[MS-CEAP]:
Compressed Extensible Authentication Protocol (CEAP) Specification

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** Portions of this document were contributed by The Technical Committee (www.thetc.org) and are reproduced with permission. Other portions are covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
06/17/2011	0.1	New	This document was created by The Technical Committee.

Contents

1 Introduction

EAP, specified in [RFC3748], is an **authentication** framework that supports multiple authentication methods.

This document defines the Compressed Extensible Authentication Protocol (CEAP), which specifies extensions to the EAP-TLS Authentication Protocol (EAP-TLS), as specified in [RFC5216], to support the Protected Extensible Authentication Protocol, as specified in [MS-PEAP], in establishing tunneled authentication using TLS and an inner EAP authentication method. It specifies the syntax and semantics of new protocol messages and builds upon and relies heavily upon the EAP-TLS Authentication Protocol [RFC5216].

1.1 Glossary

The following terms are defined in [MS-GLOS]:

access point
authentication
BLOB
certificate
ciphersuite
decryption
EAP identity
EAP method
EAP server
encryption
enforcement client (EC)
Extensible Authentication Protocol (EAP)
fast reconnect
Group Policy
handshake
key derivation
man in the middle (MITM)
network access server (NAS)
network byte order
padding
peer
phase
realm
session
statement of health (SoH)
statement of health response (SoHR)
Transport Layer Security (TLS)
trust root
tunnel

The following terms are specific to this document:

cleartext: In cryptography, **cleartext** is the form of a message (or data) that is transferred or stored without cryptographic protection.

context handle: An opaque handle returned by a TLS implementation to the higher layer (CEAP layer) after a TLS session is established successfully. This is a handle to the TLS session's

security parameter structure ([RFC5246] section A.6) maintained by the TLS layer. As a TLS implementation can handle multiple sessions simultaneously, it relies on the context handle to identify the corresponding session when receiving calls to encrypt and decrypt message functions from the higher layer.

EAP Peer: A network access client requesting access to a network using **EAP** as the **authentication** method.

Network Access Identifier (NAI): The identity included within EAP–Response/Identity (section 5.1 of [RFC3748]). As defined in [RFC4282], this includes an optional username portion as well as a realm portion.

MPPE Keys: Specifies the key material generated by the EAP methods which can be used to perform data encryption between **peer** and **NAS**. There are two types **MPPE Keys** based on the direction of data flow they are used with - MPPE Send Key and MPPE Receive key. Each EAP method has its own mechanism of generating these keys. For example, section 2.3 of [RFC5216] specifies the mechanism to generate the **MPPE Keys** (MS-MPPE-Send-Key and MS-MPPE-Recv-Key) for EAP-TLS authentication protocol.

PEAP Peer: An implementation of a PEAP method on a **EAP peer** that takes care of the PEAP method peer-side requirements.

CEAP Peer: An implementation of a TLS tunneled EAP exchange with an "inner" authentication method performed on an **EAP peer**.

Identity Privacy: A feature on the peer in which the identity sent in the EAP-Response/Identity is different from the identity authenticated by the EAP method. This helps by not sending the actual identity over unprotected network in clear text for privacy reasons (section 7.3 of [RFC3748]).

CEAP Server: An implementation of a TLS tunneled EAP exchange with an "inner" authentication method performed on an **EAP server**.

PEAP Server: An implementation of a PEAP method on a **EAP server** that takes care of the PEAP method server-side requirements.

type-length-value (TLV): Information element encoded within a protocol. Type and length fields are a fixed size (that is, 1 to 4 bytes), and the value field is variable. "Type" indicates what kind of field is encoded; "Length" indicates the size of "Value"; "Value" defines the data portion of the **TLV** element.

virtual private network (VPN): A private communications network existing within a shared or public network platform such as the Internet.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site,

http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[IANA-EAP] Internet Assigned Numbers Authority, "Extensible Authentication Protocol (EAP) Registry", October 2006, http://www.iana.org/assignments/eap-numbers

[IANA-ENT] Internet Assigned Numbers Authority, "Private Enterprise Numbers", January 2007, http://www.iana.org/assignments/enterprise-numbers

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

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-NAPSO] Microsoft Corporation, "Network Access Protection System Overview".

[MS-SOH] Microsoft Corporation, "Statement of Health for Network Access Protection (NAP) Protocol Specification".

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

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

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC2548] Zorn, G., "Microsoft Vendor-Specific RADIUS Attributes", RFC 2548, March 1999, http://www.ietf.org/rfc/rfc2548.txt

[RFC2865] Rigney, C., Willens, S., Rubens, A., and Simpson, W., "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000, http://www.ietf.org/rfc/rfc2865.txt

[RFC3174] Eastlake III, D., and Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001, http://www.ietf.org/rfc/rfc3174.txt

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., et al., "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004, http://www.ietf.org/rfc/rfc3748.txt

[RFC5216] Simon, D., Aboda, B., and Hurst, R., "The EAP-TLS Authentication Protocol", RFC 5216, March 2008, http://www.ietf.org/rfc/rfc5216.txt

[RFC5246] Dierks, T., and Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008, http://www.ietf.org/rfc/rfc5246.txt

[RFC5280] Cooper, D., Santesson, S., Farrell, S., et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008, http://www.ietf.org/rfc/rfc5280.txt

1.2.2 Informative References

[IEEE802.1X] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control", December 2004, http://ieeexplore.ieee.org/iel5/9828/30983/01438730.pdf

[MS-CHAP] Microsoft Corporation, "Extensible Authentication Protocol Method for Microsoft Challenge Handshake Authentication Protocol (CHAP) Specification".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-GPWL] Microsoft Corporation, "Group Policy: Wireless/Wired Protocol Extension".

[RFC1661] Simpson, W., Ed., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994, http://www.ietf.org/rfc/rfc1661.txt

[RFC1750] Eastlake III, D., Crocker, S., and Schiller, J., "Randomness Recommendations for Security", RFC 1750, December 1994, http://www.ietf.org/rfc/rfc1750.txt

[RFC4017] Stanley, D., Walker, J., and Aboba, B., "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, March 2005, http://www.ietf.org/rfc/rfc4017.txt

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005, http://www.ietf.org/rfc/rfc4306.txt

1.3 Overview

The Extensible Authentication Protocol (EAP), specified in [RFC3748], provides support for multiple authentication methods. Transport Layer Security (TLS) provides for mutual authentication, integrity protected ciphersuite negotiation, and key exchange between a peer and a server. The EAP-TLS Authentication Protocol defined in [RFC5216], is specified as an EAP authentication method which provides both certificate-based mutual authentication and key derivation. Although EAP-TLS is well documented and has been analyzed extensively, including cryptanalysis of the protocol which has not yet revealed significant weaknesses in the protocol, EAP-TLS requires organizations to fully deploy PKI before it can be used in a network. Certificate management is a time-consuming and cumbersome administrative task, especially because certificates must be revoked as users lose access to the wireless network.

A final weakness of using EAP-TLS by itself as the EAP authentication method is that, although it protects the user's authentication material, it does not protect the user identity.

This document defines [MS-CEAP], an extension to the EAP-TLS Authentication Protocol which, when used in network authentication scenarios with [MS-PEAP], provides additional protection of the both the user's authentication material and the user identity.

CEAP is initialized and invoked by PEAP, which is acting as the outer EAP authentication method. The CEAP peer establishes a TLS **session** with a CEAP server by using the server's certificate. Then, the client is authenticated using its credential of choice within that TLS session.

The flow of a successful CEAP authentication is as follows:

1. PEAP will invoke CEAP to start a tunneled inner EAP authentication session.
2. The CEAP server and the CEAP peer exchange TLS messages until a TLS session has been established.
3. The CEAP server and the CEAP peer negotiate and agree on an inner authentication method.
4. The CEAP peer and the CEAP server exchange inner method messages until the CEAP peer is successfully authenticated.
5. CEAP notifies PEAP of the result of the authentication.

The security provided by the TLS session established in TLS Tunnel Establishment protects the CEAP peer authentication in phase 2 so that passwords or other dictionary-attackable tokens can be used confidentially.

CEAP is typically deployed in an environment such as the one depicted in the following figure. The EAP peer mutually authenticates with an EAP server using CEAP through a **network access server (NAS)** (that is, a wireless **access point** or VPN gateway). The actual CEAP messages are carried from the EAP peer to the NAS over lower-layer protocols such as the Point-to-Point Protocol (PPP) or [IEEE802.1X], and from the NAS to the EAP server over a lower-layer protocol such as the Remote Authentication Dial-In User Service (RADIUS) [RFC2865].

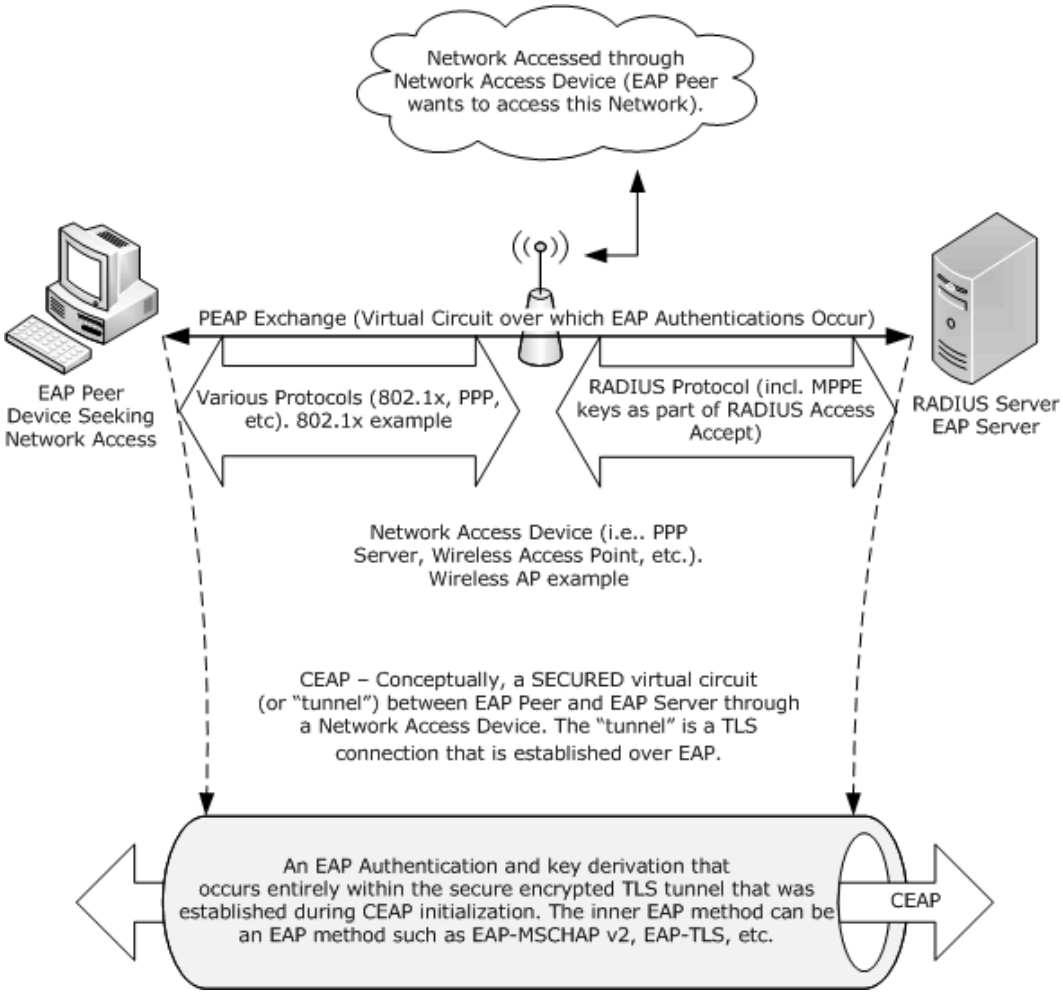


Figure 1: CEAP inside of typical PEAP deployment environment

To understand CEAP, it is necessary to understand both PEAP and EAP-TLS. An overview of PEAP is specified in [MS-PEAP] while an overview of EAP-TLS is specified in [RFC5216] section 2.

1.4 Relationship to Other Protocols

CEAP provides encoding and decoding of compressed EAP packets for successful communication between EAP protocol participants.

The following diagram shows protocol layering involving CEAP:

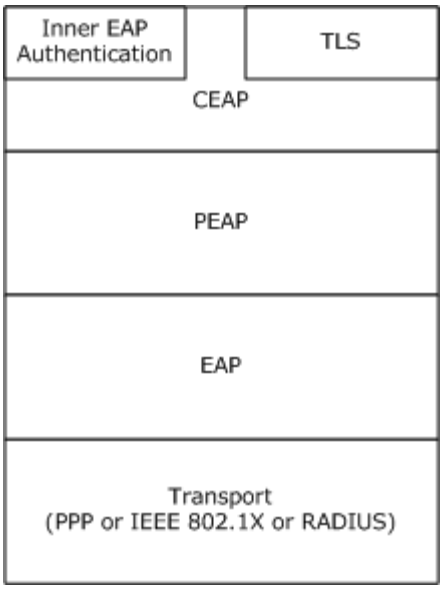


Figure 2: Protocol layering with CEAP

CEAP, like EAP, can run over any PEAP transport that is compliant with [MS-PEAP], such as PPP (for more information, see [RFC1661]).

1.5 Prerequisites/Preconditions

CEAP requires that the EAP authentication method be configured both on the CEAP peer and the server in an implementation-specific manner. EAP and PEAP have their own prerequisites, as specified in [RFC3748] section 3.1 and [MS-PEAP] section 3.1.3 respectively.

EAP requires that both EAP server and peer be configured with the methods which they support, in this case the authentication method provided by PEAP.

1.6 Applicability Statement

CEAP was designed for use in network access authentication.

The use of CEAP is appropriate as the basis for any network authentication scenario.

For more information on CEAP security issues, see section 5.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

CEAP defines Vendor-Specific TLV (section 2.2.3) that can be used by vendors to exchange their own TLVs.

1.9 Standards Assignments

Parameter	Value	Reference
CEAP EAP method type	25	[IANA-EAP]
EAP **Type-Length-Value (TLV)** extensions method type (also known as MS-Authentication-TLV)	33	[IANA-EAP]
Vendor-ID for Microsoft (SMI Private Enterprise Code)	311	[IANA-ENT]

2 Messages

The following sections specify how CEAP messages are encapsulated on the wire and EAP extensions methods.

2.1 Transport

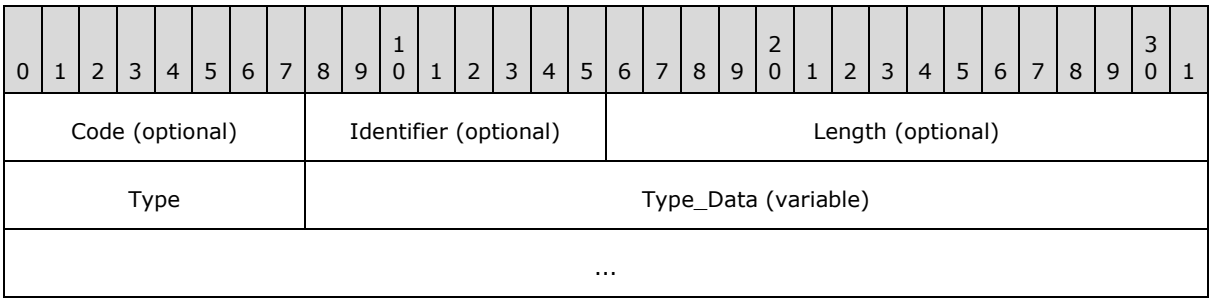
As an authentication method, CEAP MUST be transported by EAP. As a result, protocols that carry EAP (for example, PPP [RFC1661], IEEE802.1x [IEEE802.1X], and RADIUS [RFC2865]) ultimately provide the transport of the associated messages, as specified in [RFC3748] section 3.

