certificate provisioning ([Section 3.11.1](#)) from it.

Unless the peer requests server unauthenticated provisioning, it **MUST** authenticate the server, and fail the current authentication session fails if the server cannot be authenticated.

An additional complication arises when an inner 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 the peer should request these services or whether the server should provide them.

The server **MUST** also authenticate the peer before providing these services. The alternative is to send provisioning data to unauthenticated and potentially malicious peers, which can have negative impacts on security.

When a device is provisioned via TEAP, any subsequent authorization **MUST** be done on the authenticated credentials. That is, there may be no credentials (or anonymous credentials) passed in Phase 1, but there will be credentials passed or provisioned in Phase 2. If later authorizations are done on the Phase 1 identity, then a device could obtain the wrong authorization. If authorization is done on the authenticated credentials instead, then the device will obtain the correct kind of network access.

The correct authorization must also be applied to any resumption, as noted in [[RFC9190](#)], [Section 5.7](#). However, as it is possible in TEAP for the credentials to change, the new credentials **MUST** be associated with the session ticket. If this association cannot be done, then the server **MUST**

invalidate any session tickets for the current session. This invalidation will force a full re-authentication on any subsequent connection; at which point, the correct authorization will be associated with any session ticket.

Note that the act of re-provisioning a device is essentially indistinguishable from any initial provisioning. The device authenticates and obtains new credentials via the standard provisioning mechanisms. The only caveat is that the device **SHOULD NOT** discard the old credentials unless either they are known to have expired or the new credentials have been verified to work.

### 3.11.1. 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] unsigned (i.e., degenerate) "Certificates Only" message encoded into the body of a PKCS#7 TLV (see [Section 4.2.16](#)) only after an inner 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 cipher suite 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 signed certificate request is being presented by an authenticated TEAP peer that is in possession of the private key.

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.

### 3.11.2. Certificate Content and Uses

It is not enough to verify that the CSR provided by the peer to the authenticator is from an authenticated user. The CSR itself should also be examined by the authenticator or CA before any certificate is issued.

The format of a CSR is complex and contains a substantial amount of information. That information could be incorrect, such as a user claiming a wrong physical address, email address, etc. It is **RECOMMENDED** that systems provisioning these certificates validate that the CSR contains the expected data and that it does not contain unexpected data. For example, a CA could refuse to issue the certificate if the CSR contained unknown fields or if a known field contained an unexpected or invalid value. The CA can modify or refuse a particular CSR to address these deficiencies for any reasons, including local site policy. We note that the "A" in "CA" means for "Authority", while the "R" in "CSR" means "Request". There is no requirement for a CA to sign any and all CSRs that are presented to it.

Once an EAP peer receives the signed certificate, the peer could potentially be (ab)used for in TLS contexts other than TEAP. For example, the certificate could be used with EAP-TLS, or even with HTTPS. It is **NOT RECOMMENDED** to use certificates provisioned via TEAP for any non-TEAP protocol. One method of enforcing this restriction is to have different CAs (or different intermediate CAs) that issue certificates for different uses. For example, TLS-based EAP methods could share one CA, and even use different intermediary CAs for different TLS-based EAP methods. HTTPS servers could use an entirely different CA. The different protocols could then be configured to validate client certificates only from their preferred CA, which would prevent peers from using certificates outside of the intended use case.

Another method of limiting the uses of a certificate is to provision it with an appropriate value for the Extended Key Usage field [[RFC7299](#)]. For example, the id-kp-eapOverLAN [[RFC4334](#)] value could be used to indicate that this certificate is intended for use only with EAP.

It is difficult to give more detailed recommendations than the above. Each CA or organization may have its own local policy as to what is permitted or forbidden in a certificate. All we can do in this document is to highlight the issues and make the reader aware that they have to be addressed.

### 3.11.3. Server Unauthenticated Provisioning Mode

In Server Unauthenticated Provisioning Mode, an unauthenticated tunnel is established in Phase 1, and the peer and server negotiate an inner method or methods in Phase 2. This inner method **MUST** support mutual authentication, provide key derivation, and be resistant to attacks such as on-path and dictionary attacks. In most cases, this inner method will be an EAP authentication method. Example inner methods that satisfy these criteria include EAP-pwd [[RFC5931](#)] and EAP-EKE [[RFC6124](#)] but not EAP-FAST-MSCHAPv2.

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 cipher suite negotiated does not provide authentication and in which the cipher suite negotiated provides the authentication, but the peer is unable to validate the identity of the server for some reason.

Upon successful completion of the inner 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 an on-path attack has not been attempted.

Support for the Server Unauthenticated Provisioning Mode is optional. The cipher suite `TLS_DH_anon_WITH_AES_128_CBC_SHA` is **RECOMMENDED** when using Server Unauthenticated Provisioning Mode, but other anonymous cipher suites **MAY** be supported as long as the TLS pre-master secret is generated from contribution from both peers.

When a strong inner method is not used with Server Unauthenticated Provisioning Mode, it is possible for an attacker to perform an on-path attack. In effect, Server Unauthenticated Provisioning Mode has similar security issues as just running the inner method in the open without the protection of TLS. All of the information in the tunnel should be assumed to be visible to, and modifiable by, an attacker.

Implementations **SHOULD** exchange minimal data in Server Unauthenticated Provisioning Mode. Instead, they should use that mode to set up a secure/authenticated tunnel and then use that tunnel to perform any needed data exchange.

It is **RECOMMENDED** that client implementations and deployments authenticate TEAP servers if at all possible. Authenticating the server means that a client can be provisioned securely with no chance of an attacker eaves-dropping on the connection.

Note that server unauthenticated provisioning can only use anonymous cipher suites in TLS 1.2 and earlier. These cipher suites have been deprecated in TLS 1.3 ([[RFC8446](#)], [Appendix C.5](#)). For TLS 1.3, the server **MUST** provide a certificate, and the peer performs server unauthenticated provisioning by not validating the certificate chain or any of its contents.

### 3.11.4. Channel Binding

[[RFC6677](#)] defines channel bindings for EAP that 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 authentication 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 did not initiate the channel-binding request and the peer still wants to

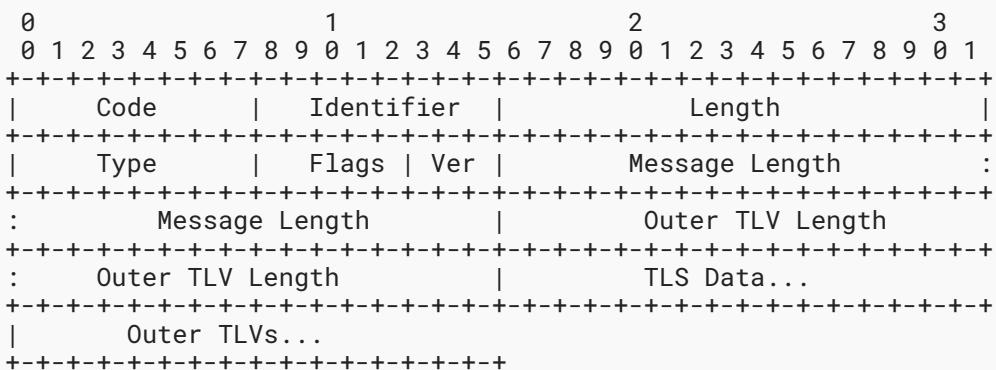
send the channel-binding information to the server, it can do that by using the Request-Action TLV along with the Channel-Binding TLV. The peer **MUST** only send channel-binding information after it has successfully authenticated the server and established the protected tunnel.

## 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

0	1	2	3	4
+	+	+	+	+
	L	M	S	O
+	+	+	+	+

- 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 acknowledgment 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** only send a NAK TLV for a TLV that is marked mandatory but is not understood and **MUST NOT** otherwise send a NAK TLV. If all TLVs in a message are marked optional and none are understood by the peer, then a Result TLV **SHOULD** be sent to the other side in order to continue the conversation. It is also possible to send a NAK TLV when all TLVs in a message are marked optional.

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.

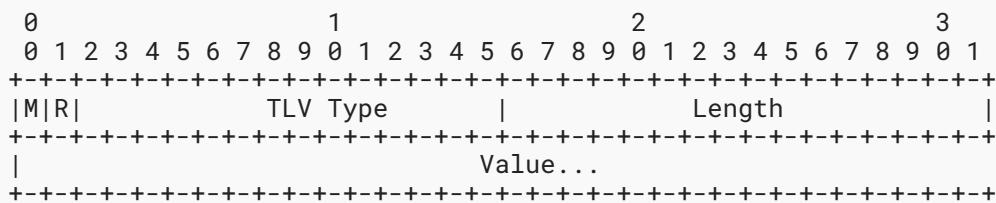
EAP implementations compliant with this specification **MUST** support TLV exchanges as well as the processing of mandatory/optional settings on the TLV. Implementations conforming to this specification **MUST** support the following TLVs:

- 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.

If a peer or server receives a TLV that is not of the correct format, the TLV **MUST** be discarded. It is generally useful to log an error or debugging message that indicates which TLV had an issue and what the problem is. However, TLVs that are malformed are invalid and cannot be used.



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 (DEPRECATED)

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](#))

## 18 CSR-Attributes TLV (Section 4.2.19)

## 19 Identity-Hint TLV (Section 4.2.20)

## Length

The length of the Value field in octets.

### Value

## The value of the TLV.

#### 4.2.2. Authority-ID TLV

M

## 0 - Optional TLV

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.

A server sending the Identity-Type TLV **MUST** also include an EAP-Payload TLV or a Basic-Password-Auth-Resp TLV. A peer sending an Identity-Type TLV **MUST** also include EAP-Payload TLV or a Basic-Password-Auth-Resp TLV.

An EAP peer receiving an Identity-Type request **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.

A server receiving an Identity-Type in the response **MUST** check if the value of the Identity-Type in the response matches the value of the Identity-Type that was sent in the request. A match means that the server can proceed with authentication.

However, if the values do not match, the server can proceed with authentication if and only if the following two conditions match. If either of the following two conditions does not match, the server **MUST** respond with a Result TLV of Failure.

1. The Identity-Type contains a value permitted by the server configuration.
2. The Identity-Type value was not previously used for a successful authentication.