As an EAP method, CEAP relies entirely on EAP for the reliable delivery of its messages.

2.2 Message Syntax

2.2.1 CEAP Packet

The following shows a CEAP packet. This is a modified version of an EAP Packet as specified in [RFC3748] section 4.



Code (1 byte): Indicates whether this packet is a Request or a Response, as specified in [RFC3748] section 4.1. If **Type** is set to 33 and the **Type_Data** field contains one of the EAP Expanded Type 2 TLVs (section 2.2.4.2), or **Type** is set to 254 and the **Type_Data** field contains one of the EAP Expanded Type 1 TLVs (section 2.2.4.1), this field MUST be present. Otherwise, this field MUST NOT be present.

Identifier (1 byte): Identifier for this packet, as specified in [RFC3748] section 4.1. If **Type** is set to 33 and the **Type_Data** field contains one of the EAP Expanded Type 2 TLVs (section 2.2.4.2), or **Type** is set to 254 and the **Type_Data** field contains one of the EAP Expanded Type 1 TLVs (section 2.2.4.1), this field MUST be present. Otherwise, this field MUST NOT be present.

Length (2 bytes): If **Type** is set to 33 and the **Type_Data** field contains one of the EAP Expanded Type 2 TLVs (section 2.2.4.2), or **Type** is set to 254 and the **Type_Data** field contains one of the EAP Expanded Type 1 TLVs (section 2.2.4.1), this field MUST be present. Otherwise, this field MUST NOT be present.

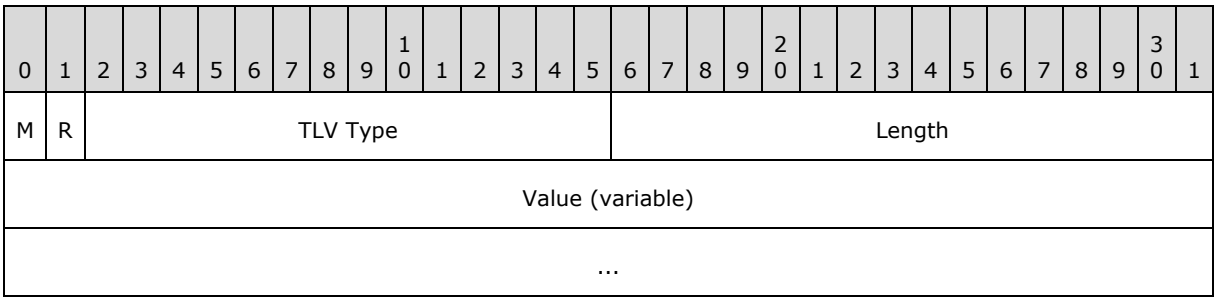
Type (1 byte): The Type of Request or Response, as specified in [RFC3748] section 5.

Type_Data (variable): A field that varies with the Type of Request and the associated Response, as specified in [RFC3748] section 5.

2.2.2 TLV

The following diagram specifies the standard TLV structure that is used by the EAP Expanded Type 1 (section 2.2.4.1) and Type 2 packet types (section 2.2.4.2).

The fields of the structure MUST be transmitted in network byte order from left to right.



M (1 bit): The M bit has the following possible values and MUST be set:

Value	Meaning
0	This TLV MAY be supported (is nonmandatory) by the recipient. If the TLV is not supported, the recipient MUST ignore the TLV.
1	This TLV MUST be supported (is mandatory) by the recipient. If the TLV is not supported, the recipient MUST discard the CEAP packet that contains the TLV.

R (1 bit): The R bit is reserved and MUST be set to zero when sent and MUST be ignored on receipt.

TLV Type (14 bits): A 14-bit unsigned integer in network byte order that indicates the type of data in the **Value** field. Allocated types include the following:

Value	Description
1	SoH TLV
2	SoH Request TLV
3	Result TLV or SoH Response TLV (when transmitted in a Vendor-Specific TLV)
7	Vendor-Specific TLV
12	Cryptobinding TLV

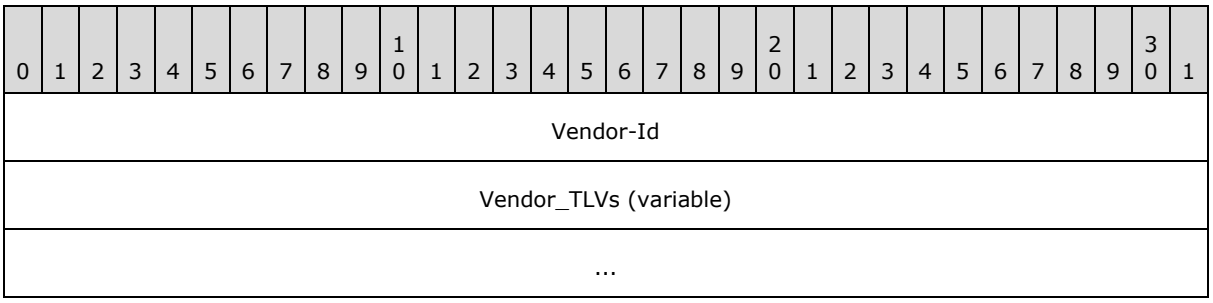
Length (2 bytes): A 16-bit unsigned integer in network-byte order that indicates the length, in bytes, of the **Value** field.

Value (variable): The value MUST be formatted in accordance with the type specified in the **TLV Type** field.

2.2.3 Vendor-Specific TLV

A vendor-specific TLV is used to carry a set of TLVs specific to a vendor (indicated by the **Vendor-Id** field). The TLV **Type** field MUST be set to 7 (see section 2.2.2).

The following diagram shows the format of the vendor-specific TLV, which is placed in the **Value** field of the TLV. The fields of the header are transmitted as bytes from left to right.



Vendor-Id (4 bytes): A 32-bit unsigned integer in network byte order, with the most significant 8 bits set to 0 and the remaining 24 bits set to the Structure and Identification of Management Information (SMI) code of the vendor, taken from [IANA-ENT]. Microsoft vendor-specific TLVs MUST have the **Vendor-Id** field set to 311 (0x00000137).

Vendor_TLVs (variable): One or more TLVs defined by the vendor, as indicated by the preceding **Vendor-Id** field.

2.2.4 EAP Expanded Types

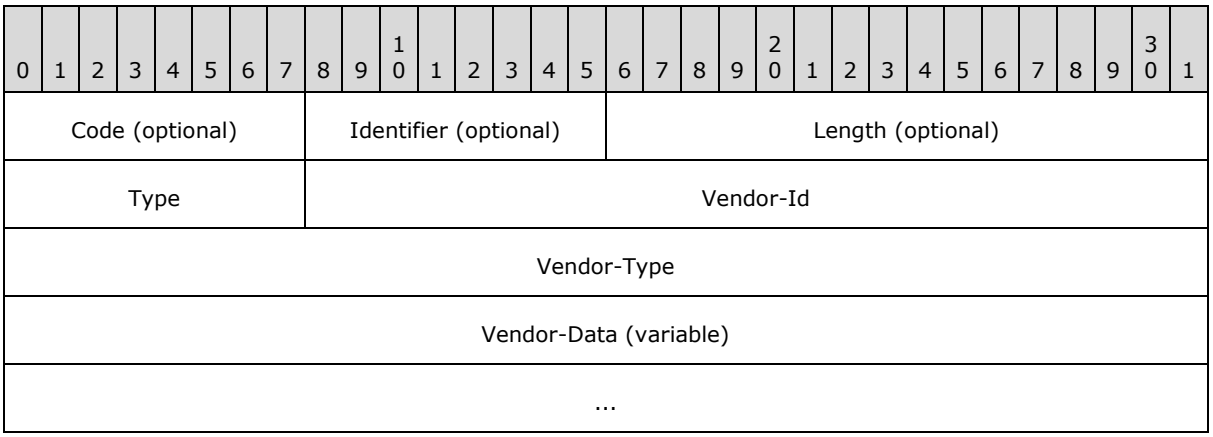
CEAP introduces several expanded types of an EAP packet. These types are categorized as follows:

- EAP Expanded Type1
- EAP Expanded Type2

The structures of the two types and the specific packets that belong to these two types are shown in the following sections.

2.2.4.1 EAP Expanded Type 1

The following diagram shows a CEAP packet carrying an EAP Expanded Type1 packet, as specified in [RFC3748] section 5.7. The Type is 254 and the **Vendor-Id**, **Vendor-Type**, and **Vendor-Data** are part of the **Type_Data** field of an EAP packet (section 2.2.1).



Type (1 byte): MUST be set to 254, as specified in [RFC3748] section 5.7.

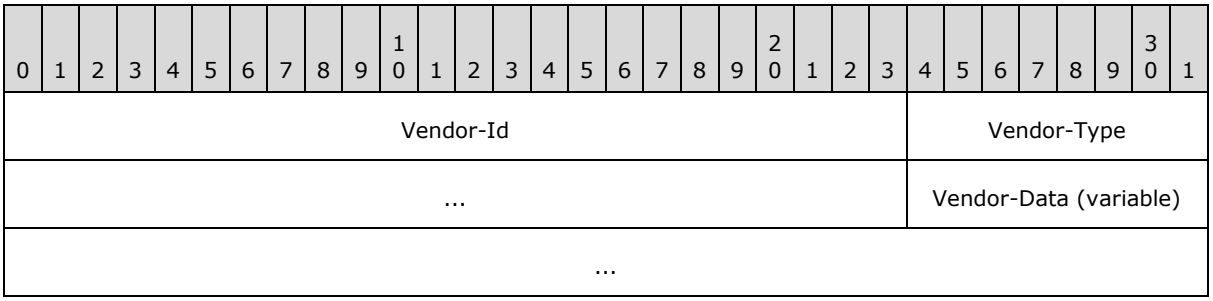
Vendor-Id (3 bytes): The SMI Network Management Private Enterprise Code of the vendor, as specified in [RFC3748] section 5.7.

Vendor-Type (4 bytes): The vendor-specific Type, as specified in [RFC3748] section 5.7.

Vendor-Data (variable): This field is defined by the vendor, as specified in [RFC3748] section 5.7.

2.2.4.1.1 SoH EAP Expanded Type

This structure is an Expanded EAP Type1 with the following values for the fields.



Vendor-Id (3 bytes): A 3-byte unsigned integer that MUST be set to 311.

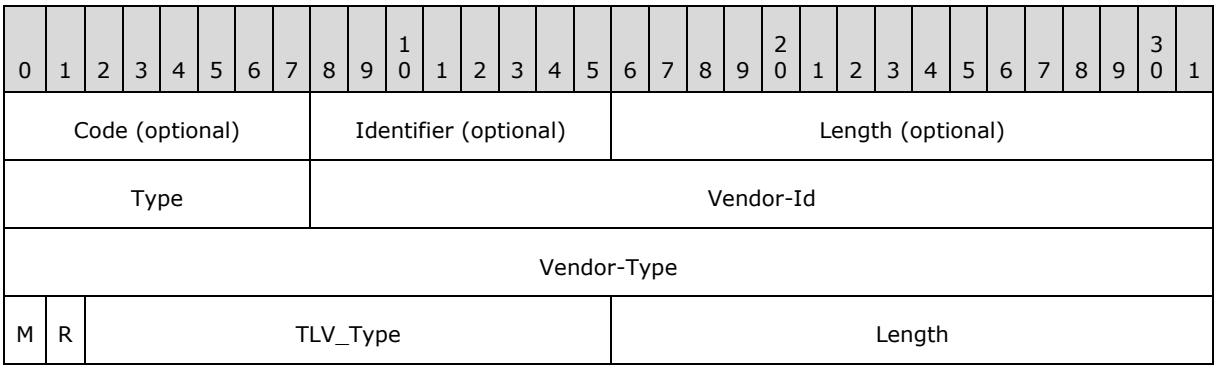
Vendor-Type (4 bytes): A 4-byte unsigned integer that MUST be set to 33.

Vendor-Data (variable): This contains one of the following structures:

- SoH Request TLV,
- SoH TLV,
- Capabilities Negotiation Request,
- Capabilities Negotiation Response.

2.2.4.1.1.1 SoH Request TLV

The SoH Request TLV is a vendor TLV sent within a Microsoft vendor-specific TLV ([MS-PEAP] section 2.2.6) to the CEAP peer by the CEAP server. Its purpose is to trigger transmission of an SoH message by the peer's Statement of Health for Network Access Protection Protocol [MS-SOH] entity. The SoH Request TLV MUST be present only in an EAP request packet.



M (1 bit): The M bit MUST be set to 0.

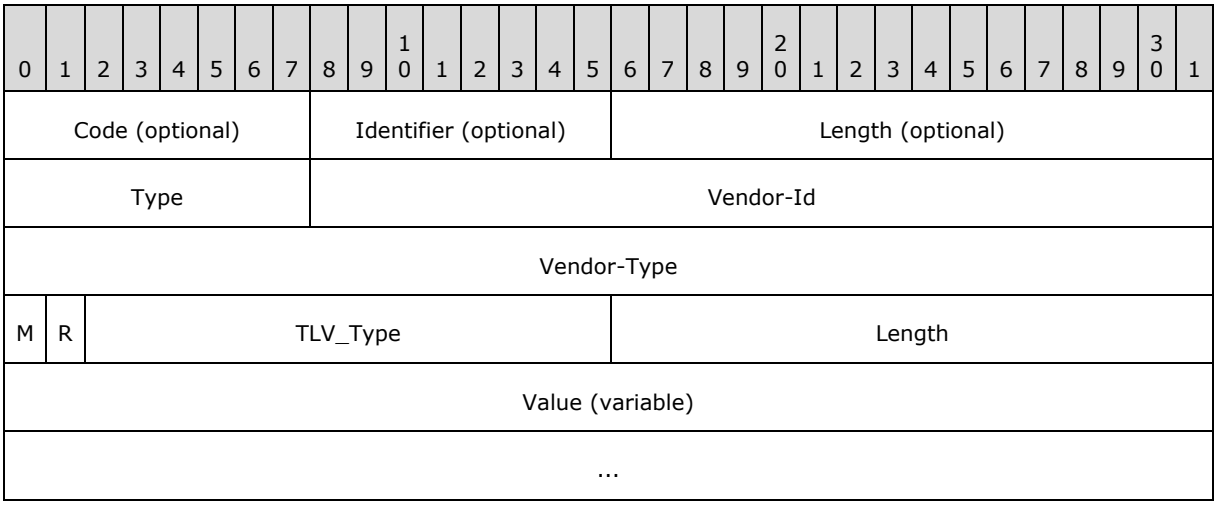
R (1 bit): The R bit is reserved. It MUST be set to zero when sent and MUST be ignored on receipt.

TLV_Type (14 bits): A 14-bit unsigned integer that MUST be set to 2.

Length (2 bytes): A 16-bit unsigned integer in network byte order that indicates the length, in bytes, of the **Value** field. This MUST be set to 0.

2.2.4.1.1.2 SoH TLV

The SoH TLV is a vendor TLV sent within a Microsoft vendor-specific TLV to the CEAP server by the CEAP peer. Its ultimate recipient is the SoH entity [MS-SOH] at the CEAP server. The SoH TLV MUST be present only in an EAP response packet.



M (1 bit): The M bit MUST be set to 0.

R (1 bit): The R bit is reserved. It MUST be set to zero when sent and MUST be ignored on receipt.

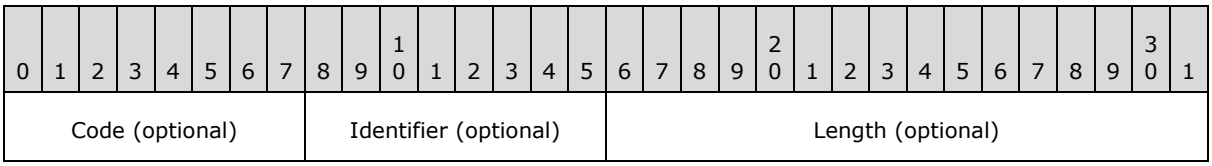
TLV_Type (14 bits): A 14-bit unsigned integer that MUST be set to 1.

Length (2 bytes): A 16-bit unsigned integer in network byte order that indicates the length, in bytes, of the **Value** field.

Value (variable): This MUST contain an SoH message, as defined in [MS-SOH] section 2.2.5.

2.2.4.1.1.3 Capabilities Negotiation Request

The Capabilities Negotiation Request packet is sent by the CEAP server after receiving the identity response and before SOH/PEAP negotiation. The structure is show below.



Type	Vendor-Id
Vendor-Type	
Vendor-Data	

Type (1 byte): MUST be set to 254, as specified in [RFC3748] section 5.7.

Vendor-Id (3 bytes): A 3-byte unsigned integer that MUST be set to 311.

Vendor-Type (4 bytes): A 4-byte unsigned integer that MUST be set to 34.

Vendor-Data (4 bytes): This contains 32 bits, used to denote various capabilities of the CEAP server. Bits 0-30 are reserved for future use, and MUST be set to zero when sent and MUST be ignored on receipt. The bit field 'F' shown above denotes the PEAP Phase2 Fragmentation Capability. This flag is set to 1 if the CEAP server is PEAP Phase2 Fragmentation Capable, and set to 0 otherwise.

2.2.4.1.1.4 Capabilities Negotiation Response

The Capabilities Negotiation Response packet is sent by the CEAP peer after receiving the capabilities negotiation request packet. The structure is shown below:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Code (optional)								Identifier (optional)								Length (optional)																	
Type								Vendor-Id																							
Vendor-Type																															
Vendor-data																															

Vendor-data (4 bytes): This contains 32 bits, and is used to denote various capabilities of the CEAP peer. Bits 0-30 are reserved for future use. The bit field 'F' shown above denotes the PEAP Phase2 Fragmentation Capability. This flag is set to 1 if the CEAP peer is PEAP Phase2 Fragmentation Capable, and set to 0 otherwise.

2.2.4.2 EAP Expanded Type 2

The following diagram shows a CEAP packet carrying an EAP Expanded Type 2 packet. These packets are used to facilitate the exchange of TLVs between a CEAP peer and a CEAP server.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Code (optional)								Identifier (optional)								Length (optional)																	
Type								Type_Data (variable)																							

...

Type (1 byte): MUST be set to 33.

Type_Data (variable): Specific EAP TLV types as defined in the following sections. See TLV (section 2.2.2) for the structure of the TLVs. CEAP implementations MUST transmit only the following TLVs: Cryptobinding TLV (section 2.2.4.2.1), Result TLV (section 2.2.4.2.2), and SoH Response TLV (section 2.2.4.2.3).

Within an EAP TLV type, the Result TLV, Cryptobinding TLV, and SoH Response TLV can be sent in any order. The receiver MUST NOT assume any order of the TLVs.

2.2.4.2.1 Cryptobinding TLV

The cryptobinding TLV is a TLV used to ensure that the EAP peer and the EAP server participated in both the inner and the outer EAP authentications of a PEAP authentication. Its structure is shown below:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Code (optional)								Identifier (optional)								Length (optional)																	
Type								M	R	TLV_Type																		Length			
...								Value																							
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(Value cont'd for 6 rows)																															
...																															

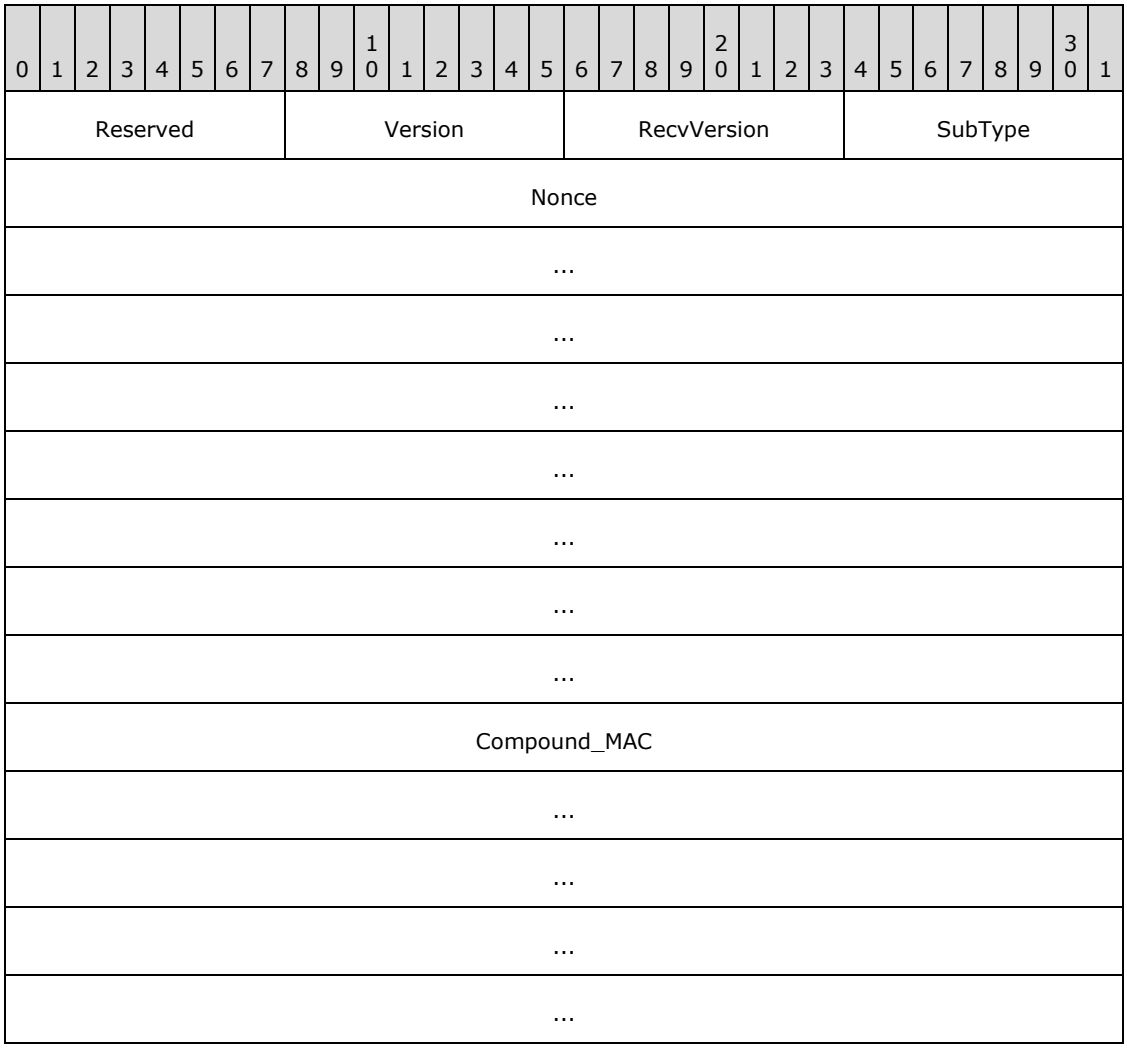
M (1 bit): The M bit MUST be set to 0.

R (1 bit): The R bit is reserved and MUST be set to zero when sent and MUST be ignored on receipt.

TLV_Type (14 bits): A 14-bit unsigned integer in network byte order that indicates the type of data in the **Value** field. The **TLV_Type** MUST be set to 12 (0x0C) for the cryptobinding TLV.

Length (2 bytes): A 16-bit unsigned integer in network byte order that indicates the length, in bytes, of the **Value** field. The value of this field MUST be 56 (0x38).

Value (56 bytes): The **Value** field of the cryptobinding TLV MUST be formatted as follows.



Reserved (1 byte): An 8-bit unsigned integer that is reserved and MUST be set to zero when sent and MUST be ignored on receipt.

Version (1 byte): An 8-bit unsigned integer that indicates the version of the cryptobinding TLV and MUST be set to 0.

RecvVersion (1 byte): An 8-bit unsigned integer field that MUST be set to 0.

SubType (1 byte): An 8-bit unsigned integer that indicates whether the cryptobinding TLV is a request or a response. Its value MUST be one of the following.

Value	Meaning
0	This cryptobinding TLV represents a request.
1	This cryptobinding TLV represents a response.

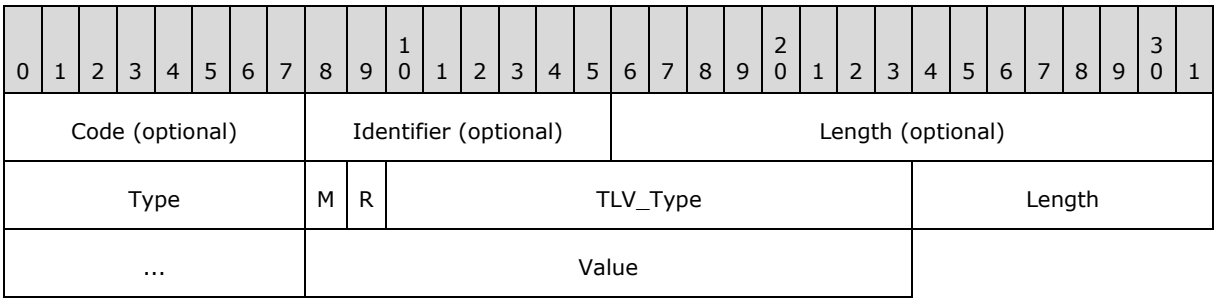
Nonce (32 bytes): A 256-bit unsigned integer containing a temporally unique (random) value. For more information, see [RFC1750].

Compound_MAC (20 bytes): A 160-bit unsigned integer containing the value used to cryptographically associate the TLS Tunnel Establishment and phase 2 authentications of PEAP. For more information, see [MS-PEAP] section 3.1.5.4.

2.2.4.2.2 Result TLV

The Result TLV is a TLV, as specified in section 2.2.2. It is used to represent the status (success or failure) of the CEAP method negotiation or to indicate the sender's consent (ability or inability) to participate in a fast-reconnect.

The Result TLV structure is shown below.



M (1 bit): The M bit MUST be set to 1, indicating that the recipient MUST support the result TLV.

R (1 bit): The R bit is reserved and MUST be set to zero when sent and MUST be ignored on receipt.

TLV_Type (14 bits): A 14-bit unsigned integer that MUST be set to 0x03.

Length (2 bytes): A 16-bit unsigned integer in network byte order that indicates the length, in bytes, of the **Value** field. This MUST be set to 2.

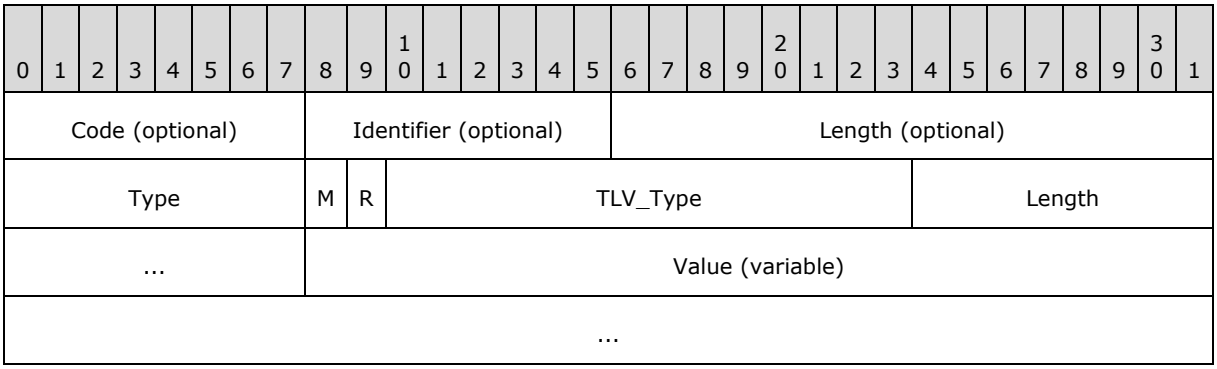
Value (2 bytes): A 16-bit unsigned integer in network byte order. The value indicates the authentication result. Possible values are listed in the table below.

Value	Meaning
0	Reserved and MUST NOT be sent
1	Success
2	Failure
3 — 65535	Reserved and MUST NOT be sent

2.2.4.2.3 SoH Response TLV

The SoH Response TLV is a vendor TLV sent within a Microsoft vendor-specific TLV. Sent to the CEAP peer by the CEAP server, its ultimate recipient is the Statement of Health (SoH) entity, as specified in [MS-SOH], at the peer.

The SoH Response TLV is carried in the **Type-data** field of the EAP Expanded Type 1 structure (section 2.2.4.1).



M (1 bit): The M bit MUST be set to 0.

R (1 bit): The R bit is reserved and MUST be set to zero when sent and MUST be ignored on receipt.

TLV_Type (14 bits): A 14-bit unsigned integer that MUST be set to 3.

Length (2 bytes): A 16-bit unsigned integer in network byte order that indicates the length, in bytes, of the **Value** field.

Value (variable): This MUST contain a Statement of Health Response (SoHR) message, as defined in [MS-SOH] section 2.2.6.

3 Protocol Details

The following sections specify details of CEAP, including abstract data models and message processing rules.

3.1 Common Details

This section defines common details CEAP for all participants of the Protected Extensible Authentication Protocol, as specified in sections 3.2 and 3.3.

3.1.1 Abstract Data Model

This section describes a conceptual model that an implementation can maintain to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

A CEAP protocol participant MUST maintain the following data elements:

isFastReconnectAllowed: A Boolean flag indicating whether **fast reconnect** is allowed (TRUE) for the session or not (FALSE).

isSoHEnabled: A Boolean flag indicating whether SoH is enabled (TRUE) or not (FALSE). This is a configurable field on both peer and server.

isCryptoSupported: A Boolean flag indicating whether the implementation supports Cryptobinding TLVs (section 2.2.4.2.1) (TRUE) or not (FALSE). If the implementation does not support Cryptobinding TLV, then it neither validates (if any are received) nor sends a Cryptobinding TLV.<1>

isCryptoRequired: A Boolean flag indicating whether the implementation requires Cryptobinding TLVs to be exchanged for the final authentication to be successful (TRUE) or not (FALSE). This is a configurable field on both peer and server.

BypassCapNegotiation: A Boolean flag indicating whether the machine is configured to exchange Capabilities Negotiation Method (section 2.2.9.3) packets (TRUE) or not (FALSE).<2>

AssumePhase2Frag: A Boolean flag which indicates whether the counterpart (at the remote end) supports fragmentation and reassembly of the CEAP inner method packets (TRUE) or not (FALSE). This value is meaningful only when **BypassCapNegotiation** is set to TRUE.<3>

isCapabilitiesSupported: A Boolean flag indicating whether the implementation supports Capabilities Negotiation Method (section 2.2.8.3) packets for the session (TRUE) or not (FALSE).<4>

isFragmentationCapabilitySupported: A Boolean flag indicating whether fragmentation and reassembly of the CEAP inner method packets is supported for the session by both peer and server (TRUE) or not (FALSE).<5>

MaxSendPacketSize: An integer indicating the maximum EAP packet size. Values are obtained for the CEAP peer and CEAP server as specified in sections 3.2.3 and 3.3.3.

TunnelKey: The CEAP Tunnel Key (TK) is a 64-octet key generated as specified in section 3.1.5.5.2.1. This variable is used while generating Cryptobinding TLVs (section 3.1.5.5) and the final **MPPE keys** (section 3.1.5.7).