The first condition allows a server to be configured to permit only user authentication, or else only machine authentication. A server could also use an Identity-Hint TLV sent in the response to permit different types of authentication for different identities. A server could also permit or forbid different kinds of authentication based on other information, such as outer EAP Identity, fields in an outer EAP client certificate, or other fields received in a RADIUS or Diameter packet along with the TEAP session. There is no requirement that a server has to support both user and machine authentication for all TEAP sessions.

The second condition ensures that if a particular inner method succeeds, the server does not attempt a subsequent inner method for the same Identity-Type. For example, if a user is authenticated via an inner method of EAP-TLS, there is no benefit to also requesting additional authentication via a different inner method. Similarly, there is no benefit to repeating an authentication sessions for the same user; the result will not change.

This second condition also forbids multiple rounds of challenge/response authentication via the Basic-Password-Auth-Req TLV. TEAPv1 supports only one round of Basic-Password-Auth-Req followed by Basic-Password-Auth-Resp. The result of that round **MUST NOT** be another Basic-Password-Auth-Req TLV.

This second condition also means that a server **MUST NOT** send an Identity-Hint TLV that has the same value as was previously used for a successful authentication.

The Identity-Type TLV is defined as follows:

M

Mandatory, set to one (1)

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 3.6.6 and 3.9.3.

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

A Result TLV indicating success **MUST** be accompanied by a Crypto-Binding TLV.

The Result TLV is defined as follows:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
M R	TLV Type	Length	
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
	Status		
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----

M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

3 - Result TLV

Length

2

## Status

The Status field is two octets. Values include:

- 1 Success
- 2 Failure

## 4.2.5. NAK TLV

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

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
M R	TLV Type	Length	
	Vendor-Id		
	NAK-Type	TLVs...	

M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

4 - 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 3.9.3](#).

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:

0	1	2	3	
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1	
++	++	++	++	
M   R	TLV Type		Length	
++	++	++	++	++
		Error-Code		
++	++	++	++	++

M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

## TLV Type

## 5 - 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]
- 1032 Inner method not supported
- 2001 Tunnel Compromise Error
- 2002 Unexpected TLVs Exchanged
- 2003 The Crypto-Binding TLV is invalid (Version, Received-Ver, or Sub-Type)
- 2004 The first inner method did not derive EMSK
- 2005 The Crypto-Binding TLV did not include a required MSK Compound-MAC
- 2006 The MSK Compound-MAC fails verification
- 2007 The Crypto-Binding TLV did not include a required EMSK Compound-MAC
- 2008 The EMSK Compound-MAC fails verification
- 2009 The EMSK Compound-MAC exists, but the inner method did not derive EMSK

#### 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 TLV

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 particular 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.

Where a Vendor-Specific TLV carries an authentication protocol in the inner method, it **MUST** define values for MSK and EMSK. Where these values cannot be derived from cryptographic primitives, they **MUST** be set to zero, as happens when Basic-Password-Auth-Req is used.

The Vendor-Specific TLV is defined as follows:

0	1	2	3																
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1																
+-----+-----+-----+-----+										+-----+-----+-----+-----+									
M R	TLV Type		Length																
+-----+-----+-----+-----+										+-----+-----+-----+-----+									
	Vendor-Id																		
+-----+-----+-----+-----+										+-----+-----+-----+-----+									
	Vendor TLVs....																		
+-----+-----+-----+-----+										+-----+-----+-----+-----+									

M

0 or 1

R

Reserved, set to zero (0)

**TLV Type**

7 - 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 at any time. The Request-Action TLV allows the peer or server to request that the other side negotiates additional inner methods or process TLVs that are passed inside of the Request-Action TLV.

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. Otherwise, the receiving entity **MAY** send a Request-Action TLV containing an Error TLV of value 2002 (Unexpected TLVs Exchanged).

After processing the TLVs or inner method in the request, another round of Result TLV exchange **MUST** 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 have been processed and the other side has signaled it wants to end the EAP conversation.

The Request-Action TLV is defined as follows:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
M R	TLV Type	Length	
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
Status	Action	TLVs....	
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

8 - Request-Action TLV

Length

2 + cumulative length of all included TLVs

Status

The Status field is one octet. This indicates the result if the party who receives this TLV does not process the action. 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 coalescing 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 signals intermediate Success and Failure messages for all inner methods. The Intermediate-Result TLV **MUST** be used for all inner methods.

An Intermediate-Result TLV indicating success **MUST** be accompanied by a Crypto-Binding TLV.

An Intermediate-Result TLV indicating failure **SHOULD** be accompanied by an Error TLV that indicates why the authentication failed.

The optional TLVs associated with this TLV are provided for future extensibility to provide hints about the current result. The Intermediate-Result TLV is defined as follows:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+	+	+	+
M R	TLV Type	Length	
+	+	+	+
	Status	TLVs...	
+	+	+	+

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

[RFC7170] defined a Protected Access Credential (PAC) to mirror EAP-FAST [RFC4851]. However, implementation experience and analysis determined that the PAC was not necessary. Instead, TEAP performs session resumption using the NewSessionTicket message as defined in Sections 2.1.2 and 2.1.3 of [RFC9190]. As such, the PAC TLV has been deprecated.

As the PAC TLV is deprecated, an entity receiving it **SHOULD** send a Result TLV indicating failure and an Error TLV of Unexpected TLVs Exchanged.

#### 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.

A Crypto-Binding **MUST** be accompanied by an Intermediate-Result TLV indicating success.

The Crypto-Binding TLV **MUST** be exchanged and validated before any Intermediate-Result or Result TLV value is examined, regardless of whether there is an inner method or not. It **MUST** be included with the Intermediate-Result TLV to perform cryptographic binding after each successful inner method in a sequence of inner methods, before proceeding with another inner method. If no MSK or EMSK has been generated and a Crypto-Binding TLV is required, then the MSK Compound-MAC field contains the MAC using keys generated according to [Section 6.3](#).

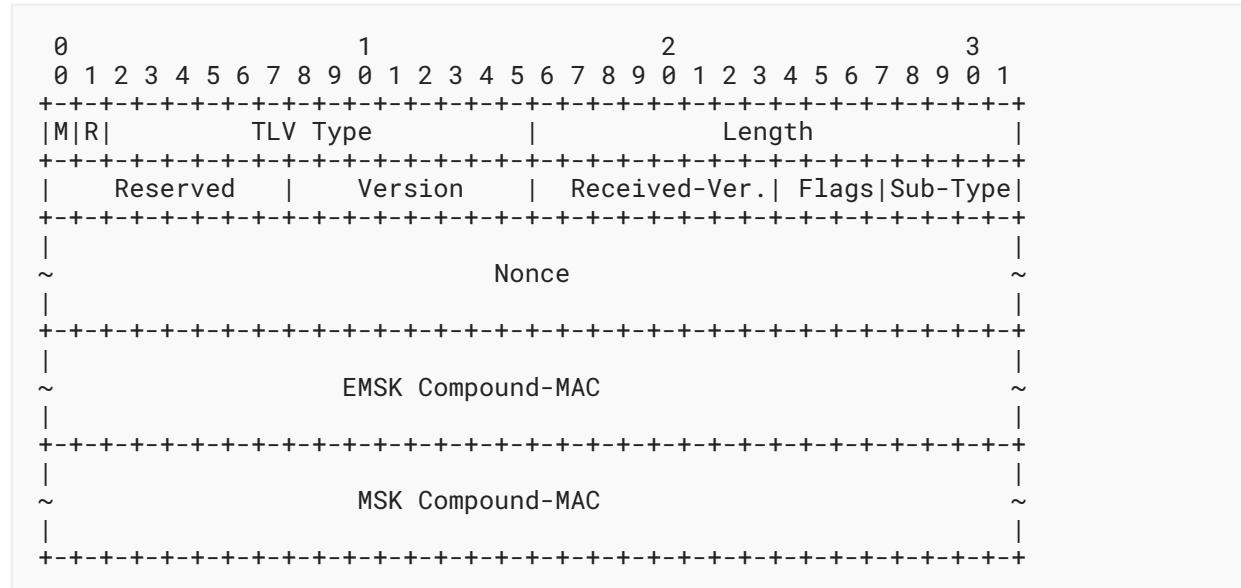
The Crypto-Binding TLV is valid only if the following checks pass on its contents:

- The Version field contain a known value.
- The Received-Ver field matches the TEAP version sent by the receiver during the EAP version negotiation.
- The Sub-Type field is set to the correct value for this exchange.
- The Flags field is set to a known value.
- The Compound-MAC(s) verify correctly.

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 3.9.3](#).

See [Section 6](#) for a more detailed discussion of how the Compound-MAC fields are constructed and verified.

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 TEAPv1, 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.

For TEAPv1, this version number **MUST** be set to one (1).

**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

All other values of the Flags field are invalid.

**Sub-Type**

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

- 0 Binding Request
- 1 Binding Response

All other values of the Sub-Type field are invalid.

**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 6.3](#).

Note that this field is always 20 octets in length. Any larger MAC is simply truncated. All validations or comparisons **MUST** be done on the truncated value.

#### 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 6.3](#).

Note that this field is always 20 octets in length. Any larger MAC is simply truncated. All validations or comparisons **MUST** be done on the truncated value.

#### 4.2.14. Basic-Password-Auth-Req TLV

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

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

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
M R	TLV Type	Length	
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
Prompt ....			
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----

M Mandatory, set to one (1)

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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
++	++	++	++
M R	TLV Type	Length	
++	++	++	++
Userlen	Username		
++	++	++	++
... Username	...		
++	++	++	++
Passlen	Password		
++	++	++	++
... Password	...		
++	++	++	++

M Mandatory, set to one (1)

R Reserved, set to zero (0)

TLV Type 14 - Basic-Password-Auth-Resp TLV

Length variable

Userlen Length of Username field in octets.

The value of Userlen **MUST NOT** be zero.

Username Username in UTF-8 [[RFC3629](#)] format.

The content of Username **SHOULD** follow the guidelines set in [[RFC9427](#)], [Section 3.1](#).

Passlen Length of Password field in octets.

The value of Passlen **MUST NOT** be zero.

Password Password in UTF-8 [[RFC3629](#)] format.

Note that there is no requirement that passwords be humanly readable. Octets in a password may have values less than 0x20, including 0x00.