InnerMPPESendKey: A variable-length string returned by the CEAP method when the CEAP authentication is successful. This variable is used while generating **InnerSessionKey (ISK)** as specified in section 3.1.5.5.2.2.

InnerMPPESendKeyLength: Specifies the length of **InnerMPPESendKey** in octets.

InnerMPPERecvKey: A variable-length string returned by inner method when the CEAP authentication is successful. This variable is used while generating **ISK** as specified in section 3.1.5.5.2.2.

InnerMPPERecvKeyLength: Specifies the length of **InnerMPPERecvKey** in octets.

InnerSessionKey (ISK): ISK is a 32-octet string generated from keys provided by the inner method. This variable is used while generating Cryptobinding TLVs, as specified in section 3.1.5.5.

CtxtHandle: A 128-bit **context handle** obtained, as specified in sections 3.2.7.1 and 3.3.7.1, when the TLS Tunnel Establishment tunnel is established. This handle is used in **encryption** and **decryption** of messages during phase 2 of CEAP.

InnerIdentity: An LPWSTR string (as specified in [MS-DTYP] section 2.2.35) for storing the identity exchanged as part of CEAP method authentication.

3.1.2 Timers

None.

3.1.3 Initialization

Processing specific to the CEAP peer and the CEAP server initialization is specified in sections 3.2.3 and 3.3.3, respectively.

3.1.4 Higher-Layer Triggered Events

Higher-layer triggered events are specified in sections 3.2.4 and 3.3.4.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Status and Error Handling

If a CEAP implementation receives a packet that does not satisfy the MUST clauses of this specification, the packet MUST be silently discarded.

CEAP supports TLS alert messages (as specified in [RFC2246] section 7.2) when establishing the TLS session. CEAP does not have its own error messaging capabilities.

CEAP implementations MUST support the EAP Expanded Type 2 as specified in section 2.2.4.2 for the communication of authentication status between the CEAP peer and the CEAP server.

In EAP, success or failure packets are sent as the last packet in a conversation. However, these packets are not protected, and they can be forged by an attacker. Also, success and failure packets are not retransmitted and, therefore, may be lost. As a result, CEAP provides its own protected and reliable success/failure indications via the Result TLV (section 2.2.4.2.2) of the EAP Expanded Type 2 (section 2.2.4.2). A CEAP peer implementation MUST consider authentication successful only if it receives both an EAP success packet and an EAP Expanded Type 2 packet containing a Result TLV with the **Value** field set to 1 (which indicates success, as specified in section 2.2.4.2.2). This

behavior is also evident in the processing rules specified in sections 3.2.5.4.7, 3.2.5.4.8, and 3.2.5.4.9.

3.1.5.2 CEAP Packet Processing

This section describes the CEAP packet processing common to peer and server. In contrast, CEAP packet processing specific to peer and server is described in sections 3.2.5.4 and 3.3.5.4 respectively.

3.1.5.2.1 Received CEAP Packet with L and M Bit Set

If **isFragmentationCapabilitySupported** is TRUE and the CEAP phase 2 is in progress, then store the first fragment and send a CEAP Fragment Acknowledgement packet (section 2.2.3) request (server) or response (peer). For all the next fragments (M bit set and L bit not set), store the fragments and send a CEAP Fragment Acknowledgement packet request (server) or response (peer). After receiving the last fragment (L and M bits not set), reassemble all the fragments and do the packet processing as specified in sections 3.2.5.4 and 3.3.5.4.

If **isFragmentationCapabilitySupported** is FALSE and the CEAP phase 2 is in progress, then the packet is ignored.

3.1.5.2.2 Sending CEAP Packet with packet size more than MaxSendPacketSize

If **isFragmentationCapabilitySupported** is TRUE and the CEAP phase 2 is in progress, then fragment the packet and send the first fragment (L and M bit set). After receiving a CEAP Fragment Acknowledgement packet (section 2.2.3) response (server) or request (peer), send the next fragment (M bit set and L bit not set). Continue sending the fragments until the last fragment (L and M bits not set).

If **isFragmentationCapabilitySupported** is FALSE and the CEAP phase 2 is in progress, then the packet is ignored.

3.1.5.3 Version Negotiation

CEAP relies on PEAP Version Negotiation as specified in [MS-PEAP] section 3.1.5.3.

3.1.5.4 TLS Tunnel Establishment (TLS Tunnel Establishment)

TLS Tunnel Establishment of CEAP is a slightly modified implementation of EAP-TLS, as specified in [RFC5216], the only differences being:

1. A CEAP peer MAY send a certificate when requested by a CEAP server.
2. Implementations MUST set the **Type** field of the CEAP packets to 25 (CEAP).
3. The TLS version supported MUST correspond to TLS v1.0.
4. To ensure interoperability, CEAP peers and CEAP servers MUST be able to negotiate the following TLS **cipher-suites** (as specified in [RFC2246] section A.5):
   - TLS_RSA_WITH_RC4_128_MD5
   - TLS_RSA_WITH_RC4_128_SHA

For more information on the processing associated with TLS Tunnel Establishment during CEAP, see sections 3.2.5.2 and 3.3.5.2.

3.1.5.5 Cryptobinding

By deriving and exchanging values from the CEAP TLS Tunnel Establishment key material (**Tunnel Key**) and from the CEAP phase 2 CEAP method key material (**Inner Session Key**), it is possible to prove that the two authentications terminate at the same two entities (CEAP peer and CEAP server). This process, termed "cryptobinding", is used to protect the CEAP negotiation against "**Man in the Middle**" attacks.

To facilitate this, a two-way handshake between the CEAP peer and the CEAP server is initiated with two messages: the cryptobinding request (sent from the CEAP server to CEAP peer) and the cryptobinding response (sent from the CEAP peer to CEAP server); both messages use the same format (see Cryptobinding TLV (section 2.2.4.2.1)).

Implementations SHOULD<6> choose to support the cryptobinding feature of CEAP.

The **Compound_MAC** field in the cryptobinding packet MUST be the output of an HMAC-SHA1-160 operation, as specified in [RFC2104] and [RFC3174]. The HMAC-SHA1-160 operation requires the data and the key as inputs, both of which are derived from the CEAP TLS Tunnel Establishment and the inner method. For more details on how an implementation generates the data used in the HMAC-SHA1-160 operation for the cryptobinding packet, see section 3.1.5.5.1. For more details on how an implementation generates the key used in the HMAC-SHA1-160 operation for the cryptobinding packet, see section 3.1.5.5.2.

3.1.5.5.1 Input Data Used in the Cryptobinding HMAC-SHA1-160 Operation

The data used as the input to the HMAC-SHA1-160 operation used in the creation of the Compound MAC MUST be constructed, through concatenation, as follows:

1. 60 bytes containing the cryptobinding TLV with the **Compound_MAC** field zeroed out.
2. 1 byte containing the EAP type sent by the peer in the first processed CEAP message. For CEAP, the value MUST be the IANA-assigned EAP type code (25) for CEAP (see [IANA-EAP]).
3. The **Outer_TLV_Data** field of a CEAP start packet (as specified in section 2.2.6.2 when the HMAC-SHA1-160 operation is performed on a Peer, or the **Outer_TLV_Data** field of a Client Hello Packet (as specified in section 2.2.6.1) when the HMAC-SHA1-160 operation is performed on a Server.

3.1.5.5.2 Key Used in the Cryptobinding HMAC-SHA1-160 Operation

The key used by the HMAC-SHA1-160 operation to create the Compound MAC field is called the Compound MAC Key (CMK). The CMK MUST be constructed by following the steps specified later in this section. These steps produce the following intermediate values:

- Tunnel key (TK): A 64-octet key generated by TLS Tunnel Establishment of CEAP. For details, see section 3.1.5.5.2.1. The generated Tunnel Key is stored in the variable **TunnelKey**.
- Inner Session Key (ISK): A 32-octet string generated from keys provided by the inner method (or 32 zero octets if the inner method does not provide keys), if CEAP did not resume an authentication using fast-reconnect (as specified in 3.1.5.5.2.2). An ISK is not generated in the case of fast-reconnect, because the Intermediate CEAP MAC Key (IPMK) is generated from TK (as specified in 3.1.5.5.2.2). The generated Inner Session Key is stored in the variable **InnerSessionKey**.
- Intermediate CEAP MAC Key (IPMK): The intermediate combined key used to derive the Compound MAC (as specified in section 3.1.5.5.2.2).

- IPMK Seed: The seed value used in the call to the PRF+ operation (for more information, see [RFC4306] section 2.13). For details, see section 3.1.5.5.2.2.

3.1.5.5.2.1 CEAP Tunnel Key (TK)

The CEAP Tunnel Key (TK) is calculated using the first 60 octets of the (secret) key material generated, as described in the EAP-TLS algorithm (see [RFC5216] section 2.3). More explicitly, the TK is the first 60 octets of the Key_Material, as specified in [RFC5216]: TLS-PRF-128 (master secret, "client EAP encryption", client.random || server.random).

3.1.5.5.2.2 Intermediate CEAP MAC Key (IPMK) and Compound MAC Key (CMK)

The Intermediate CEAP MAC key (IPMK) and Compound MAC Key (CMK) are constructed using the following steps:

- If the CEAP peer and the CEAP server resumed an authentication using fast reconnect, then IPMK and CMK are obtained from TK as shown in the following steps.
- If the CEAP peer and the CEAP server did not resume an authentication using fast reconnect, and an inner method was used for authenticating the CEAP peer, then the IPMK is generated using the following steps:

  1. Generate an ISK:

     - If the CEAP method generates keys, then an implementation MUST obtain the **InnerMPPESendKey**, **InnerMPPERecvKey** and their lengths from the inner method as specified in sections 3.2.5.4.7 and 3.3.7.3. The **InnerMPPESendKey** and **InnerMPPERecvKey** are the same as MS-MPPE-Send-Key and MS-MPPE-Recv-Key respectively as specified in [RFC2548], sections 2.4.2 and 2.4.3.

       Each inner method decides how to generate these keys. The Protected Extensible Authentication Protocol uses the keys returned by the inner method and calculates ISK as follows: (The following Microsoft Point-to-Point Encryption (MPPE) keys are not encrypted by RADIUS shared secret, and contain only the key itself and no length, salt, or type field.)

       ```
       Peer ISK = InnerMPPESendKey | InnerMPPERecvKey
       Server ISK = InnerMPPERecvKey | InnerMPPESendKey
       ```

       If the concatenated string length (obtained from **InnerMPPESendKeyLength** and **InnerMPPERecvKeyLength**) is more than 32 octets, then the first 32 octets form the **ISK**. If the concatenated string length is less than 32 octets, then the string is appended with 0x00 at the end as **padding** to obtain a total length of 32 octets.

     - If the CEAP method did not generate any keys, then the ISK MUST be 32 octets of 0x00.

  2. Generate the IPMK Seed as follows:

     To obtain a seed value for the PRF+ function (see [RFC4306], section 2.13) in order to generate an IPMK, an implementation MUST create a byte array containing the ASCII values for the string "Inner Methods Compound Keys" and MUST concatenate the ISK as follows (where "|" denotes concatenation of strings):

     ```
     IPMK Seed = "Inner Methods Compound Keys" | ISK
     ```

3. Generate the IPMK and CMK as follows:

   To generate the IPMK, implementations MUST use the first 40 octets of TK (see section 3.1.5.5.2.1), and MUST use the PRF+ seed value as the input to a PRF+ operation, and MUST generate 60 bytes. The first 40 bytes are the IPMK, while the last 20 bytes are the CMK.

```
TempKey = First 40 octets of TK
IPMK = First 40 octets of PRF+ (TempKey, IPMK Seed, 60);
```

   The PRF+ function (see [RFC4306], section 2.13) is extended to accept the following parameters:

```
K = Key, S = Seed, LEN = output length
```

   The following algorithm is used by the PRF+ function ("|" denotes concatenation):

   In generating IPMK and CMK, 60 bytes are required. Therefore, LEN=60 in this case.

```
PRF+(K, S, LEN) = T1 | T2 | ... |Tn
Where:
T1 = HMAC-SHA1 (K, S | 0x01 | 0x00 | 0x00)
T2 = HMAC-SHA1 (K, T1 | S | 0x02 | 0x00 | 0x00)

...

Tn = HMAC-SHA1 (K, Tn-1 | S | n | 0x00 | 0x00)
```

   As shown, PRF+ is computed in iterations. The number of iterations (n) depends on the output length (LEN). The computation stops when the concatenated length of T1, T2, ..., Tn is equal to or greater than the output length. When calculating IPMK and CMK, required output length is 60 bytes (LEN=60). Because each HMAC-SHA1 operation generates 20 bytes, n=3 iterations (that is, T1, T2 and T3) are required to compute IPMK and CMK.

   The preceding PRF+ definition is valid only when LEN < 256 and n < 256.

3.1.5.6 Phase 2 (EAP Encapsulation)

Once TLS Tunnel Establishment successfully completes, all subsequent EAP messages are exchanged inside the tunnel established in phase 1. The exceptions are the EAP success or the EAP failure packets (as specified in [RFC3748] section 4.2), which are never sent within the tunnel because result indications are handled by the CEAP implementation itself instead of the CEAP method (via the Result TLV (section 2.2.4.2.2)).

CEAP

Likewise, CEAP can decompress an EAP packet before passing it to an CEAP method for processing. It does this by setting the **Code** and **Identifier** fields of the CEAP packet to the values stored in the **Code** and **Identifier** fields of the outer EAP packet, and by setting the **Length** field of the CEAP packet to the length of the decrypted CEAP message plus 4. This decompression scheme MUST be applied to all CEAP method types except for the EAP TLV Extensions Method, the Capabilities Negotiation Method, and the SoH EAP Extensions Method; in these cases, the decompression scheme MUST NOT be used.

CEAP implementations MUST only support a single EAP authentication method per session with a type greater than or equal to 4, in addition to supporting EAP TLV Extensions Method (and optionally SoH EAP Extensions Method) in the same session.

3.1.5.7 Key Management

CEAP methods MUST generate MPPE keys as follows.

1. If a CEAP server and CEAP peer have successfully exchanged cryptobinding TLVs, then the keys are generated as follows:

    1. The Compound Session Key (CSK) is derived with the following equation.

        ```
        CSK = PRF+ (IPMK, "Session Key Generating Function", 128)
        ```

        The output length of the CSK MUST be 128 bytes. IPMK and PRF+ function is defined in section 3.1.5.5.2.2.

        For the seed value for the PRF+ function for the CSK, an implementation MUST create a byte array containing the ASCII values for the string "Session Key Generating Function" appended with a NULL(0x00) byte.

    2. The first 64 bytes of the CSK are split into two MPPE keys, as follows.

	First 32 bytes of CSK	Second 32 bytes of CSK
CEAP peer	MS-MPPE-Send-Key	MS-MPPE-Recv-Key
CEAP server	MS-MPPE-Recv-Key	MS-MPPE-Send-Key

2. When an endpoint (either a CEAP server or CEAP peer) is incapable of sending cryptobinding TLVs, and the other endpoint is configured to accept such authentications, then the keys are obtained from the TK (see section 3.1.5.5.2.1).

	First 32 bytes of TK	Second 32 bytes of TK
CEAP peer	MS-MPPE-Send-Key	MS-MPPE-Recv-Key
CEAP server	MS-MPPE-Recv-Key	MS-MPPE-Send-Key

3.1.6 Timer Events

CEAP relies on the timer events in EAP, as specified in [RFC3748] section 4.3.

3.1.7 Other Local Events

CEAP relies on the TLS Protocol, as specified in [RFC2246], for session disconnects and other conditions that may occur during the course of a TLS session.

3.1.7.1 SendRadiusReply

This event MUST be signaled by the Send SoHR Task specified in [MS-NAPSO] section 9 with the following arguments:

- An SoHR ([MS-SOH] section 2.2.5)

If the higher-layer business logic completes an SoHR request as specified in section 3.2.5.4.5, the CEAP Peer MUST perform the following steps:

- Prepare an SoH TLV (section 2.2.4.1.1.2) with the provided SoH message.
- Encrypt the SoH TLV by passing it to the TLS layer using the EncryptMessage method.
- Prepare a CEAP packet using the encrypted result as the Data field.
- Send the CEAP packet to the server as specified in section 3.1.5.2.2.

3.2 CEAP Peer Details

3.2.1 Abstract Data Model

This section describes a model of possible data organization that a peer-side implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that an implementation adhere to this model as long as the external behavior of the implementation is consistent with what is described in this specification.

The CEAP peer MUST maintain all the data elements as specified in section 3.1.1.

The CEAP peer participating in this protocol MUST also maintain the following data elements:

currentState: A 4-byte unsigned integer indicating the current state of authentication.

innerEAPAuthenticationMethods: An array of unsigned integers whose values correspond to the EAP authentication method types ([IANA-EAP]) supported by the CEAP implementation.

isFastReconnectConfigured: A Boolean flag indicating whether **fast reconnect** is configured to be allowed (TRUE) or not allowed (FALSE) for the session.

InnerEapType: A 4-byte unsigned integer that indicates the Extensible Authentication Protocol (EAP) type (section 5 of [RFC3748]) of the EAP authentication method.

isIdPrivacyEnabled: A Boolean flag indicating whether **Identity Privacy** feature is enabled (TRUE) or not (FALSE) for the session.<7>

IdentityPrivacyString: A NULL terminated Unicode string indicating the identity to be used in the outer EAP-Identity response packet.<8>

isValidateServerCertEnabled: A Boolean flag indicating whether a server certificate should (TRUE) or should not (FALSE) be validated for the session.

ServerNames: An array of NULL terminated Unicode strings indicating the names of authenticating servers that the client configured to authenticate to.

isValidateServerNameEnabled: A Boolean flag indicating whether the subject name of the server certificate should (TRUE) or should not (FALSE) be validated against the configured **ServerNames** for the session.

isPromptForValidationDisabled: A Boolean flag indicating whether a user can (TRUE) or cannot (FALSE) be prompted to override the validation failures on the server certificate.

TrustedCertHashInfoList: An array of 20-byte SHA1 hash ([RFC3174]) specifying the subset of certificates from a **trust root** that needs to be used by the peer to validate the trust anchor (section 6 of [RFC5280]) of the server certificate obtained during the TLS Tunnel Establishment TLS Tunnel establishment.

currentState is initialized when the CEAP peer starts authentication and remains valid until the authentication is done. At any point in time, **currentState** can have one of the following enumeration values, each one representing the current state of the PEAP peer.

- CEAP_BEGIN
- TUNNEL ESTABLISHED
- INNER_EAP_INPROGRESS
- INNER_IDENTITY_SENT
- SUCCESS_TLV_SENT
- FAILURE_TLV_SENT
- CEAP_SUCCESS
- CEAP_FAILED

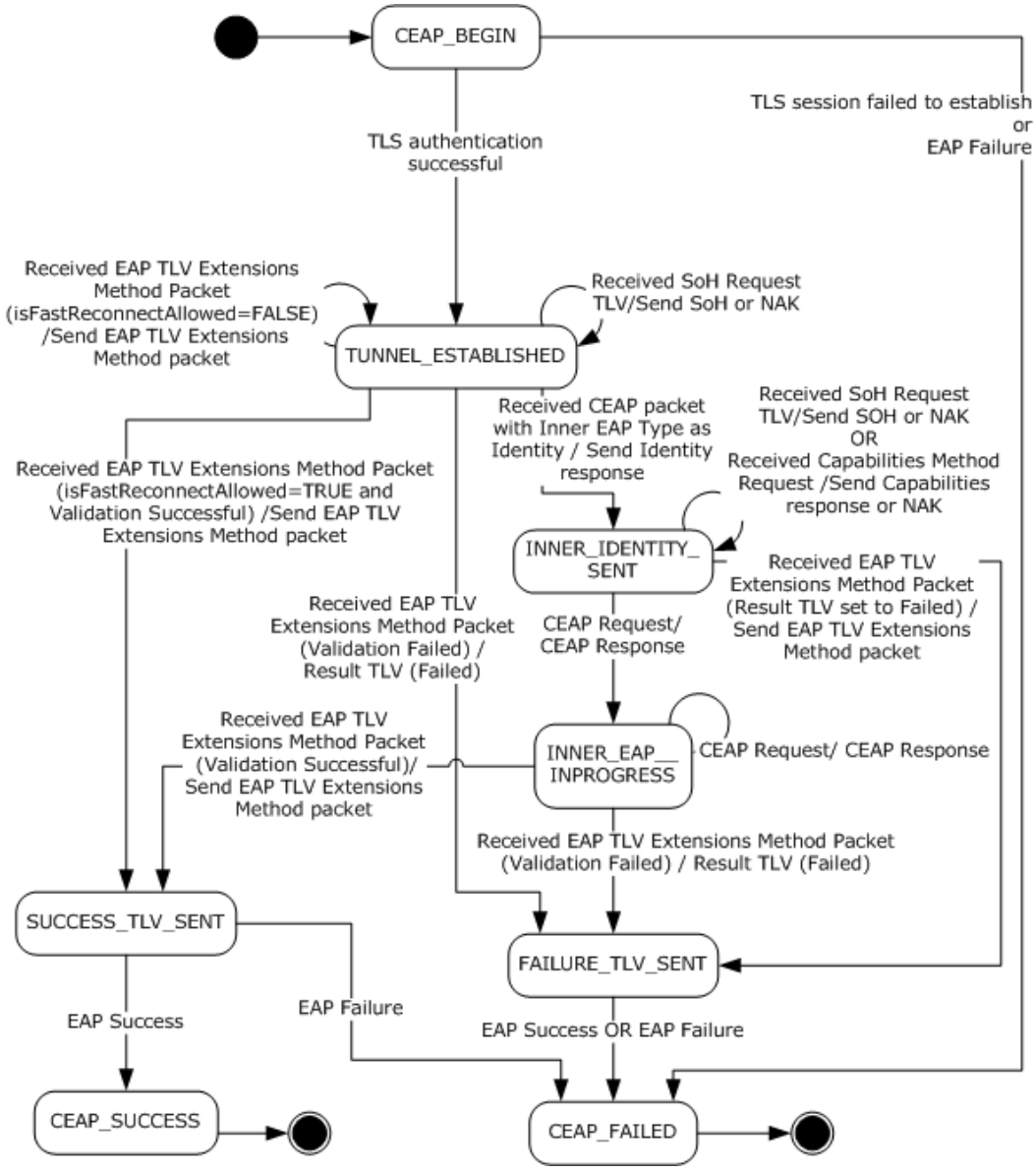


Figure 3: CEAP Peer State Machine

3.2.2 Timers

None.

3.2.3 Initialization

Use of EAP is triggered by attempts to access the network. A transport (such as [IEEE802.1X]) is typically invoked, which in turn invokes EAP, which results in an EAP server proposing use of PEAP as the authentication method and which ultimately invokes CEAP.

CEAP MUST be initialized on the peer by PEAP when it is invoked by EAP as the authentication method (EAP Type=25), as specified in [RFC3748] section 5. The CEAP peer MUST be initialized with the following data element:

- **MaxSendPacketSize**: An integer indicating the maximum EAP packet size.

The following initialization processing MUST also be performed:

- **currentState** MUST be initialized to CEAP_BEGIN.
- **isFastReconnectAllowed** MUST be initialized to FALSE.
- **BypassCapNegotiation** MUST be initialized in an implementation-specific manner.<9>
- **AssumePhase2Frag** MUST be initialized in an implementation-specific manner.<10>
- **innerEAPAuthenticationMethods** MUST be initialized in an implementation-specific manner. The list of Authentication methods MUST correspond with the Authentication methods supported by the CEAP implementation.
- **isFragmentationCapabilitySupported** MUST be initialized to TRUE, if the CEAP method implementation supports phase 2 fragmentation and either of the following conditions are true. Otherwise, it is initialized to FALSE.
  - **BypassCapNegotiation** is FALSE.
  - **BypassCapNegotiation** is TRUE and **AssumePhase2Frag** is TRUE.

The data elements listed in the following table are initialized on the CEAP peer from BLOB-based wireless profiles, as specified in [MS-GPWL] section 2.2.3.1, or from XML-based wired and wireless profiles as specified in section 2.2.3.2:

Abstract Data Model (ADM) element	BLOB element from [MS-GPWL]	Schema element from [MS-GPWL]
isSoHEnabled	PeapEnableQuarantine (2.2.3.1.2)	EnableQuarantineChecks (2.2.3.2.6)
isCryptoRequired	PeapEnforceCryptoBinding (2.2.3.1.2)	RequireCryptoBinding (2.2.3.2.6)
isFastReconnectAllowed	PeapFastRoaming (2.2.3.1.2)	FastReconnect (2.2.3.2.6)
InnerEapType	InnerEapType (2.2.3.1.2.2)	baseEap:Eap (2.2.3.2.4)
isIdPrivacyEnabled	PeapEnableIdentityPrivacy (2.2.3.1.2)	EnableIdentityPrivacy (2.2.3.2.6)
IdentityPrivacyString	IdentityPrivacyString (2.2.3.1.2)	AnonymousUserName (2.2.3.2.6)
isValidateServerCertEnabled	PeapTlsPhase1NoValidateServerCert (2.2.3.1.2.1)	PerformServerValidation (2.2.3.2.5)
isValidateServerNameEnabled	PeapTlsPhase1NoValidateName (2.2.3.1.2.1)	AcceptServerName (2.2.3.2.5)
isPromptForValidationDisabled	PeapTlsPhase1DisablePromptValidation (2.2.3.1.2.1)	DisableUserPromptForServerValidation (2.2.3.2.8)

Abstract Data Model (ADM) element	BLOB element from [MS-GPWL]	Schema element from [MS-GPWL]
ServerNames	ServerName (2.2.3.1.2.1)	ServerNames (2.2.3.2.8)
TrustedCertHashInfoList	**TrustedCertHashInfoList** (2.2.3.1.2.1) **NumberOfCAs** (2.2.3.1.2.1) field indicates the number of elements in the **TrustedCertHashInfoList** ADM element.	TrustedRootCA (2.2.3.2.8) Number of <TrustedRootCA> elements (2.2.3.2.8) indicates the number of elements in the **TrustedCertHashInfoList** ADM element.

3.2.4 Higher-Layer Triggered Events

The CEAP peer MUST be prepared to process a set of higher-layer events described in this section.

If the processing of a higher-layer triggered event requires receipt of a response message from the CEAP server, the associated Message Processing event (section 3.2.5) MUST communicate to the higher-layer business logic the result of the request initiated by the higher-layer triggered event processing.

3.2.4.1 Send SoH TLV

This event MUST be signaled by the higher-layer with the following arguments:

- An SoH ([MS-SOH] section 2.2.5)

If the higher-layer business logic completes an SoHR request as specified in section 3.2.5.4.5, the CEAP Peer MUST perform the following steps:

- Prepare an SoH TLV (section 2.2.4.1.1.2) with the provided SoH message.
- Encrypt the SoH TLV by passing it to the TLS layer using the EncryptMessage method.
- Prepare a CEAP packet using the encrypted result as the Data field.
- Send the CEAP packet to the server as specified in section 3.1.5.2.2.

3.2.4.2 Retrieve Privacy Identity String

This event MUST be signaled by the higher-layer when it requires an EAP method to supply a privacy identity.

- If isIdPrivacyEnabled is set to TRUE, return the IdentityPrivacyString to the higher-layer.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Status and Error Handling

Status and error handling are specified in section 3.1.5.1.

3.2.5.2 TLS Tunnel Establishment (TLS Tunnel Establishment)

The first CEAP packet received from the CEAP server is the CEAP start packet. It specifies the version of the CEAP protocol and indicates that the CEAP server is prepared to begin the CEAP TLS Tunnel Establishment negotiation. Implementations MUST reset the TLS session upon receiving a

CEAP packet with the S flag on packets other than the first packet. Implementations MUST set the EAP **Type** field of all CEAP packets to 25 (CEAP).

Once the CEAP version is negotiated, all subsequent CEAP request and response packets MUST include the negotiated version. The CEAP peer MUST set the CEAP version to 0 in CEAP responses, regardless of the version sent in the initial or subsequent CEAP requests. The CEAP server MUST set the CEAP version to 0 in CEAP requests. When a peer negotiates a version other than zero, the CEAP server MUST fail the authentication by sending an EAP failure packet.

The CEAP peer response begins the negotiation of a TLS session (as specified in [RFC2246]) with the CEAP server. The TLS tunnel can be established via a TLS session resume (as specified in [RFC2246] section F.1.4).

Note that CEAP relies on the TLS Protocol [RFC2246] to manage the TLS session (including the handling of any error or other conditions that may occur within the TLS Protocol). The TLS packets are exchanged encapsulated in CEAP packets as explained in section 3.1.5.4.

3.2.5.3 CEAP Peer Cryptobinding Validation

Upon receipt of the cryptobinding request, the CEAP peer MUST validate the message using the following process.

The cryptobinding TLV MUST specify the appropriate subtype (for example, a request must specify a request and a response must specify a response); otherwise the validation is declared as failed.

The CEAP peer MUST then construct the cryptobinding structure (see cryptobinding TLV), populating its **Nonce** field with the nonce supplied in the corresponding cryptobinding request. The implementation then MUST compute the Compound MAC as specified in 3.1.5.5.

A CEAP peer implementation MUST then compare the Compound MAC contained in the cryptobinding request with the Compound MAC that the peer itself computed. If the Compound MACs do not match, then the validation is declared as failed; otherwise, the validation is declared as success.

3.2.5.4 Packet Processing

If a packet is received with L and M bits set, then reassembly is done as specified in section 3.1.5.2.1. After reassembly, the packet is processed as specified in the following sections.

3.2.5.4.1 General Packet Validation

When receiving a packet, the CEAP peer MUST validate that the packet conforms to the syntax as specified in Message Syntax (section 2.2) and its subsections. If an invalid packet is received, the error is handled as specified in section 3.2.5.1.

3.2.5.4.2 Received CEAP Request

If the **currentState** variable is set to TUNNEL ESTABLISHED, then:

1. Change the **Type** field in the CEAP packet to EAP-TLS [IANA-EAP], and process the packet as specified in [RFC5216].
2. Prepare the EAP Response packet as specified in [RFC5216].
3. Change the **Type** field to CEAP, and then send the packet to the server.

If **currentState** is set to TUNNEL_ESTABLISHED, INNER_IDENTITY_SENT, or INNER_EAP_INPROGRESS, then:

1. Pass the **Data** field in the CEAP packet to the TLS layer for decryption using the **DecryptMessage** method.
2. If the decrypted data returned by **DecryptMessage** is compressed data, apply the decompression method as specified in section 3.1.5.6.
3. Compare the data for the respective message on each state and then process further based on the value of the **currentState** and the contents of the decrypted data as described in the state diagram and processing steps in this section.

If **currentState** is not set to TUNNEL ESTABLISHED, TUNNEL_ESTABLISHED, INNER_IDENTITY_SENT, or INNER_EAP_INPROGRESS, then the packet is ignored.

3.2.5.4.3 Received CEAP Packet with S Bit Set

If the **currentState** variable is set to CEAP_BEGIN, then:

1. Change the **Type** field in the CEAP packet to EAP-TLS [IANA-EAP], and process the packet as specified in [RFC5216].
2. Prepare the EAP Response packet as specified in [RFC5216].
3. Change the **Type** field to CEAP, and then send the packet to the server.
4. Change **currentState** to CEAP_PHASE1_IN_PROGRESS.

If **currentState** is not set to CEAP_BEGIN, then the packet is ignored.