#### 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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
M R	TLV Type	Length	
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
	PKCS#7 Data...		
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

M 0 - Optional TLV

R Reserved, set to zero (0)

TLV Type 15 - PKCS#7 TLV

Length The length of the PKCS#7 Data field.

PKCS#7 Data This field contains the 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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
M R	TLV Type	Length	
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+
	PKCS#10 Data...		
+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+	+-----+-----+-----+-----+

## M 0 - Optional TLV

R Reserved, set to zero (0)

### TLV Type 16 - PKCS#10 TLV

Length The length of the PKCS#10 Data field.

PKCS#10 Data This field contains the 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 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:

0	1	2	3	
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1	
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	
M R	TLV Type		Length	
----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----
	Credential-Format		Cred TLVs...	
----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----	----- ----- ----- ----- -----

## M 0 - 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.

#### 4.2.19. CSR-Attributes TLV

The CSR-Attributes TLV provides information from the server to the peer on how certificate signing requests should be formed. The purpose of CSR attributes is described in [Section 4.5](#) of [\[RFC7030\]](#). Servers **MAY** send the CSR-Attributes TLV directly after the TLS session has been established. A server **MAY** also send in the same message a Request-Action frame for a PKCS#10 TLV. This is an indication to the peer that the server would like the peer to renew its certificate using the parameters provided in this TLV. Servers shall construct the contents of the CSR-Attributes TLV as specified in [\[RFC7030\]](#), [Section 4.5.2](#) with the exception that the DER encoding **MUST NOT** be encoded in base64. The base64 encoding is used in [\[RFC7030\]](#) because the transport protocol used there requires textual encoding. In contrast, TEAP attributes can transport arbitrary binary data.

Servers and peers **MUST** follow the guidance provided in [\[RFC9908\]](#) when creating the CSR-Attributes TLV. Peers **MAY** ignore the contents of the TLV if they are unable to do so, but then servers may not process PKCS#10 certificate requests for this or any other reason.

The CSR-Attributes TLV is defined as follows:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0 1	
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
M R	TLV Type	Length	
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----
	DER Encoded CSR Attributes		
----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----	----- ----- ----- -----

M 0 - Optional TLV

R Reserved, set to zero (0)

TLV Type 18 - CSR-Attributes

Length >=2 octets

#### 4.2.20. Identity-Hint TLV

The Identity-Hint TLV is an optional TLV that can be sent by the peer to the server at the beginning of the Phase 2 TEAP conversation. The purpose of the TLV is to provide a "hint" as to the identity or identities that the peer will be using by subsequent inner methods.

The purpose of this TLV is to solve the "bootstrapping" problem for the server. In order to perform authentication, the server must choose an inner method. However, the server has no knowledge of what methods are supported by the peer. Without an identity hint, the server needs to propose a method and then have the peer return a response indicating that the requested method is not available. This negotiation increases the number of round trips required for TEAP to conclude with no additional benefit.

When the Identity-Hint is used, the peer can signal which identities it has available, which enables the server to choose an inner method that is appropriate for that identity.

The peer **SHOULD** send an Identity-Hint TLV for each Identity-Type that is available to it. For example, if the peer can do both machine and user authentication, it can send two Identity-Hint TLVs with values "host/name.example.com" (for a machine with hostname "name.example.com") and "user@example.com" (for a person with identity "user@example.com").

The contents of the Identity-Hint TLV **SHOULD** be in the format of an NAI [[RFC7542](#)], but we note that as given in the example above, Machine identities might not follow that format. As these identities are never used for AAA routing as discussed in [[RFC7542](#)], [Section 3](#), the format and definition of these identities are entirely site local. Robust implementations **MUST** support arbitrary data in the content of this TLV, including binary octets.

As the Identity-Hint TLV is a "hint", server implementations are free to ignore the hints given and do whatever is required by site-local policies.

The Identity-Hint TLV is used only as a guide when selecting which inner methods to use. This TLV has no other meaning, and it **MUST NOT** be used for any other purpose. Specifically, server implementations **MUST NOT** compare the identities given this TLV to later identities given as part of the inner methods. There is no issue with the hint(s) failing to match any subsequent identity that is used.

The Identity-Hint TLV **MUST NOT** be used for server unauthenticated provisioning. This TLV is only used as a hint for normal authentication.

The Identity-Hint TLV is defined as follows:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0 1
<pre>+-----+-----+-----+-----+  M R       TLV Type             Length        +-----+-----+-----+-----+            Identity Hint           +-----+-----+-----+-----+</pre>			

## M 0 - Optional TLV

R Reserved, set to zero (0)

## TLV Type 19 - Identity-Hint

Length >=2 octets

### 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 in which TLVs are encoded in a TEAP packet does not matter. However, there is an order in which TLVs in a packet must be processed:

1. Crypto-Binding TLV
2. Intermediate-Result TLV
3. Result TLV or Request-Action TLV
4. Identity-Type TLV
5. EAP-Payload TLV (Identity-Request) or Basic-Password-Auth-Req TLV
6. Other TLVs

That is, cryptographic binding is checked before any result is used and identities are checked before proposing an inner method, as the identity may influence the chosen inner method.

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

Table 1

Outer TLVs **MUST** be marked as optional. Vendor TLVs inside of a 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

Request	Response	Success	Failure	TLVs
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
0-1	0	0	0	CSR-Attributes TLV
0	0+	0	0	Identity-Hint TLV

Table 2

NOTE: Vendor TLVs (included in Vendor-Specific TLVs) sent with a Result TLV **MUST** be marked as optional. Also, the CSR-Attributes TLV is never transmitted by the peer, and so is treated as a request in this table.

## 5. Limitations of TEAPv1

As noted in [Section 1.1](#), TEAPv1 implementations are limited in functionality as compared to what the protocol is theoretically capable of. These limitations mean that only a small number of inner methods are fully supported by existing TEAPv1 implementations.

While [Section 6](#) defines the cryptographic calculations used for key derivation and crypto-binding, this section documents which inner methods are known to work and why those methods work. Other inner methods may work, but those results are likely to be implementation-specific.

We discuss the issues here without naming particular implementations or making any negative inference about them. The implementations work well enough together in limited situations. Any interoperability issues are due to the complexity and incompleteness of the definitions given in [\[RFC7170\]](#) and are not due to issues with any particular implementation.

The interoperability issues are limited to the use and derivation of the Compound-MAC(s), which are derived from the inner MSK and EMSK. In short, implementations are known to derive different values for the Compound-MAC(s) when more than one inner method provides an EMSK.

## 5.1. Interoperable Inner Methods

The following inner methods are known to work. These methods work for both User and Machine credentials.

- EAP-MSCHAPv2
- EAP-TLS

The following combinations of inner methods are known to work. These methods work for any order of User and Machine credentials.

- EAP-MSCHAPv2 followed by EAP-MSCHAPv2
- EAP-TLS followed by EAP-MSCHAPv2

The following combinations of inner methods are known to work when both the supplicant and authenticator ignore the EMSK Compound-MAC field of the Crypto-Binding TLV. These methods work for any order of User and Machine credentials.

- EAP-MSCHAPv2 followed by EAP-TLS
- EAP-TLS followed by EAP-TLS

## 5.2. Explanation and Background

The main reason for the limited set of inner methods is that the most well-known TEAP supplicant supports only EAP-MSCHAPv2 and EAP-TLS for the inner methods. Further, this implementation does not encode the EMSK Compound-MAC field in all of the Crypto-Binding TLVs that it sends and ignores that field in all of the Crypto-Binding TLVs that it receives.

The known authenticator implementations support this client, but any other combination of inner methods was not tested. The result is that due to both the complexity of the cryptographic derivations and the lack of interoperability testing, each authenticator implemented entirely different derivations of the EMSK Compound-MAC field of the Crypto-Binding TLV. This difference was discovered only after multiple implementations had been shipping for years.

## 5.3. Next Steps

Any attempt to change TEAPv1 to address these issues would likely result in one or more implementations being non-compliant with the updated specification. Even worse, updates to this specification would result in multiple incompatible versions of TEAPv1.

That approach is not acceptable.

In the interest of maintaining known interoperability, this specification simply documents these issues rather than trying to correct the problem. Since the TEAP protocol and the Crypto-Binding TLV both contain a Version field, the better path forward is to publish this specification while documenting the large caveats for TEAPv1. Any changes to the TEAP protocol can then be done in a future TEAPv2 specification.

## 6. Cryptographic Calculations

The definitions given in this section are known to work with all implementations but only for a few inner methods, as described above in [Section 5](#). In the interest of avoiding additional complexity in an already complex process, those definitions are given as if they work for all possible inner methods.

We note that some interoperable implementations have been written based on these definitions, which support inner methods other than EAP-MSCHAPv2 and EAP-TLS. It is therefore useful to document the full operation of TEAPv1 despite the known issues. We only caution implementors that inner methods that are not listed above in [Section 5](#) are likely to work with only a subset of existing TEAPv1 implementations.

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 cipher suite, TEAP implementations need a mechanism to determine the version and cipher suite in use for a particular session. The implementation can then use this information to determine which PRF and MAC algorithm to use.

### 6.1. TEAP Authentication Phase 1: Key Derivations

With TEAPv1, the TLS master secret is generated as specified in TLS. If session resumption 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 as follows:

```
session_key_seed = TLS-Exporter(  
    "EXPORTER: teap session key seed", , 40)
```

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 [\[RFC5246\]](#).

### 6.2. Intermediate Compound Key Derivations

As TEAP can run multiple inner methods, there needs to be a way to cryptographically bind each inner method to the TLS tunnel and to cryptographically bind each method to the previous one. This binding is done by deriving a number of intermediate keys and exchanging that information in the Crypto-Binding TLV.

The key derivation is complicated by a number of factors. An inner method can derive an MSK or (as with basic passwords) not derive an MSK. An inner method can derive an EMSK or perhaps not derive an EMSK, or some EAP types may derive different EMSKs for the peer and the server. All of these cases must be accounted for and have recommendations made for how peers and servers can interoperate.

There are a number of intermediate keys used to calculate the final MSK and EMSK for TEAP. We give a brief overview here in order to clarify the detailed definitions and derivations given below. As each inner method can derive an MSK (or not) and an EMSK (or not), there need to be separate intermediate key calculations for MSK and for EMSK. For the purposes of this overview, we just describe the derivations at a high level and state that the MSK/EMSK issue is addressed in the more detailed text below.

For each inner method, we derive an IMSK. This key depends on the inner key (MSK or EMSK). This IMSK is then tied to the TLS session via the TLS-PRF to derive an Inner Method Compound Key (IMCK). The IMCK is used to generate a Compound-MAC key (CMK). The CMK is mixed with various data from the TEAP negotiation to create Compound-MAC field of the Crypto-Binding attribute. This TLV cryptographically binds the inner method to the protected tunnel and to the other fields that have been negotiated. The cryptographic binding prevents on-path attacks.

The IMCK for this inner method is then mixed with keys from previous inner methods, beginning with the TEAP Phase 2 session\_key\_seed derived above, to yield a Secure IMCK (S-IMCK) for this round. The S-IMCK from the final is then used to derive the MSK and EMSK for TEAP.

We differentiate keys for inner methods by counting inner methods starting from 0 and use an index "j" to refer to an arbitrary inner method. For example, IMCK[0] is the IMCK for the first, or "0" inner method. While TEAPv1 is currently limited to one or two inner methods (j=0 or j=0,1), further updates could allow for more inner method exchanges.

### 6.2.1. Generating the Inner Method Session Key

Each inner method generates an IMSK that depends on the EMSK (preferred) or the MSK if it exists, or else it is all zeros. We refer to the IMSK for inner method "j" as IMSK[j].

If an inner method supports export of an EMSK, then the IMSK **SHOULD** be derived from the EMSK, which is defined in [\[RFC5295\]](#). The optional data parameter is not used in the derivation.

The above derivation is not a requirement, as some peer implementations of TEAP are also known to not derive IMSK from EMSK and to only derive IMSK from MSK. In order to be compatible with those implementations, the use of EMSK here is not made mandatory.

Some EAP methods may also have the peer and server derive different EMSKs. Mandating an EMSK-based derivation there would prevent interoperability, as the Crypto-Binding TLV contents that depend on EMSK could not then be validated by either side. Those methods **SHOULD NOT** derive IMSK from EMSK unless the method has a way to negotiate how the EMSK is derived, along with a way to signal that both the peer and server have derived the same EMSK.

It is **RECOMMENDED** that for those EAP methods, implementations take the simpler approach of ignoring EMSK and always derive IMSK from MSK. This approach is less secure, as IMSK no longer cryptographically binds the inner method to the TLS tunnel. A better solution is to suggest that deployments of TEAP **SHOULD** avoid such methods.

The derivation of IMSK[j] from the j'th EMSK is given as follows:

```
IMSK[j] = First 32 octets of TLS-PRF(
  EMSK[j],
  "TEAPbindkey@ietf.org",
  0x00 | 0x00 | 0x40)
```

Where:

- " | " denotes concatenation
- The TLS-PRF is defined in [\[RFC5246\]](#) as:

```
PRF(secret, label, seed) = P_<hash>(secret, label | seed)
```

- The secret is the EMSK from the j'th inner method, the usage label used is "TEAPbindkey@ietf.org" consisting of the ASCII value for the label "TEAPbindkey@ietf.org" (without quotes), and the seed consists of the "\0" null delimiter (0x00) and 2-octet unsigned integer length of 64 octets in network byte order (0x00 | 0x40) specified in [\[RFC5295\]](#).

If an inner method does not support the export of EMSK but does export MSK, then the IMSK is copied from the MSK of the inner method. If the MSK is longer than 32 octets, the IMSK is copied from the first 32 octets and the rest of MSK is ignored. If the MSK is shorter than 32 octets, then the IMSK is copied from MSK and the remaining data in IMSK is padded with zeros to a length of 32 octets. IMSK[j] is then this derived value.

If the inner method does not provide either MSK or EMSK, such as when basic password authentication is used or when no inner method has been run, then both MSK and IMSK[j] are set to all zeroes (i.e., IMSK[j] = MSK = 32 octets of 0x00s).

Note that using an MSK of all zeroes opens up TEAP to on-path attacks as discussed in [Section 8.3](#). It is therefore **NOT RECOMMENDED** to use inner methods that fail to generate an MSK or EMSK. These methods should only be used in conjunction with another inner method that does provide for MSK or EMSK generation.

It is also **RECOMMENDED** that TEAP peers order inner methods such that methods that generate EMSKs are performed before methods that do not generate EMSKs. Ordering inner methods in this manner ensures that the first inner method detects any on-path attackers, and any subsequent inner method used is therefore secure.

For example, Phase 2 could perform both machine authentication using EAP-TLS, followed by user authentication via the Basic Password Authentication TLVs. In that case, the use of EAP-TLS would allow an attacker to be detected before the users' password was sent.

However, it is 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.

### 6.2.2. Generating S-IMCK

Once  $\text{IMSK}[j]$  has been determined, it is mixed via the TLS-PRF with the key  $\text{S-IMCK}[j-1]$  from a previous round. That mixing derives a new key  $\text{IMCK}[j]$ . This key is then used to derive both  $\text{S-IMCK}[j]$  for this round and  $\text{CMK}[j]$  for this round.