3.2.5.4.4 Received CEAP Packet With CEAP Type As Identity

If the **currentState** variable is set to TUNNEL_ESTABLISHED, then:

1. Get the Identity of the peer to be authenticated from the protocol to be tunneled. For an example, see [MS-CHAP] section 3.2.4, which explains how to get the Identity for the Extensible Authentication Protocol Method for the Microsoft Challenge Handshake Authentication Protocol (CHAP).
2. Prepare an **EAP Identity** response packet [RFC3748] with the Identity obtained in step 1 as **Type_Data** value.
3. Compress the EAP packet obtained in step 2 as specified in section 3.1.5.6, and then encrypt the compressed data by passing it to the TLS layer using the **EncryptMessage** method.
4. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Then, send the CEAP packet to the server (see section 3.1.5.2.2).
5. Change **currentState** to INNER_IDENTITY_SENT.

If **currentState** is not set to TUNNEL_ESTABLISHED, then the packet is ignored.

3.2.5.4.5 Received SoH Request TLV

If the **currentState** variable is set to TUNNEL_ESTABLISHED or INNER_IDENTITY_SENT, then:

- If **isSoHEnabled** is set to FALSE:

  1. Prepare an EAP NAK packet as per [RFC3748].

  2. Compress the EAP packet obtained in step 1 (as specified in section 3.1.5.6), and encrypt the compressed data by passing it to the TLS layer using the **EncryptMessage** method.

  3. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Then, send the CEAP packet to the server (see section 3.1.5.2.2).

- If **isSoHEnabled** is set to TRUE:

  - Notify the higher-layer of an SoH request TLV.

CEAPOtherwise, the packet is ignored.

3.2.5.4.6 Received Capabilities Method Request

If the **currentState** variable is set to INNER_IDENTITY_SENT, then:

1. If **isCapabilitiesSupported** is set to TRUE, prepare a Capabilities Method Response (section 3.3.5.4.4) packet with the F flag set to **isFragmentationCapabilitySupported**.<11> If the F flag of the received packet is set to zero, set **isFragmentationCapabilitySupported** to FALSE.

2. If **isCapabilitiesSupported** is set to FALSE, prepare an EAP NAK packet as per [RFC3748] section 5.3.

3. Compress the EAP packet (as specified in section 3.1.5.6) obtained above and then encrypt the compressed data by passing it to the TLS layer using the **EncryptMessage** method.

4. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Then, send the CEAP packet to the server (see section 3.1.5.2.2).

If **currentState** is not set to INNER_IDENTITY_SENT, then the packet is ignored.

3.2.5.4.7 Received EAP TLV Extensions Method Packet

If the **currentState** datum is set to TUNNEL_ESTABLISHED or INNER_EAP_INPROGRESS, then the following steps are applied in sequence:

1. If a Result TLV (section 2.2.4.2.2) is received with the **value** field set to 2, then prepare an EAP TLV Extensions Method (section 3.2.5.4.7) packet with Result TLV (the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 11.

2. If the **currentState** datum is set to INNER_EAP_INPROGRESS and the authentication result flag returned by **isEAPAuthSuccess** indicates FALSE, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 11.

3. If the **currentState** datum is set to INNER_EAP_INPROGRESS and the authentication result flag returned by **isEAPAuthSuccess** indicates TRUE, then store the **InnerMPPESendKey**, **InnerMPPESendKeyLength**, **InnerMPPERecvKey**, and **InnerMPPERecvKeyLength** as returned by **isEAPAuthSuccess**.

4. If the **currentState** datum is set to TUNNEL_ESTABLISHED and **isFastReconnectAllowed** is set to FALSE, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2) and keep the **currentState** datum set to the same value and proceed to step 11.

5. If the **currentState** datum is set to TUNNEL_ESTABLISHED and **isFastReconnectAllowed** is set to TRUE, but the peer cannot start fast reconnect because of implementation defined reasons, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2) and keep the **currentState** datum set to the same value. Set **isFastReconnectAllowed** to FALSE and proceed to step 11.

6. If **isCryptoSupported** is set to TRUE and a Cryptobinding TLV (section 2.2.4.2.1) is received whose validation (described in section 3.2.5.3) fails, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 11.

7. If **isCryptoSupported** is set to TRUE, **isCryptoRequired** is set to TRUE and the received packet has only a Result TLV (the **value** field set to 1), then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2). If the **currentState** datum is set to INNER_EAP_INPROGRESS then change it to **FAILURE_TLV_SENT** and proceed to step 11. If the **currentState** datum is set to TUNNEL_ESTABLISHED, then keep it the same and proceed to step 11.

8. If the received EAP TLV Extensions Method packet contains both a Cryptobinding TLV and a Result TLV, and **isCryptoSupported** is set to TRUE, then prepare an EAP TLV Extensions Method packet with both Result TLV (the **value** field set to 1) and Cryptobinding TLV (the **value** field set to the computed value). Change the **currentState** datum to SUCCESS_TLV_SENT and proceed to step 11.

9. If the received EAP TLV Extensions Method packet contains both a Cryptobinding TLV and a Result TLV, and **isCryptoSupported** is set to FALSE, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 1). Change the **currentState** datum to SUCCESS_TLV_SENT and proceed to step 11.

10. If the received EAP TLV Extensions Method packet contains only a Result TLV and no Cryptobinding TLV, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 1). Change the **currentState** datum to SUCCESS_TLV_SENT and stop processing the packet.

11. If the received packet does not meet any of the above conditions, then ignore the packet and keep the **currentState** datum set to the same value.

12. If the prepared EAP TLV Extensions Method packet has a Result TLV with the **value** field set to 1, pass the **SoHR** message, if any, from the received EAP TLV Extensions Method packet by calling **EapProcessSoHRData**.

13. Encrypt the EAP TLV Extensions Method packet obtained above by passing it to the TLS layer using the **EncryptMessage** method.

14. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Then, send the CEAP packet to the server (see section 3.1.5.2.2).

If the **currentState** datum is set to INNER_IDENTITY_SENT, then:

1. If a Result TLV is received with the **value** field set to 2, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT.

2. If the received packet does not meet the above condition, then ignore the packet,keep the **currentState** datum set to the same value, and stop processing the packet.

3. Encrypt the EAP TLV Extensions Method packet obtained above by passing it to the TLS layer using the **EncryptMessage** method.

4. Prepare a CEAP packet, keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field. Then, send the CEAP packet to the server (see section 3.1.5.2.2).

If the **currentState** datum is not set to TUNNEL_ESTABLISHED, INNER_EAP_INPROGRESS, or INNER_IDENTITY_SENT, then the packet is ignored.

3.2.5.4.8 Received EAP Success

If **currentState** is set to SUCCESS_TLV_SENT, then:

1. Trigger the Transport Layer with authentication result as Success.

2. Change **currentState** to CEAP_SUCCESS.

If **currentState** is set to FAILURE_TLV_SENT, then:

1. Trigger the Transport Layer with authentication result as failed.

2. Change **currentState** to CEAP_FAILED.

If **currentState** is not set to SUCCESS_TLV_SENT or FAILURE_TLV_SENT, then the packet is ignored.

3.2.5.4.9 Received EAP Failure

If **currentState** is set to SUCCESS_TLV_SENT, FAILURE_TLV_SENT, or TUNNEL ESTABLISHED, then:

1. Trigger the Transport Layer with the authentication result as Failed.

2. Change **currentState** to CEAP_FAILED.

If **currentState** is not set to SUCCESS_TLV_SENT, FAILURE_TLV_SENT, or TUNNEL ESTABLISHED, then the packet is ignored.

3.2.5.5 Key Management

See section 3.1.5.7.

3.2.6 Timer Events

For details on timer events, see section 3.1.6.

3.2.7 Other Local Events

Note that CEAP relies on the TLS Protocol [RFC2246] for session disconnects and other conditions that can occur during the course of a TLS session. The local events generated by EAP_TLS and consumed by the CEAP layer are described in the following sections.

3.2.7.1 TLS Session Established Successfully

If the TLS session established successfully:

inputParameter: **TLS message**

outputParamter:

- **CtxtHandle** (a context handle returned by TLS layer)
- **Server Certificate** (The certificate as received from the server by the TLS layer. The server certificate is a X.509 certificate as described in [RFC5280]. It is made available as part of the TLS handshake as specified in section 7.4.2 of [RFC2246].)
- **isSessionResumed** (a Boolean flag indicating whether the underlying TLS session is resumed (as defined in sections 7.3 and F.1.4 of [RFC2246]); TRUE indicates that the TLS session is resumed.)

This event will be received from the TLS layer in response to a TLS message passed to it by the CEAP layer during phase 1. If the **currentState** variable is not set to **CEAP_PHASE1_INPROGRESS**, ignore this event. Otherwise, the CEAP layer MUST take the following actions:

1. The following processing MUST be done if isValidateServerCertEnabled is TRUE:
    1. The trust anchor of the server certificate MUST be validated against the certificates in a trust root <12>as specified in section 6.1 of [RFC5280]. If the validation fails, then prepare a TLS alert message with **AlertDescription** set to unknown_ca (section 7.2 of [RFC2246]) and go to Step 5.
    2. Validate that the SHA1 hash ([RFC3174]) of the certificate which matched the trust anchor of the server certificate in the preceding step is present in **TrustedCertHashInfoList**.
    3. If the **isValidateServerNameEnabled** is set to TRUE, then verify that the subject name (Section 4.1.2.6 of [RFC5280]) or subject alternative name (section 4.2.1.6 of [RFC5280]) of the server certificate exists in **ServerNames**.
    4. If any of the validations in either of the two preceding steps fail and **isPromptForValidationDisabled** is set to FALSE, the implementation could take user's consent on whether the authentication can be succeeded. If the user has chosen to fail the authentication, or if **isPromptForValidationDisabled** is set to TRUE and validations in either of the two preceding steps fail, prepare a TLS Alert message with **AlertDescription** set to access_denied (section 7.2 [RFC2246]). The **currentState** should continue to be same. Go to Step 5.
2. Store the **CtxtHandle** returned by the TLS layer.
3. If **isSessionResumed** and **isFastReconnectConfigured** are set to TRUE, then set **isFastReconnectAllowed** to TRUE; otherwise set it to FALSE.
4. Change **currentState** to TUNNEL_ESTABLISHED.
5. Prepare an EAP response packet as specified in [RFC5216] section 3.2.
6. Change the packet **Type** field to CEAP [IANA-EAP], and then send the packet to the server.

3.2.7.2 TLS Session Failed to Establish

If the TLS session failed to establish:

- This event will be received from the TLS layer when it is unsuccessful in establishing the TLS session. If **currentState** is not set to TUNNEL ESTABLISHED, ignore this event. Otherwise, the CEAP layer MUST take the following action:
  - Change **currentState** to CEAP_FAILED.

3.3 CEAP Server Details

3.3.1 Abstract Data Model

This section describes a model of possible data organization that a server-side implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that an implementation adhere to this model as long as the external behavior of the implementation is consistent with what is described in this specification.

The CEAP server MUST maintain all the data elements as specified in section 3.1.1.

The CEAP server MUST also maintains the following data elements:

innerEAPAuthenticationMethods: An array of unsigned integers whose values correspond to the EAP authentication method types ([IANA-EAP]) supported by the CEAP implementation.

currentState: Initialized when the server starts the CEAP authentication and remains valid until the authentication is done. At any point in time, **currentState** can have one of the following enumeration value, each of which represents a possible state of the CEAP server.

- TUNNEL ESTABLISHED
- WAIT_FOR_SOH_RESPONSE
- WAIT_FOR_CAPABILITIES_RESPONSE
- INNER_IDENTITY_REQ_SENT
- INNER_EAP_INPROGRESS
- SUCCESS_TLV_SENT
- FAILURE_TLV_SENT
- CEAP_SUCCESS
- CEAP_FAILED

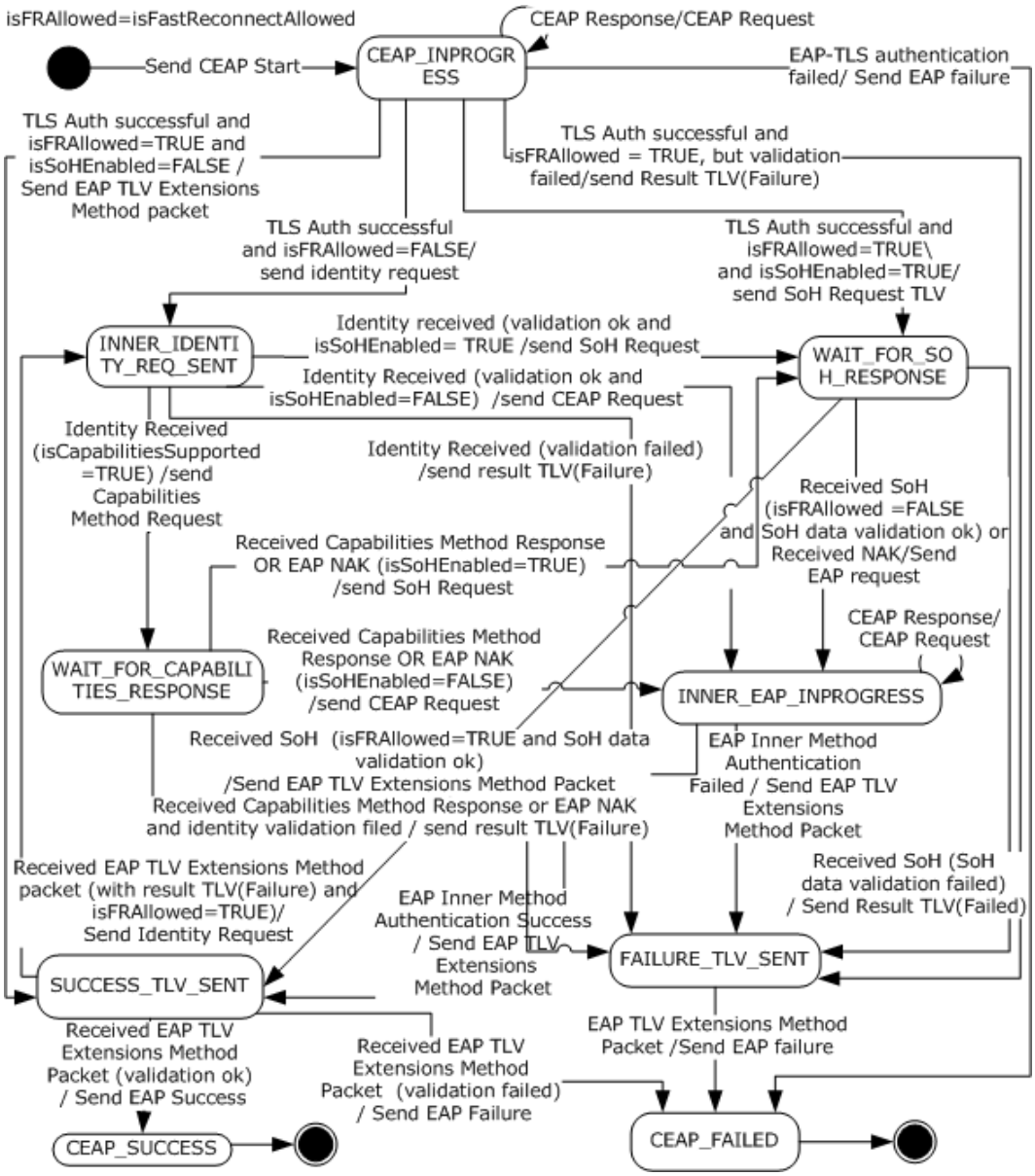


Figure 4: CEAP Server State Machine

3.3.2 Timers

None.