The derivation of S-IMCK is as follows:

```

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

where TLS-PRF is the PRF (described above) negotiated as part of TLS handshake [RFC5246]. The value  $j$  refers to a corresponding inner method 1 through  $n$ . The special value of  $\text{S-IMCK}[0]$  is used to bootstrap the calculations and can be done as soon as the TLS connection is established and before any inner methods are run.

In practice, the requirement to use either MSK or EMSK means that an implement **MUST** track two independent derivations of  $\text{IMCK}[j]$ , one that depends on the MSK, and another that depends on EMSK. That is, we have both values derived from MSK:

- $\text{IMSK\_MSK}[j]$
- $\text{S-IMCK\_MSK}[j]$
- $\text{CMK\_MSK}[j]$

and then also values derived from EMSK:

- $\text{IMSK\_EMSK}[j]$
- $\text{S-IMCK\_EMSK}[j]$
- $\text{CMK\_EMSK}[j]$

At the conclusion of a successful exchange of Crypto-Binding TLVs, a single  $\text{S-IMCK}[j]$  is selected based on which Compound-MAC value was included in the Crypto-Binding TLV from the client. If EMSK Compound-MAC was included,  $\text{S-IMCK}[j]$  is taken from  $\text{S-IMCK\_EMSK}[j]$ . Otherwise,  $\text{S-IMCK}[j]$  is taken from  $\text{S-IMCK\_MSK}[j]$ .

### 6.2.3. Choosing Inner Methods Securely

In order to further secure TEAP, implementations can take steps to increase their security by carefully ordering inner methods. Where multiple inner methods are used, implementations **SHOULD** choose an ordering so that the first inner method used is one that derives EMSK.

For an EAP server, it can select the first inner method to be one that derives EMSK. Since ordering of inner methods is not otherwise important in EAP, any chosen order is supported by the peer that receives this request.

For an EAP peer, it can choose its response to a server's request for a particular type of authentication. The peer can ignore that request and return an inner method that derives EMSK. Again, since the ordering of inner methods is not otherwise important in EAP, any chosen order is supported by the server that receives this response. Once the more secure authentication has succeeded, the server then requests the other type of authentication and the peer can respond with the chosen type of authentication.

Implementations can also provide configuration flags, policies, or documented recommendations that control the type of inner methods used or verify their order. These configurations allow implementations and administrators to control their security exposure to on-path attackers.

Implementations can permit administrators to configure TEAP so that the following security checks are enforced:

- Verifying that the first inner method used is one that derives EMSK. If this is not done, a fatal error can be returned.
- Verifying that if any inner method derives EMSK, the received Crypto-Binding TLV for that method contains an EMSK Compound-MAC. If an EMSK has been derived and no EMSK Compound-MAC is seen, a fatal error can be returned.

The goal of these suggestions is to enforce the use of the EMSK Compound-MAC to protect the TEAP session from on-path attackers. If these suggestions are not enforced, then the TEAP session is vulnerable.

Most of these suggestions are not normative, as some existing implementations are known to not follow them. Instead, these suggestions are here to inform new implementors, along with administrators, of the issues surrounding this subject.

### 6.2.4. Managing and Computing Crypto-Binding

After an inner method has been completed successfully and the inner keys have been derived, the server sends a Crypto-Binding TLV to the peer. If the inner method has failed, the server does not send a Crypto-Binding TLV.

The peer verifies the Crypto-Binding TLV by applying the rules defined in [Section 4.2.13](#). If verification passes, the peer responds with its own Crypto-Binding TLV, which the server in turn verifies. If at any point verification fails, the party that makes this determination terminates the session.

The Crypto-Binding TLV is normally sent in conjunction with other TLVs that indicate intermediate or final results or that begin negotiation of a new inner method. This negotiation does not otherwise affect the Crypto-Binding TLV.

While [Section 4.2.13](#) defines that the Compound-MAC fields exist in the Crypto-Binding TLV, it does not describe the derivation and management of those fields. This derivation is complex and is therefore located here along with the other key derivations.

The following text defines how the server and peer compute, send, and then verify the Compound-MAC fields Crypto-Binding TLV. Depending on the inner method and site policy, the Crypto-Binding TLV can contain only an MSK Compound-MAC (Flags=2), only the EMSK Compound-MAC (Flags=2), or both Compound-MACs (Flags=3). Each party to the TEAP session follows its own set of procedures to compute and verify the Compound-MAC fields.

The determination of the contents of the Crypto-Binding TLV is done separately for each inner method. If at any point the verification of a Compound-MAC fails, the determining party returns a fatal error as described in [Section 3.9.3](#).

We presume that each peer and server have site policies that require (or not) the use of the MSK Compound-MAC and/or the EMSK Compound-MAC. These policies can be enforced globally for all inner methods, or they can be enforced separately on each inner method. These policies could be enabled automatically when the EAP method is known to always generate an EMSK and could otherwise be configurable.

The server initiates crypto binding by determining which Compound-MAC(s) to use, computing their value(s), placing the resulting Compound-MAC(s) into the Crypto-Binding TLV, and then sending it to the peer.

Then, the steps taken by the server are as follows:

- If the inner method is known to generate only MSK, or if the server's policy is to not use EMSK Compound-MACs:
  - The server computes the MSK Compound-MAC using the MSK of the inner method. The server does not use the EMSK Compound-MAC field (Flags=2).

Otherwise, the EMSK is available.

- If the server's policy permits the use of the MSK Compound-MAC:
  - The sender computes the MSK Compound-MAC along with the EMSK Compound-MAC (Flags=3).

Otherwise, the server's policy does not allow the use of the MSK Compound-MAC:

- The server computes only the EMSK Compound-MAC (Flags=1).

The peer verifies the Crypto-Binding TLV it receives from the server. It then replies with its own crypto binding response by determining which Compound-MAC(s) to use, computing their value(s), placing the resulting Compound-MAC(s) into the Crypto-Binding TLV, and then sending it to the server. The result of this process is either a fatal error or one or more Compound-MACs that are placed in the Crypto-Binding TLV and sent to the server.

Then, the steps taken by the peer are as follows:

- If the peer site policy requires the use of the EMSK Compound-MAC:
  - The peer checks if the Flags field indicates the presence of the EMSK Compound MAC (Flags=1 or 3). If the Flags field has any other value, the peer returns a fatal error.
  - The peer checks if the inner method has derived an EMSK. If not, the peer returns a fatal error.

Otherwise, the peer site policy does not require the use of the EMSK Compound-MAC and the EMSK may or may not exist.

- If the inner method is known to generate only MSK and not EMSK:
  - The peer checks if the Flags field indicates that only the MSK Compound-MAC exists (Flags=2). If the Flags field has any other value, the peer returns a fatal error.

Otherwise, the MSK exists, the EMSK may or may not exist, and the peer allows the use of the EMSK Compound-MAC. The peer may have received one or two Compound-MACs (Flags=1,2,3). Any Compound-MAC that is present is verified. No further action is taken by the peer if a particular Compound-MAC is not present. No further action is taken by the peer if an unexpected Compound-MAC is present.

Note that due to earlier validation of the Flags field ([Section 4.2.13](#)), at least one Compound-MAC must now exist (Flags=1,2,3).

- If the peer has received an MSK Compound-MAC, it verifies it and returns a fatal error if verification fails.
- If EMSK is available and the peer has received an EMSK Compound-MAC, it verifies it and returns a fatal error if verification fails.

The peer creates a crypto binding response by determining which Compound-MAC(s) to use, computing their value(s), placing the resulting Compound-MAC(s) into the Crypto-Binding TLV, and then sending it to the server.

The steps taken by the peer are then as follows.

- If the peer received an MSK Compound-MAC from the server:
  - Since the MSK always exists, this step is always possible. The peer computes the MSK Compound-MAC for the response (Flags=2).

- If the peer site policy requires the use of the EMSK Compound-MAC:
  - The preceding steps taken by the peer ensures that the EMSK exists and the server had sent an EMSK Compound-MAC. The peer computes the EMSK Compound-MAC for the response. The Flags field is updated (Flags=1,3).

Otherwise, if the EMSK exists:

- The peer computes the EMSK Compound-MAC for the response. The Flags field is updated (Flags=1,3).

The server processes the response from the peer via the following steps:

- If the server site policy requires the use of the EMSK Compound-MAC:
  - The server checks if the Flags field indicates the presence of the EMSK Compound MAC (Flags=1 or 3). If the Flags field has any other value, the server returns a fatal error.
  - The server checks if the inner method has derived an EMSK. If not, the server returns a fatal error.
- If the inner method is known to generate only MSK and not EMSK:
  - The server checks if the Flags field indicates that only the MSK Compound-MAC exists (Flags=2). If the Flags field has any other value, the server returns a fatal error.

Otherwise, the MSK exists and the EMSK may or may not exist. The server may have received one or two Compound-MACs (Flags=1,2,3). Any Compound-MAC that is present is verified. No further action is taken by the server if a particular Compound-MAC is not present. No further action is taken by the server if an unexpected Compound-MAC is present.

- If the server has received an MSK Compound-MAC, it verifies it and returns a fatal error if verification fails.
- If EMSK is available and the server has received an EMSK Compound-MAC, it verifies it and returns a fatal error if verification fails.

Once the above steps have concluded, the server either continues authentication with another inner method or it returns a Result TLV.

#### 6.2.5. Unintended Side Effects

In earlier drafts of this document, the descriptions of the key derivations had issues that were only discovered after TEAP had been widely implemented. These issues need to be documented in order to enable interoperable implementations.

As noted above, some inner EAP methods derive MSK but do not derive EMSK. When there is no EMSK, it is therefore not possible to derive IMCK\_EMSK[j] from it. The choice of multiple implementations was then to simply define:

IMCK\_EMSK[j] = IMCK\_EMSK[j - 1]

This definition can be trivially implemented by simply keeping a cached copy of IMCK\_EMSK in a data structure. If EMSK is available, IMCK\_EMCK is updated from it via the TLS-PRF function as defined above. If EMSK is not available, then the IMCK\_EMSK value is unmodified.

This behavior was not explicitly anticipated by earlier drafts of this document. It instead appears to be an accidental outcome of implementing the derivations above with the limitation of a missing EMSK. This behavior is explicitly called out here in the interest of fully documenting TEAP.

Another unintended consequence is in the calculation of the Crypto-Binding TLV. That TLV includes compound MACs that depend on the MSK and EMSK of the current authentication method. Where the current method does not provide an EMSK, the Crypto-Binding TLV does not include a compound MAC that depends on the EMSK. Where the current method does not provide an MSK, the Crypto-Binding TLV includes a compound MAC that depends on a special "all zero" IMSK as discussed earlier.

The result of this definition is that the final Crypto-Binding TLV in an inner TEAP exchange may not include a compound MAC that depends on EMSK, even if earlier EAP methods in the Phase 2 exchange provided an EMSK. This result likely has negative effects on security, though the full impact is unknown at the time of writing this document.

These design flaws have nonetheless resulted in multiple interoperable implementations. We note that these implementations seem to support only EAP-TLS and the EAP-FAST-MSCHAPv2 variant of EAP-MSCHAPv2. Other inner EAP methods may work by accident but are not likely to work by design. For this document, we can only ensure that the behavior of TEAPv1 is fully documented, even if that behavior was an unintended consequence of unclear text in earlier versions of this document.

We expect that these issues will be addressed in a future revision of TEAP.

### 6.3. Computing the Compound-MAC

For inner methods that generate keying material, further protection against on-path 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:

```
Compound-MAC = the first 20 octets of MAC( CMK[n], BUFFER )
```

where  $n$  is the number of the last successfully executed inner method,  $MAC$  is the  $MAC$  function negotiated in  $TLS$  (e.g.,  $TLS$  1.2 in [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**, which **MUST** be **TEAP**, encoded as one octet of **0x37**.
3. All the **Outer TLVs** from the first **TEAP message** sent by the **EAP server** to the **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.

If no inner method is run, then no **MSK** or **EMSK** will be generated. If an **IMSK** needs to be generated, then the **MSK** and therefore the **IMSK** is set to all zeroes (i.e.,  $IMSK = MSK = 32$  octets of **0x00s**).

Note that there is no boundary marker between the fields in steps (3) and (4). However, the server calculates the compound  $MAC$  using the outer **TLVs** it sent and the outer **TLVs** it received from the **peer**. On the other side, the **peer** calculates the compound  $MAC$  using the outer **TLVs** it sent and the outer **TLVs** it received from the **server**. As a result, any modification in transit of the outer **TLVs** will be detected because the two sides will calculate different values for the compound  $MAC$ .

If no key-generating inner method is run, then no **MSK** or **EMSK** will be generated. If an **IMSK** needs to be generated, then the **MSK** and therefore the **IMSK** is set to all zeroes (i.e.,  $IMSK = MSK = 32$  octets of **0x00s**)

#### 6.4. EAP Master Session Key Generation

**TEAP** authentication assures the **MSK** and **EMSK** output from running **TEAP** are the combined result of all inner methods by generating an **IMCK**. The **IMCK** is mutually derived by the **peer** and the **server** as described in [Section 6.2](#) by combining the **MSKs** from inner methods with key material from **TEAP Phase 1**. The resulting **MSK** and **EMSK** are generated from the final (" $n$ "th) inner method, as part of the **IMCK[n]** key hierarchy via the following derivation:

```

MSK  = the first 64 octets of TLS-PRF(S-IMCK[n],
    "Session Key Generating Function")
EMSK = the first 64 octets of TLS-PRF(S-IMCK[n],
    "Extended Session Key Generating Function")

```

The secret is **S-IMCK[n]**, where  $n$  is the number of the last generated **S-IMCK[j]** from [Section 6.2](#). The label is the ASCII value for the string without quotes. The seed is empty (0 length) and is omitted from the derivation.

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 inner method has created an MSK or EMSK, the MSK and EMSK will be generated directly from the session\_key\_seed meaning S-IMCK[0] = session\_key\_seed.

As we noted above, not all inner methods generate both MSK and EMSK, so we have to maintain two independent derivations of S-IMCK[j], one for each of MSK[j] and EMSK[j]. The final derivation using S-IMCK[n] must choose only one of these keys.

If the Crypto-Binding TLV contains an EMSK compound MAC, then the derivation is taken from the S-IMCK\_ESMSK[n]. Otherwise, it is taken from the S-IMCK\_MSK[n].

## 7. 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 [[RFC8126](#)].

Except as noted below, IANA has updated the "Tunnel Extensible Authentication Protocol (TEAP) Parameters" registry to change the Reference field in all tables from [[RFC7170](#)] to RFC 9930.

### 7.1. TEAP TLV Types

IANA has updated the references in the "TEAP TLV Types" registry from [[RFC7170](#)] to RFC 9930 and added TLV 18 and TLV 19 to the registry. The Unassigned values then begin at 20 instead of at 18.

Value	Description	Reference
18	CSR-Attributes TLV	RFC 9930
19	Identity-Hint TLV	RFC 9930
20-16383	Unassigned	

Table 3

IANA has closed the "TEAP PAC TLV (value 11) PAC Attribute Type Codes" and "TEAP PAC TLV (value 11) PAC-Type Type Codes" registries to new registrations and updated those registries with the following note:

This registry has been closed. See RFC 9930.

## 7.2. TEAP Error TLV (value 5) Error Codes

IANA has updated the "TEAP Error TLV (value 5) Error Codes" registry to add the following entries:

Value	Description	Reference
1032	Inner method not supported	RFC 9930
2003	The Crypto-Binding TLV is invalid (Version, or Received-Ver, or Sub-Type)	RFC 9930
2004	The first inner method did not derive EMSK	RFC 9930
2005	The Crypto-Binding TLV did not include a required MSK Compound-MAC	RFC 9930
2006	The MSK Compound-MAC fails verification	RFC 9930
2007	The Crypto-Binding TLV did not include a required EMSK Compound-MAC	RFC 9930
2008	The EMSK Compound-MAC fails verification	RFC 9930
2009	The EMSK Compound-MAC exists, but the inner method did not derive EMSK	RFC 9930

Table 4

## 7.3. TLS Exporter Labels

IANA has updated the "TLS Exporter Labels" registry to change the Reference field for Value "EXPORTER: teap session key seed" as follows:

Value	DTLS-OK	Recommended	Reference
EXPORTER: teap session key seed	N	Y	RFC 9930

Table 5

## 7.4. Extended Master Session Key (EMSK) Parameters

IANA has updated the "User Specific Root Keys (USRK) Key Labels" registry to change the Reference field for Value "TEAPbindkey@ietf.org" as follows:

Label	Description	Reference
TEAPbindkey@ietf.org	TEAP binding usage label	RFC 9930

Table 6

## 7.5. Extensible Authentication Protocol (EAP) Registry

IANA has updated the "Method Types" registry to change the Reference field for Value "55" as follows:

Value	Description	Reference
55	TEAP	RFC 9930

Table 7

## 8. 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].

### 8.1. Mutual Authentication and Integrity Protection

As a whole, TEAP provides message and integrity protection by establishing a secure tunnel for protecting the inner 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 inner 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.

Where server unauthenticated provisioning is performed, TEAP requires that the inner provisioning method provide for both peer and server authentication.

### 8.2. Method Negotiation

As is true for any negotiated EAP protocol, EAP NAK messages used to suggest an alternate EAP 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 bid-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 methods, so TEAP should be selected when available.

An attacker cannot readily determine the inner 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.

### 8.3. 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
- On-path active 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 TEAP servers and Inner servers allows for crypto-binding based on the inner method MSK to be thwarted as described in [\[RFC7029\]](#). If the inner method derives an EMSK, then this threat is mitigated as TEAP uses the Crypto-Binding TLV to tie the inner EMSK to the TLS session via the TLS-PRF, as described above in [Section 6](#).

On the other hand, if the inner method is not deriving EMSK, as with password authentication or unauthenticated provisioning, then this threat still exists. Implementations therefore need to limit the use of inner methods as discussed above in [Section 3.6.5](#)

### 8.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies

TEAP addresses the known deficiencies and weaknesses in some EAP authentication methods. 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 inner method negotiation, including EAP method

- Sequencing of inner methods, including EAP methods
- Strong mutually derived MSKs
- Acknowledged success/failure indication
- Faster re-authentications through session resumption
- Mitigation of offline dictionary attacks
- Mitigation of on-path 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 inner methods that typically rely on weak credentials, such as password-based secrets. To that extent, the TEAP authentication mitigates several vulnerabilities, such as offline dictionary attacks, by protecting the weak credential-based inner 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.

#### 8.4.1. User Identity Protection and Verification

The initial identity request response exchange is sent in cleartext outside the protection of TEAP. Typically, the NAI [\[RFC7542\]](#) 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.

When the server is authenticated, identity request/response exchanges sent after the TEAP tunnel is established are protected from modification and eavesdropping by attackers. For server unauthenticated provisioning, the outer TLS session provides little security, and the provisioning method must provide this protection instead.

When a client certificate is sent outside of the TLS tunnel in Phase 1, the peer **MUST** include Identity-Type as an outer TLV in order to signal the type of identity which that client certificate is for. Further, when a client certificate is sent outside of the TLS tunnel, the server **MUST** proceed with Phase 2. If there is no Phase 2 data, then the EAP server **MUST** reject the session.

Issues related to confidentiality of a client certificate are discussed above in [Section 3.4.1](#)

Note that the Phase 2 data could simply be a Result TLV with value Success, along with a Crypto-Binding TLV. This Phase 2 data serves as a protected success indication as discussed in [RFC190], [Section 2.1.1](#).

## 8.5. Dictionary Attack Resistance

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

TEAP mitigates dictionary attacks by permitting inner methods, such as EAP-pwd, that are not vulnerable to dictionary attacks.

TEAP implementations can mitigate online "brute force" dictionary attempts by limiting the number of failed authentication attempts for a particular identity.

### 8.5.1. Protection Against On-Path Attacks

TEAP provides protection from on-path attacks in a few ways:

1. By using a certificates or a session ticket to mutually authenticate the peer and server during TEAP authentication Phase 1 establishment of a secure TLS tunnel.
2. When the TLS tunnel is not secured, by using the keys generated by the inner method (if the inner methods are key generating) in the crypto-binding exchange and in the generation of the key material exported by the inner method described in [Section 6](#).

TEAP crypto binding does not guarantee protection from on-path attacks 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 cipher suite derives the master secret solely from the contribution of secret data from one side of the conversation (such as cipher suites 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 cipher suite derives the master secret from the contribution of secrets from both sides of the conversation (such as in cipher suites based on Diffie-Hellman), then crypto binding can detect an attacker in the conversation if a strong inner method is used.

TEAP crypto binding does not guarantee protection from on-path attacks when the client does not verify the server, and the inner method does not produce an EMSK. The only way to close this vulnerability is to define TEAPv2, which would then have different crypto binding derivations.

## 8.6. 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 3.6.6](#).

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.

## 8.7. Use of Cleartext Passwords

TEAP can carry cleartext passwords in the Basic-Password-Auth-Resp TLV. Implementations should take care to protect this data. For example, passwords should not normally be logged, and password data should be securely scrubbed from memory when it is no longer needed.

## 8.8. Accidental or Unintended Behavior

Due to the complexity of TEAP, and the long time between [\[RFC7170\]](#) and any substantial implementation, there are many accidental or unintended behaviors in the protocol.

The first one is that EAP-FAST-MSCHAPv2 is used instead of EAP-MSCHAPv2. While [\[RFC7170\]](#) defined TEAP to use EAP-MSCHAPv2, an early implementor or implementors instead used EAP-FAST-MSCHAPv2. The choice for this document was either to define a new version of TEAP that used EAP-MSCHAPv2 or instead to document implemented behavior. The choice taken here was to document running code.

The issues discussed in [Section 6.2.5](#) could have security impacts, but no analysis has been performed. The choice of using a special "all zero" IMSK in [Section 6.2](#) was made for simplicity but could also have negative security impacts.

The definition of the Crypto-Binding TLV means that the final Crypto-Binding TLV values might not depend on all previous values of MSK and EMSK. This limitation could have negative security impacts, but again, no analysis has been performed.

We suggest that the TEAP protocol be revised to TEAP version 2, which could address these issues. There are proposals at this time to better derive the various keying materials and cryptographic binding derivations. However, in the interest of documenting running code, we are publishing this document with the acknowledgment that there are improvements to be made.

## 8.9. Implicit Challenge

Certain authentication protocols that use a challenge/response mechanism rely on challenge material that is not generated by the authentication server; therefore, the material may require special handling. For EAP-TTLS, these challenges are defined in [[RFC5281](#)], [Section 11.1](#).

In EAP-MSCHAPv2, the authenticator issues a challenge to the supplicant. Then, the supplicant hashes the challenge with the password and forwards the response to the authenticator. The response also includes a Peer-Challenge, which is created by the supplicant. Since the challenge is random, it is not associated with the TLS tunnel and the protocol may be susceptible to a replay attack.

The Crypto-Binding TLV provides protection against intermediaries, but it does not provide protection against a replay attack. We suggest that any TEAPv2 specification correct this issue.

## 8.10. 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.

Cipher Suite 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. Server unauthenticated provisioning provides its own protection mechanisms.

Replay protection: Yes

Confidentiality: Yes

Key derivation: Yes

Key strength: See Note 1 below.

Dictionary attack prot.: See Note 2 below.

Fast reconnect: Yes

Cryptographic binding: Yes

Session independence: Yes

Fragmentation: Yes

Key Hierarchy: Yes

Channel binding: Yes

Notes:

- Note 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 6](#) advises use of the following required RSA or Diffie-Hellman (DH) module and Digital Signature Algorithm (DSA) subgroup size in bits for a given level of attack resistance in bits. Based on the table below, a 2048-bit RSA key is required to provide 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

*Table 8*

- Note 2. TEAP protects against offline dictionary attacks when secure inner methods are used. TEAP protects against online dictionary attacks by limiting the number of failed authentications for a particular identity.

## 9. Changes from RFC 7170

Alan DeKok was added as an editor.

The document was converted to Markdown from the [RFC7170] text output.

Any formatting changes mostly result from differences between using Markdown versus XML for source.

The IANA Considerations section was replaced with a note to change the IANA registry references to this document.

A new section was added to explain that the inner EAP-MSCHAPv2 derivation follows EAP-FAST. This is the largest technical change from the previous revision of this document and follows existing implementations.

Many small changes have been made throughout the document to correct inconsistencies and to address mistakes. At a high level:

- All open errata have been addressed.
- A new term "inner method" has been defined.
- The definitions and derivation of IMSK, S-IMCK, etc. have been corrected and clarified.
- The diagrams in [Appendix C](#) have been updated to match the TEAP state machine.

All uses of the PAC were removed. It had not been implemented, and there were no plans by implementors to use it.

Text was added on recommendations for inner and outer identities.

[Section 6.2.5](#) was added late in the document life cycle in order to document accidental behavior that could result in interoperability issues.

## 10. References

### 10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <<https://www.rfc-editor.org/info/rfc2985>>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<https://www.rfc-editor.org/info/rfc2986>>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.

**[RFC5077]** Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <<https://www.rfc-editor.org/info/rfc5077>>.

**[RFC5216]** Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <<https://www.rfc-editor.org/info/rfc5216>>.

**[RFC5246]** Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.

**[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, DOI 10.17487/RFC5295, August 2008, <<https://www.rfc-editor.org/info/rfc5295>>.

**[RFC5705]** Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.

**[RFC5746]** Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <<https://www.rfc-editor.org/info/rfc5746>>.

**[RFC5929]** Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.

**[RFC6677]** Hartman, S., Ed., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, DOI 10.17487/RFC6677, July 2012, <<https://www.rfc-editor.org/info/rfc6677>>.

**[RFC7030]** Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

**[RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

**[RFC8446]** Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

**[RFC8996]** Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <<https://www.rfc-editor.org/info/rfc8996>>.

**[RFC9190]** Preuß Mattsson, J. and M. Sethi, "EAP-TLS 1.3: Using the Extensible Authentication Protocol with TLS 1.3", RFC 9190, DOI 10.17487/RFC9190, February 2022, <<https://www.rfc-editor.org/info/rfc9190>>.

- [RFC9427] DeKok, A., "TLS-Based Extensible Authentication Protocol (EAP) Types for Use with TLS 1.3", RFC 9427, DOI 10.17487/RFC9427, June 2023, <<https://www.rfc-editor.org/info/rfc9427>>.
- [RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <<https://www.rfc-editor.org/info/rfc9525>>.
- [RFC9908] Richardson, M., Ed., Friel, O., von Oheimb, D., and D. Harkins, "Clarification and Enhancement of the CSR Attributes Definition in RFC 7030", RFC 9908, DOI 10.17487/RFC9908, January 2026, <<https://www.rfc-editor.org/info/rfc9908>>.

## 10.2. Informative References

- [IEEE.802.1X.2020] IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Port-Based Network Access Control", IEEE Std 802.1X-2020, DOI 10.1109/IEEEESTD.2020.9018454, February 2020, <<https://doi.org/10.1109/IEEEESTD.2020.9018454>>.
- [KAMATH] Kamath, V. and A. Palekar, "Microsoft EAP CHAP Extensions", Work in Progress, Internet-Draft, draft-kamath-pppext-eap-mschapv2-02, 19 June 2007, <<https://datatracker.ietf.org/doc/html/draft-kamath-pppext-eap-mschapv2-02>>.
- [MSCHAP] Microsoft Corporation, "Master Session Key (MSK) Derivation", 23 April 2024, <[https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-chap/5a860bf5-2aeb-485b-82ee-fac1e8e6b76f](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-chap/5a860bf5-2aeb-485b-82ee-fac1e8e6b76f)>.
- [NIST-SP-800-57] Barker, E., "Recommendation for Key Management: Part 1 - General", NIST SP 800-57 Part 1 Rev. 5, DOI 10.6028/NIST.SP.800-57pt1r5, May 2020, <<https://doi.org/10.6028/NIST.SP.800-57pt1r5>>.
- [PEAP] Microsoft Corporation, "[MS-PEAP]: Protected Extensible Authentication Protocol (PEAP)", 24 June 2021, <[https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-peap/5308642b-90c9-4cc4-beec-fb367325c0f9](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-peap/5308642b-90c9-4cc4-beec-fb367325c0f9)>.
- [RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <<https://www.rfc-editor.org/info/rfc2315>>.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <<https://www.rfc-editor.org/info/rfc3579>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, DOI 10.17487/RFC3766, April 2004, <<https://www.rfc-editor.org/info/rfc3766>>.

**[RFC4017]** Stanley, D., Walker, J., and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, DOI 10.17487/RFC4017, March 2005, <<https://www.rfc-editor.org/info/rfc4017>>.

**[RFC4072]** Eronen, P., Ed., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, DOI 10.17487/RFC4072, August 2005, <<https://www.rfc-editor.org/info/rfc4072>>.

**[RFC4086]** Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.

**[RFC4334]** Housley, R. and T. Moore, "Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN)", RFC 4334, DOI 10.17487/RFC4334, February 2006, <<https://www.rfc-editor.org/info/rfc4334>>.

**[RFC4648]** Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

**[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, DOI 10.17487/RFC4851, May 2007, <<https://www.rfc-editor.org/info/rfc4851>>.

**[RFC4945]** Korver, B., "The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX", RFC 4945, DOI 10.17487/RFC4945, August 2007, <<https://www.rfc-editor.org/info/rfc4945>>.

**[RFC4949]** Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

**[RFC4962]** Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, DOI 10.17487/RFC4962, July 2007, <<https://www.rfc-editor.org/info/rfc4962>>.

**[RFC5247]** Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <<https://www.rfc-editor.org/info/rfc5247>>.

**[RFC5272]** Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.

**[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, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.

---

- [RFC5281]** Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, DOI 10.17487/RFC5281, August 2008, <<https://www.rfc-editor.org/info/rfc5281>>.
- [RFC5421]** Cam-Winget, N. and H. Zhou, "Basic Password Exchange within the Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5421, DOI 10.17487/RFC5421, March 2009, <<https://www.rfc-editor.org/info/rfc5421>>.
- [RFC5422]** Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, DOI 10.17487/RFC5422, March 2009, <<https://www.rfc-editor.org/info/rfc5422>>.
- [RFC5652]** Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC5931]** Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, DOI 10.17487/RFC5931, August 2010, <<https://www.rfc-editor.org/info/rfc5931>>.
- [RFC6066]** Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6124]** Sheffer, Y., Zorn, G., Tschofenig, H., and S. Fluhrer, "An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol", RFC 6124, DOI 10.17487/RFC6124, February 2011, <<https://www.rfc-editor.org/info/rfc6124>>.
- [RFC6238]** M'Raihi, D., Machani, S., Pei, M., and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm", RFC 6238, DOI 10.17487/RFC6238, May 2011, <<https://www.rfc-editor.org/info/rfc6238>>.
- [RFC6678]** Hooper, K., Hanna, S., Zhou, H., and J. Salowey, Ed., "Requirements for a Tunnel-Based Extensible Authentication Protocol (EAP) Method", RFC 6678, DOI 10.17487/RFC6678, July 2012, <<https://www.rfc-editor.org/info/rfc6678>>.
- [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, DOI 10.17487/RFC6960, June 2013, <<https://www.rfc-editor.org/info/rfc6960>>.
- [RFC6961]** Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <<https://www.rfc-editor.org/info/rfc6961>>.
- [RFC7029]** Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, DOI 10.17487/RFC7029, October 2013, <<https://www.rfc-editor.org/info/rfc7029>>.

- [RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <<https://www.rfc-editor.org/info/rfc7170>>.
- [RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, DOI 10.17487/RFC7299, July 2014, <<https://www.rfc-editor.org/info/rfc7299>>.
- [RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <<https://www.rfc-editor.org/info/rfc7542>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8146] Harkins, D., "Adding Support for Salted Password Databases to EAP-pwd", RFC 8146, DOI 10.17487/RFC8146, April 2017, <<https://www.rfc-editor.org/info/rfc8146>>.
- [RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <<https://www.rfc-editor.org/info/rfc9325>>.
- [X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", February 2021, <<https://www.itu.int/rec/T-REC-X.690>>.

## 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 [RFC3748], [RFC4017], [RFC5247], and [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: Cipher Suite Negotiation

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

#### **A.4. Requirement 4.2.1.1.2: Tunnel Data Protection Algorithms**

TEAPv1 meets this requirement by mandating cipher suites 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 cipher suites that only include cipher suites that use strong cryptographic algorithms. They do not include cipher suites providing mutually anonymous authentication or static Diffie-Hellman cipher suites as defined in [Section 3.2](#).

#### **A.6. Requirement 4.2.1.2: Tunnel Replay Protection**

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 3.5.1](#) and TLS session resumption using the methods defined in [[RFC9190](#)].

#### **A.10. Requirement 4.2.2: Fragmentation**

TEAPv1 meets this requirement by leveraging fragmentation support provided by TLS as defined in [Section 3.10](#).

#### **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 TEAP 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 3.6.2](#).

## 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](#)]. TLS 1.1 and earlier **MUST NOT** be used with TEAP.
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](#)].
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. The conversation will appear as follows:

```

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

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

EAP-Response/          EAP-Request/
EAP-Type=TEAP, V=1 ->          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.) ...

          <- 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.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) ->

-> 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) ->

-> Intermediate-Result TLV (Failure),
Result TLV (Failure)

Intermediate-Result TLV (Failure),
Result TLV (Failure) ->

TLS channel torn down
(messages sent in cleartext)

-> EAP-Failure

```

### C.3. Full TLS Handshake Using Certificate-Based Cipher Suite

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-Response/           <- EAP-Request/Identity
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
SessionTicket extension)->

// If the server rejects the session resumption,
// 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 coalesced with 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 for TLS 1.2:

```

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

EAP-Response/           <- EAP-Request/
EAP-Type=TEAP, V=1      EAP-Type=TEAP, V=1
(TLS client_key_exchange,
 TLS change_cipher_spec,
 TLS finished) ->  (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/          EAP-Request/
Identity          Identity
EAP-Response/          EAP-Request/
Identity (MyID) ->          EAP-Type=TEAP, V=1
                           (TEAP Start, S bit set, Authority-ID)

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

EAP-Response/          EAP-Request/
EAP-Type=TEAP, V=1 ->          EAP-Type=TEAP, V=1
                           (Fragment 3)

EAP-Response/          EAP-Request/
EAP-Type=TEAP, V=1          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-Response/          EAP-Request/
EAP-Type=TEAP, V=1          EAP-Type=TEAP, V=1
                           (Fragment 2)->
                           (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 coalesced with 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           EAP-Type=TEAP, V=1
(TLS client_hello)->          (TEAP Start, S bit set, Authority-ID)

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

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

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

// First EAP Payload TLV is coalesced with 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] ->          <- EAP-Payload TLV
[EAP-Request/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-Request/Identity])

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

// Next EAP conversation started (with EAP-Request/Identity)
// after successful completion of previous method X. The
// Intermediate-Result and Crypto-Binding TLVs are sent in
// the next packet to minimize round trips.

Intermediate Result TLV (Success),
Crypto-Binding TLV (Response),
EAP-Payload TLV [EAP-Response/Identity (MyID2)] ->

    // Optional additional EAP method Y 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          EAP-Type=TEAP, V=1
(TLS client_hello) ->          (TEAP Start, S bit set, Authority-ID)

EAP-Response/
EAP-Type=TEAP, V=1 ->          <- EAP-Request/
EAP-Type=TEAP, V=1          EAP-Type=TEAP, V=1
(TLS Client Key Exchange    (TLS Server Key Exchange
TLS change_cipher_spec,    TLS Server Hello Done)
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 coalesced with the TLS Finished as
// Application Data and protected by the TLS tunnel.