3.3.3 Initialization

CEAP MUST be initialized on the EAP server when it is invoked by EAP as an authentication method. This occurs when an EAP-enabled protocol (such as RADIUS [RFC2865]) invokes EAP, the EAP server proposes CEAP, and the peer agrees to perform CEAPauthentication. The CEAP server MUST be initialized with the following data element:

- **MaxSendPacketSize**: An integer indicating the maximum EAP packet size.

CEAPThe following initialization processing MUST also be performed:

- **BypassCapNegotiation** MUST be initialized in an implementation-specific manner.<13>
- **AssumePhase2Frag** MUST be initialized in an implementation-specific manner.<14>
- **isFragmentationCapabilitySupported** MUST be initialized to TRUE, if the CEAP method implementation supports phase 2 fragmentation and either of the following conditions are true. Otherwise, it is initialized to FALSE.
  - **BypassCapNegotiation** is FALSE.
  - **BypassCapNegotiation** is TRUE and **AssumePhase2Frag** is TRUE.

3.3.4 Higher-Layer Triggered Events

The CEAP server MUST be prepared to process a set of higher-layer events described in this section.

If the processing of a higher-layer triggered event requires receipt of a response message from the EAP peer, the associated Message Processing event (section 3.3.5) MUST communicate to the higher-layer business logic the result of the request initiated by the higher-layer triggered event processing.

3.3.4.1 SendRadiusReply

This event MUST be signaled by the Send SoHR Task specified in [MS-NAPSO] section 9 with the following arguments:

- An SoHR ([MS-SOH] section 2.2.6)

If the higher-layer business logic provides an SoHR to send to the CEAP peer, the CEAP server MUST perform the following steps:

If the **currentState** variable is set to WAIT_FOR_SOH_RESPONSE, then, return the SoH to the higher-layer business logic.

1. If **isFastReconnectAllowed** is set to FALSE, prepare an EAP Request packet to start the CEAP method negotiation as described in [RFC3748]. Compress the EAP Request packet as specified in section 3.1.5.6. Change **currentState** to INNER_EAP_INPROGRESS and proceed to step 3.
2. If **isFastReconnectAllowed** is set to TRUE, prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 1), SoH Response TLV (section 2.2.4.2.3) (the **value** field is obtained by calling EapGetSoHRData), and Cryptobinding TLV (section 2.2.4.2.1) (the **value** field set to the computed value) if **isCryptoSupported** is set to TRUE. Change **currentState** to SUCCESS_TLV_SENT and proceed to step 4.<15>
3. Encrypt the EAP TLV Extensions Method or EAP Request packet obtained in the preceding steps by passing it to the TLS layer using the **EncryptMessage** method.
4. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet.
5. Invoke the **SendRadiusReplyToPeer** event specified in [MS-RNAP] section 3.3.7.1 providing the CEAP packet as the value of an EAP-Message Radius Attribute.

Otherwise, the packet is ignored.

3.3.4.2 Send Invalid SoH Received

This event MUST be signaled by the higher-layer when the SoH returned by the message processing specified in section 3.3.5.4.5 is invalid.

If the **currentState** variable is set to WAIT_FOR_SOH_RESPONSE, then, return the SoH to the higher-layer business logic.

1. Prepare an EAP TLV Extensions Method (section 3.2.5.4.7) packet with Result TLV (section 2.2.4.2.2) (the **value** field set to 2). Change **currentState** to FAILURE_TLV_SENT.
2. Encrypt the EAP TLV Extensions Method or EAP Request packet obtained in the preceding step by passing it to the TLS layer using the **EncryptMessage** method.
3. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Send the CEAP packet to the peer (see section 3.1.5.2.2).
4. Invoke the **SendRadiusReplyToPeer** event specified in [MS-RNAP] section 3.3.7.1 providing the CEAP packet as the value of an EAP-Message Radius Attribute.

Otherwise, the packet is ignored.

3.3.5 Message Processing Events and Sequencing Rules

The CEAP message is processed by CEAP and then information is passed along to the NAP layer as described in [MS-NAPSO]. The attribute processing is described in sections 3.3.5.1 through 3.3.5.7.

After the attributed are processed the information is passed along to the NAP layer by calling the NAPSO "Connect TO NPS" task, [MS-NAPSO] section 7.3.3. "Connect To NPS" MUST be called with the following parameters:

- SOHRequest MUST be set to the value of the SOH parameter received from EAPE.
- SOHReqLength MUST be set to the value of the SOHReqLength parameter received from EAPE.
- NotQuarantineCapable MUST be set to the value of the NotQuarantineCapable parameter received from EAPE.
- ClientName MUST be set to the value of the ClientName parameter received from EAPE.
- MachineName MUST be set to the value of the MachineName parameter received from EAPE.
- ClientIPv4Address MUST be set to the value of the ClientIPv4Address parameter received from EAPE.
- ClientIPv6Address MUUST be set to the value of the ClientIPv6Address parameter received from EAPE.
- DhcpServiceClass MUST be set to the value of the DhcpServiceClass parameter received from EAPE.
- NASIdentifier MUST be set to the value of the NASIdentifier parameter received from EAPE.
- SecurityIdentity MUST be set to the value of the SecurityIdentity parameter received from EAPE.
- CallingStationID MUST be set to the value of the CallingStationID parameter received from EAPE.

- NetworkAccessServerType MUST be set to the value of the NetworkAccessServerType parameter received from EAPE.
- TunnelType MUST be set to the value of the TunnelType parameter received from EAPE.
- NASPortType MUST be set to the value of the NASPortType parameter received from EAPE.
- NASIPv4Address MUST be set to the value of the NASIPv4Address parameter received from EAPE.
- NASIpv6Address MUST be set to the value of NASIPv6Address parameter received from EAPE.
- HCAPLocationGroup MUST be set to the value of the HCAPLocationGroup parameter received from EAPE.
- HCAPUserGroup MUST be set to the value of HCAPUserGroup parameter received from EAPE.
- HCAPUserName MUST be set to the value of the HCAPUserName parameter received from EAPE.

3.3.5.1 Status and Error Handling

Status and error handling is specified in section 3.1.5.1.

3.3.5.2 CEAP TLS Tunnel Establishment

When the EAP implementation negotiates CEAP as the method on the EAP server, CEAP TLS Tunnel Establishment begins.

The first packet in a CEAP negotiation is referred to as a CEAP start packet. Version 0 implementations MUST set the **L** bit to 0, the M bit based on the description in the CEAP packet, the **S** bit to 1, and all of the reserved bits to 0. These flag fields are specified in the CEAP packet.

After the CEAP start packet is sent to the peer, the CEAP server expects a CEAP response from the peer that indicates the version of CEAP that the peer supports. At the EAP level (see section 2.1), these interactions are specified in [RFC3748] section 2.

The peer MUST then start to negotiate a TLS session.

When the TLS tunnel is established successfully, implementations SHOULD skip phase 2 if the session is a resumption of a previous session (as specified in [RFC2246] section F.1.4). This process is known as "fast reconnection".

3.3.5.3 CEAP Server Cryptobinding Validation

Upon receipt of the cryptobinding response, the CEAP server MUST validate the message using the following process.

The server implementation MUST construct the cryptobinding structure, populating its **Nonce** field with the nonce supplied in the corresponding cryptobinding response. The implementation MUST then compute the Compound MAC, as specified in section 3.1.5.5.

A CEAP server implementation MUST then compare the Compound MAC contained in the cryptobinding response with the Compound MAC that it computed. If the computed Compound MAC and the Compound MAC reported within the cryptobinding response do not match, then the validation is declared as failed. Otherwise it is declared as success.

3.3.5.4 Packet Processing

If a packet is received with L and M bits set, then reassembly is done as specified in section 3.1.5.2.1. After reassembly, the packet is processed as specified in the following sections.

3.3.5.4.1 General Packet Validation

When receiving a packet, the CEAP server MUST validate that the packet conforms to the syntax as specified in Message Syntax (section 3.3.5) and its subsections. If an invalid packet is received, the error is handled as specified in section 3.3.5.1.

3.3.5.4.2 Received CEAP Response

If the **currentState** variable is set to TUNNEL ESTABLISHED:

1. Change the **Type** field in CEAP packet to EAP-TLS (as specified in [IANA-EAP]), and process the packet as specified in [RFC5216].
2. Prepare the EAP Request packet as specified in [RFC5216].
3. Change the **Type** field to CEAP, then send the packet to the client.

If **currentState** is set to INNER_IDENTITY_REQ_SENT, WAIT_FOR_SOH_RESPONSE, WAIT_FOR_CAPABILITIES_RESPONSE, INNER_EAP_INPROGRESS, SUCCESS_TLV_SENT, or FAILURE_TLV_SENT:

1. Pass the **Data** field in the CEAP packet to the TLS layer for decryption using the **DecryptMessage** method.
2. If the decrypted data returned by **DecryptMessage** is compressed data as specified in 3.1.5.6, then apply the decompression method as specified in 3.1.5.6.
3. Compare the data for the respective message on each state and then process further based on the value of the **currentState** and the contents of the decrypted data as described in the "CEAP Server State Machine" state diagram (section 3.3.1) and processing steps in this section.

If **currentState** is not set to TUNNEL ESTABLISHED, INNER_IDENTITY_REQ_SENT, WAIT_FOR_SOH_RESPONSE, WAIT_FOR_CAPABILITIES_RESPONSE, INNER_EAP_INPROGRESS, SUCCESS_TLV_SENT, or FAILURE_TLV_SENT, then the packet is ignored.

3.3.5.4.3 Received CEAP Packet with CEAP Type As Identity (Identity Received)

If the **currentState** variable is set to INNER_IDENTITY_REQ_SENT, then the following steps MUST be applied in sequence:

1. Store the received identity in the **InnerIdentity** datum.
2. If the **isCapabilitiesSupported** field is set to TRUE, then prepare a Capabilities Method Request (section 3.2.5.4.6) packet with F flag set to **isFragmentationCapabilitySupported**.<16> Change the **currentState** datum to WAIT_FOR_CAPABILITIES_RESPONSE and proceed to step 6.
3. Validate the received Identity in an implementation-specific manner. If the Identity validation fails, then prepare an EAP TLV Extensions Method (section 3.2.5.4.7) packet with Result TLV (section 2.2.4.2.2) (the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 6.

4. If the **isSoHEnabled** field is set to TRUE then, prepare a SoH EAP Extensions Method (section 2.2.8.2) packet with a SoH Request TLV (section 2.2.4.1.1.1) within it. Change the **currentState** datum to WAIT_FOR_SOH_RESPONSE and proceed to step 6.

5. If all of the above conditions fail, then prepare an EAP Request packet to start the CEAP method negotiation as described in [RFC3748] section 2. Compress the EAP Request packet as specified in section 3.1.5.6. Change **currentState** to INNER_EAP_INPROGRESS.

6. Send the packet prepared above to the TLS layer for encryption using the **EncryptMessage** method.

7. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet, and send it to the peer (see section 3.1.5.2.2).

If **currentState** is not set to INNER_IDENTITY_REQ_SENT, then the packet is ignored.

3.3.5.4.4 Received Capabilities Method Response

If the **currentState** variable is set to WAIT_FOR_CAPABILITIES_RESPONSE, then:

1. If the F flag of the received Capabilities Method Response (section 3.3.5.4.4) packet is set to zero, set **isFragmentationCapabilitySupported** to FALSE.

2. Validate the Identity stored in the **InnerIdentity** datum in an implementation-specific manner. If the Identity validation fails, then prepare an EAP TLV Extensions Method packet (section 3.3.5.4.7) with Result TLV (section 2.2.4.2.2) (with the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 5.

3. If **isSoHEnabled** is set to TRUE, then prepare a SoH EAP Extensions Method (section 2.2.8.2) packet with SoH Request TLV (section 2.2.4.1.1.1) within it. Change **currentState** to WAIT_FOR_SOH_RESPONSE and proceed to step 5.

4. If **isSoHEnabled** is set to FALSE, then prepare an EAP Request packet to start the CEAP method negotiation as described in [RFC3748]. Compress the EAP Request packet as specified in section 3.1.5.6. Change **currentState** to INNER_EAP_INPROGRESS.

5. Send the prepared packet above to the TLS layer for encryption using the **EncryptMessage** method.

6. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet. Then, send it to the peer (see section 3.1.5.2.2).

If **currentState** is not set to WAIT_FOR_CAPABILITIES_RESPONSE, then the packet is ignored.

3.3.5.4.5 Received EAP NAK

If the **currentState** variable is set to WAIT_FOR_CAPABILITIES_RESPONSE, then:

1. Assign the variable **isFragmentationCapabilitySupported** to FALSE.

2. Validate the received Identity in an implementation-specific manner. If the Identity validation fails, then prepare an EAP TLV Extensions Method packet (section 3.2.5.4.7) with Result TLV (section 2.2.4.2.2) (with the **value** field set to 2). Change the **currentState** datum to FAILURE_TLV_SENT and proceed to step 5.

3. If the **isSoHEnabled** variable is set to TRUE, then prepare a SoH EAP Extensions Method packet with SoH Request TLV within it. Change **currentState** to WAIT_FOR_SOH_RESPONSE and proceed to step 5.

4. If **isSoHEnabled** is set to FALSE, then prepare an EAP Request packet to start the CEAP method negotiation as specified in [RFC3748]. Compress the EAP Request packet as specified in section 3.1.5.6. Change **currentState** to INNER_EAP_INPROGRESS.

5. Send the prepared packet above to the TLS layer for encryption using the **EncryptMessage** method.

6. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of CEAP packet. Then send it to the peer (see section 3.1.5.2.2).

If the currentState is set to WAIT_FOR_SOH_RESPONSE, then:

1. Prepare an EAP Request packet to start the CEAP method negotiation as specified in [RFC3748]. Compress the EAP Request packet as specified in section 3.1.5.6. Change **currentState** to INNER_EAP_INPROGRESS.

2. Encrypt the EAP TLV Extensions Method or EAP Request packet obtained in the preceding step by passing it to the TLS layer using the **EncryptMessage** method.

3. Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of CEAP packet. Then send it to the peer (see section 3.1.5.2.2).

If **currentState** is not set to WAIT_FOR_CAPABILITIES_RESPONSE or WAIT_FOR_SOH_RESPONSE, then the packet is ignored.

3.3.5.4.6 Received SoH

If the **currentState** variable is set to WAIT_FOR_SOH_RESPONSE, then, return the SoH to the higher-layer business logic.

Otherwise, the packet is ignored.

3.3.5.4.7 Received EAP TLV Extensions Method Packet

If **currentState** is set to FAILURE_TLV_SENT, then:

- If a Result TLV (section 2.2.4.2.2) is received with the **value** field set to 2, then send an EAP Failure packet (as specified in [RFC3748]) and change **currentState** to CEAP_FAILED.

If **currentState** is set to SUCCESS_TLV_SENT, then:

1. If the received packet does not have a Result TLV, then ignore it and stop processing the packet.

2. If a Result TLV is received with the **value** field set to 2 and **isFastReconnectAllowed** is set to TRUE, then prepare an EAP Request packet with the **Type** field as Identity (as specified in [RFC3748]).

   - Set **isFastReconnectAllowed** to FALSE, and change **currentState** to INNER_IDENTITY_REQ_SENT.

   - Compress the packet, and then encrypt it by passing it to the TLS layer using the **EncryptMessage** method.

- Prepare a CEAP packet by keeping the encrypted data returned by the **EncryptMessage** method as the **Data** field of the CEAP packet.
- Send the CEAP packet to the peer (see section 3.1.5.2.2).

This completes the processing of the packet.