EAP-Payload TLV/
EAP Identity Response ->          <- EAP Payload TLV, EAP-Request,
                                      (EAP-FAST-MSCHAPV2, Challenge)

EAP Payload TLV, EAP-Response,
(EAP-FAST-MSCHAPV2, Response) ->          <- EAP Payload TLV, EAP-Request,
                                              (EAP-FAST-MSCHAPV2, Success Request)

EAP Payload TLV, EAP-Response,
(EAP-FAST-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 coalesced with 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-Response/           <- EAP-Request/Identity
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. TLS client certificate is also sent.

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

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

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

-> 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) ->

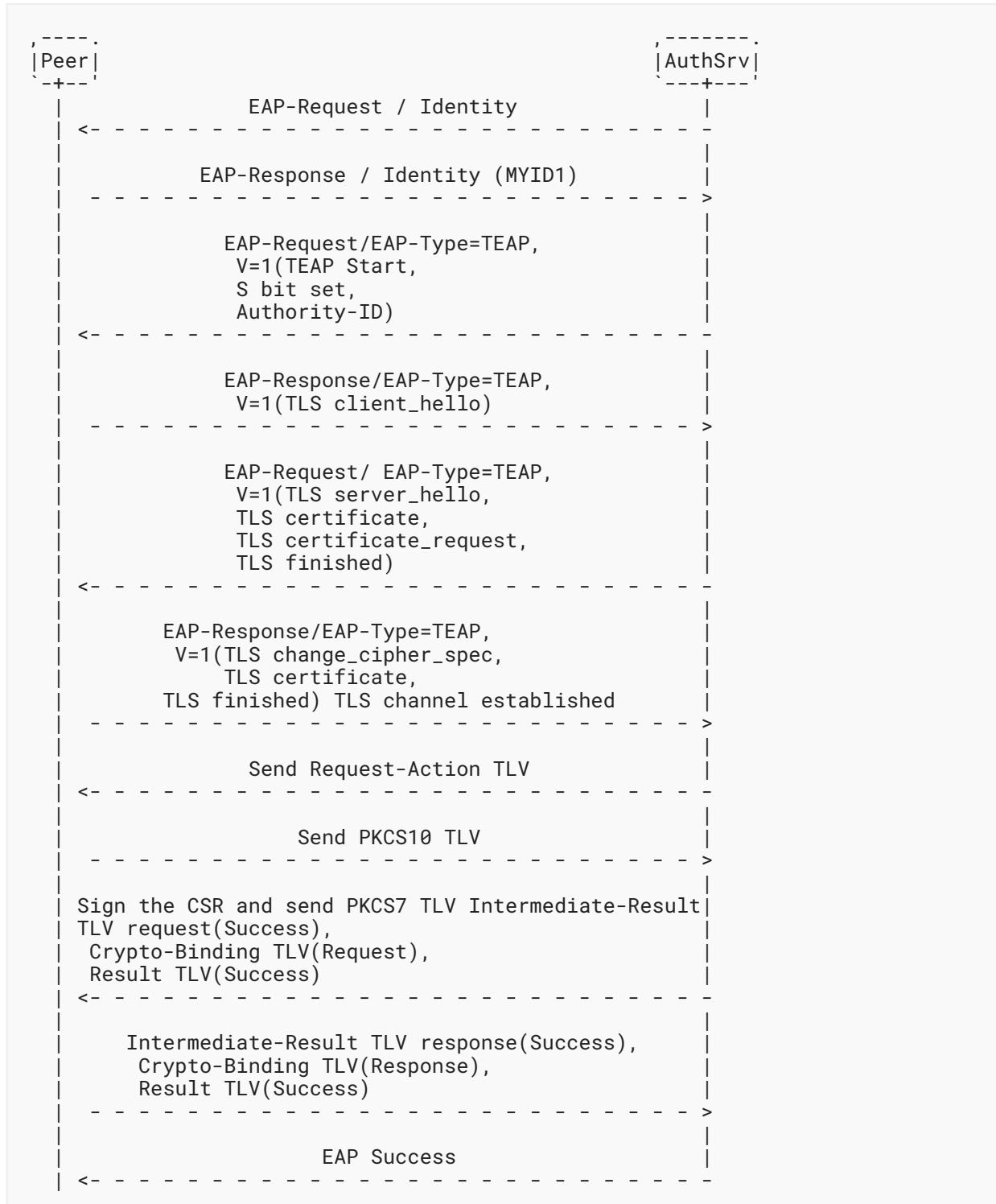
TLS channel torn down
(messages sent in cleartext)

-> EAP-Success

```

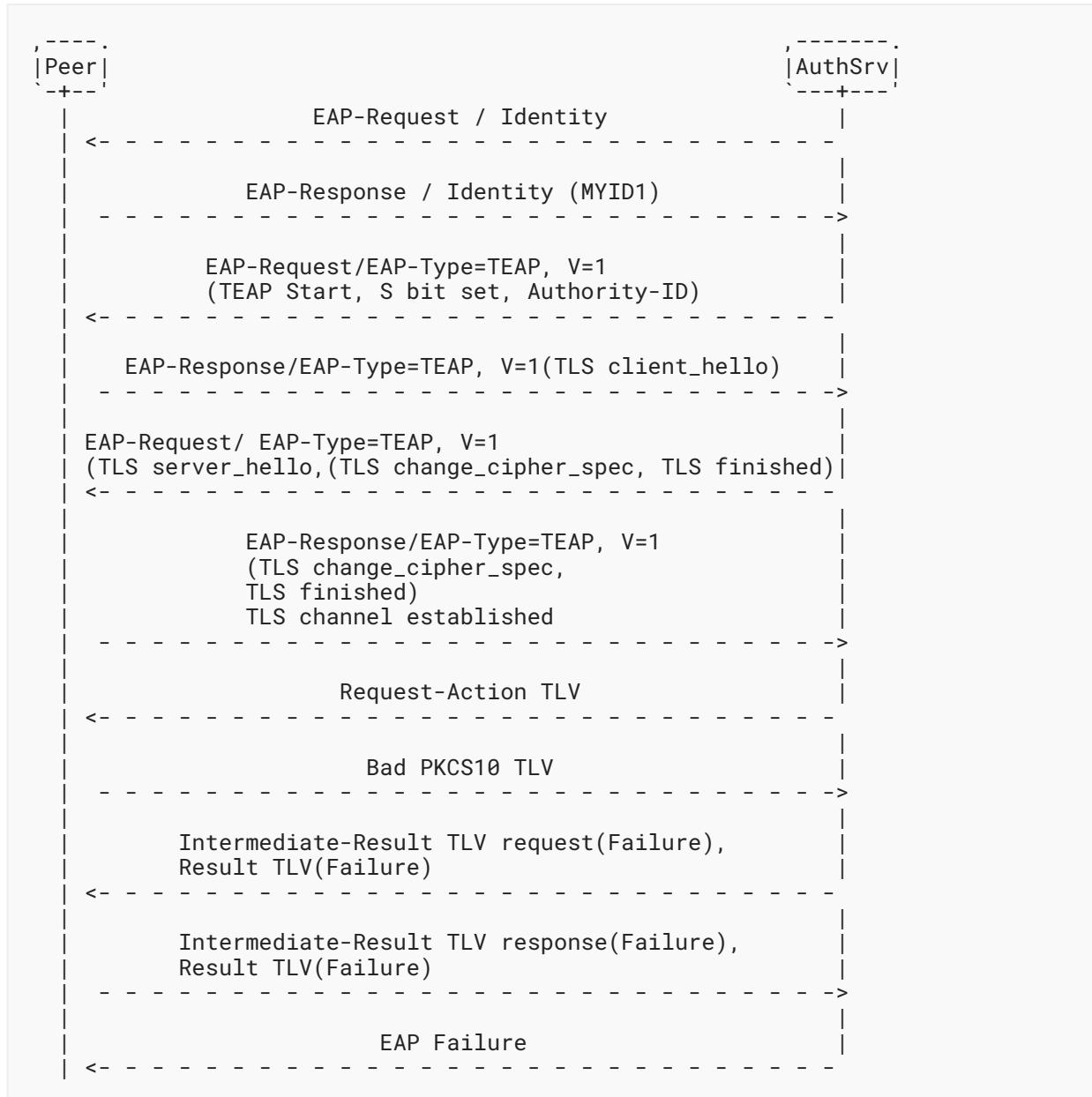
## C.11. PKCS Exchange

The following exchanges show the peer sending a PKCS#10 TLV and server replying with a PKCS7 TLV. The exchange below assumes that the EAP peer is authenticated in Phase 1, either via bidirectional certificate exchange or some other TLS method such as a proof of knowledge (TLS-POK). The conversation will appear as follows:



## C.12. Failure Scenario

The following exchanges show a failure scenario. The conversation will appear as follows:



### C.13. Client Certificate in Phase 1

The following exchanges show a scenario where the client certificate is sent in Phase 1 and no additional authentication or provisioning is performed in Phase 2. The conversation will appear as follows:



## Acknowledgments

Nearly all of the text in this document was taken directly from [RFC7170]. We are grateful to the original authors and reviewers for that document. The acknowledgments given here are only for the changes that resulted in this document.

Alexander Clouter provided substantial and detailed technical feedback on nearly every aspect of the specification. The corrections in this document are based on his work.

We wish to thank the many reviewers and commenters in the EMU WG, including Eliot Lear, Joe Salowey, Heikki Vatianen, Bruno Pereria Vidal, and Michael Richardson. Many corner cases and edge conditions were caught and corrected as a result of their feedback.

Jouni Malinin initially pointed out the issues with [RFC7170]. Those comments resulted in substantial discussion on the EMU WG mailing list, and eventually this document. Jouni also made substantial contributions in analyzing corner cases, which resulted in the text in [Section 6.2.5](#).

## Contributors

**Han Zhou**

**Joseph Salowey**

Email: [joe@salowey.net](mailto:joe@salowey.net)

**Nancy Cam-Winget**

Email: [ncamwing@cisco.com](mailto:ncamwing@cisco.com)

**Steve Hanna**

Email: [steve.hanna@infineon.com](mailto:steve.hanna@infineon.com)

## Author's Address

**Alan DeKok (EDITOR)**

Email: [aland@freeradius.org](mailto:aland@freeradius.org)