3. If Result TLV is received with the **value** field set to 2, then send an EAP Failure packet (as specified in [RFC3748]) to peer. Change **currentState** to CEAP_FAILED. This completes the processing of the packet.
4. If **isCryptoSupported** is set to FALSE, then send an EAP Success packet (as specified in [RFC3748]) to peer. Change **currentState** to CEAP_SUCCESS. This completes the processing of the packet.
5. If the received packet contains a Cryptobinding TLV (section 2.2.4.2.1) whose validation (described in section 3.3.5.3) fails, then send a EAP Failure packet (as specified in [RFC3748]) to peer. Change **currentState** to CEAP_FAILED. This completes the processing of the packet.
6. If the received packet does not contain a Cryptobinding TLV and **isCryptoRequired** is set to TRUE, then send an EAP Failure packet (as specified in [RFC3748]) to peer. Change **currentState** to CEAP_FAILED. This completes the processing of the packet.
7. If the received packet does not satisfy any of the above conditions, then send an EAP Success packet (as specified in [RFC3748]) to peer. Change **currentState** to CEAP_SUCCESS.

If **currentState** is not set to FAILURE_TLV_SENT or SUCCESS_TLV_SENT, then the packet is ignored.

3.3.5.5 Key Management

See section 3.1.5.7.

3.3.6 Timer Events

See section 3.1.6.

3.3.7 Other Local Events

3.3.7.1 TLS Session Established Successfully

If the TLS session established successfully:

inputParameter: **TLS message**

outputParameter:

- **CtxtHandle** (a context handle returned by TLS layer)
- **isSessionResumed** (a Boolean flag indicating whether the underlying TLS session is resumed (as defined in sections 7.3 and F.1.4 of [RFC2246]); TRUE indicates that the TLS session is resumed.)

This event will be received from the TLS layer in response to a TLS message passed to it by the CEAP layer during phase 1. If the **currentState** variable is not set to TUNNEL ESTABLISHED, ignore this event. Otherwise, the CEAP layer MUST do the following steps in sequence:

1. Store the *isSessionResumed* to **isFastReconnectAllowed**.

2. If **isFastReconnectAllowed** is set to TRUE, but the server is not able to start fast reconnect because of implementation-defined reasons, then prepare an EAP Identity request packet. Compress the packet as described in section 3.1.5.6. Set **isFastReconnectAllowed** to FALSE. Change **currentState** to INNER_IDENTITY_SENT. Go to Step 7.

3. If **isFastReconnectAllowed** is set to TRUE, but the server cannot continue authentication because of implementation-defined reasons, then it MUST create an EAP TLV Extensions Method (section 3.2.5.4.7) packet with Result TLV (section 2.2.4.2.2) (the **value** field set to 2). Set **isFastReconnectAllowed** to FALSE. Change **currentState** to FAILURE_TLV_SENT. Got to Step 7.

4. If **isFastReconnectAllowed** is set to FALSE, then prepare an EAP Identity Request packet. Compress the packet as described in section 3.1.5.6. Change **currentState** to INNER_IDENTITY_REQ_SENT. Go to Step 7.

5. If **isFastReconnectAllowed** is set to TRUE and the **isSoHEnabled** field is set to TRUE, prepare a SoH EAP Extensions Method (section 2.2.8.2) packet with a SoH Request TLV (section 2.2.4.1.1.1) within it. Change **currentState** to WAIT_FOR_SOH_RESPONSE and proceed to step 7.

6. If the above conditions are not satisfied, then prepare an EAP TLV Extensions Method packet with Result TLV (the **value** field set to 1) and if **isCryptoSupported** is set to TRUE, then add a Cryptobinding TLV (section 2.2.4.2.1) (with a value generated by server, as described in section 3.3.5.3). Change **currentState** to SUCCESS_TLV_SENT. Go to Step 7.

7. Store the **CtxtHandle** returned by the TLS layer. Encrypt the packet generated above by passing it to the TLS layer using the **EncryptMessage** method, and after receiving the encrypted data, prepare a CEAP packet with the encrypted data as the **Data** field, and send it to the peer (see section 3.1.5.2.2). Change **currentState** to SUCCESS_TLV_SENT.

3.3.7.2 TLS Session Failed to Establish

If the TLS session failed to establish:

- This event will be received from the TLS layer when it is unsuccessful in establishing the TLS session. If **currentState** is not set to TUNNEL ESTABLISHED, ignore this event. Otherwise, the CEAP layer MUST do the following:

  1. Send an EAP failure packet to the peer.

  2. Change the **currentState** to CEAP_FAILED.

3.3.7.3 EAP Inner Method Authentication Success

Input: EAP Packet

Output: MPPE send and receive keys, and their lengths.

If EAP inner method authentication is successful, then:

- This event will be received from the respective CEAP method layer in response to an EAP packet passed to it. If **currentState** is not set to INNER_EAP_INPROGRESS, ignore this event. Otherwise, the CEAP layer MUST do the following:

1. Store **InnerMPPESendKey**, **InnerMPPESendKeyLength**, **InnerMPPERecvKey** and **InnerMPPERecvKeyLength** as returned by the CEAP method.

2. Create an EAP TLV Extensions Method (section 3.2.5.4.7) packet with Result TLV (section 2.2.4.2.2) (the **value** field set to 1) and if **isCryptoSupported** is set to TRUE, add a Cryptobinding TLV (section 2.2.4.2.1) (with a value generated by the server, as described in section 3.3.5.3) and if both peer and server have exchanged SoH Request (section 2.2.4.1.1.1) and SoH (section 2.2.4.1.1.2) TLVs, add a SoH Response TLV (section 2.2.4.2.3; the **value** field is obtained by calling **EapGetSoHRData**).

3. Encrypt the packet generated in the preceding step by passing it to the TLS layer using the **EncryptMessage** method, and after receiving the encrypted data, prepare a CEAP packet with encrypted data as the **Data** field and send it to the peer (see section 3.1.5.2.2). Change **currentState** to SUCCESS_TLV_SENT.

3.3.7.4 EAP Inner Method Authentication Failed

If EAP inner method authentication failed, then:

- This event will be received from the respective EAP method layer in response to an EAP packet passed to it. If **currentState** is not set to INNER_EAP_INPROGRESS, ignore this event. Otherwise, the CEAP layer SHOULD do the following:

  1. Create an EAP TLV Extensions Method (section 3.2.5.4.7) packet with result TLV (the **value** field set to 2).

  2. Encrypt the packet generated above by passing it to the TLS layer using the **EncryptMessage** method, and after receiving the encrypted data prepare a CEAP packet with encrypted data as **Type_Data** and send it to the peer. Change **currentState** to FAILURE_TLV_SENT.

4 Protocol Examples

The following sections provide common scenarios that illustrate the function of CEAP.

4.1 Examples with No Support for Cryptobinding and SoH Processing

This section provides examples of CEAP interactions when cryptobinding and SoH processing are supported by neither CEAP peer implementation nor CEAP server implementation.

4.1.1 Successful CEAP TLS Tunnel Establishment and 2 Negotiation

The following diagram depicts a complete CEAP authentication in which both TLS Tunnel Establishment and phase 2 negotiations take place successfully.

As the authentication begins with a CEAP packet with the S bit set being sent to the peer, TLS negotiation occurs until a TLS session has been established. Once the TLS session has been established (the end of CEAP phase 1), all traffic is subsequently encrypted between the CEAP peer and the server, and phase 2 has begun. Phase2 begins with CEAP capabilities negotiation. During phase 2, the CEAP method is negotiated and authentication occurs in a series of exchanges that depend upon the specific CEAP method that is used.

Phase 2 concludes with an exchange of the EAP Extensions Method with the Result TLV (with success in the following case) within the TLS session. Subsequently, and outside the TLS session, an EAP success packet is sent to the peer by the EAP server.

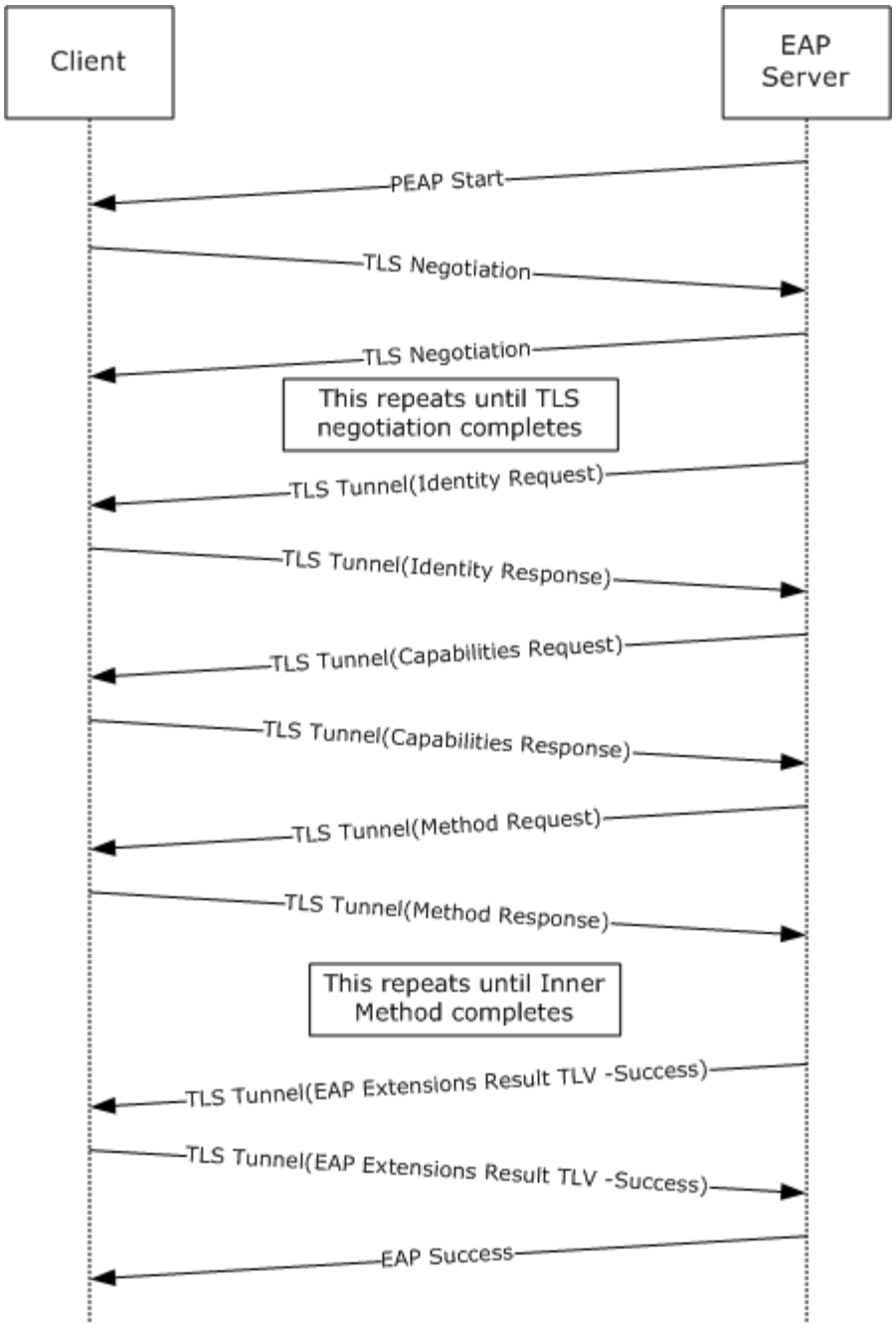


Figure 5: Successful CEAP TLS Tunnel Negotiation and Establishment

4.1.2 Successful CEAP TLS Tunnel Establishment with Failed Authentication Method Negotiation

The following diagram depicts a complete CEAP authentication in which TLS Tunnel Establishment completes successfully and phase 2 fails on the CEAP server side, with both the CEAP server and the peer not supporting cryptobinding and SoH TLVs.

This example is similar to the one described in section 4.1.1; however, note that the EAP Extensions Method with the Result TLV is a failure rather than a success, and the EAP failure packet is sent outside the TLS session.

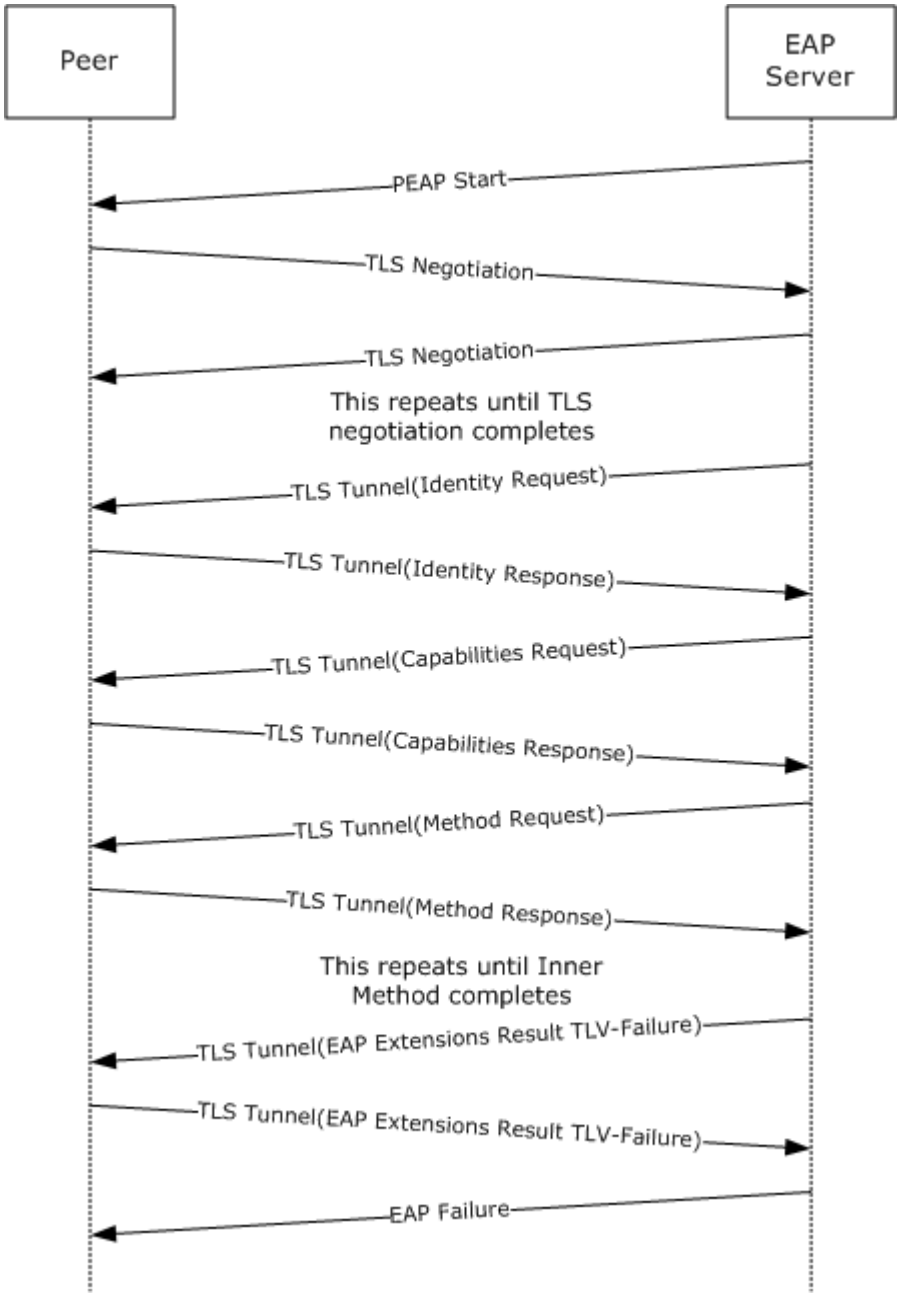


Figure 6: Successful CEAP TLS Tunnel Establishment with failed phase 2 negotiation

4.1.3 Successful CEAP TLS Tunnel Establishment with Fast Reconnect

The following diagram depicts a complete and successful CEAP authentication in which fast reconnect was used. Note that with fast reconnect, no CEAP authenticationor capabilities negotiation takes place.

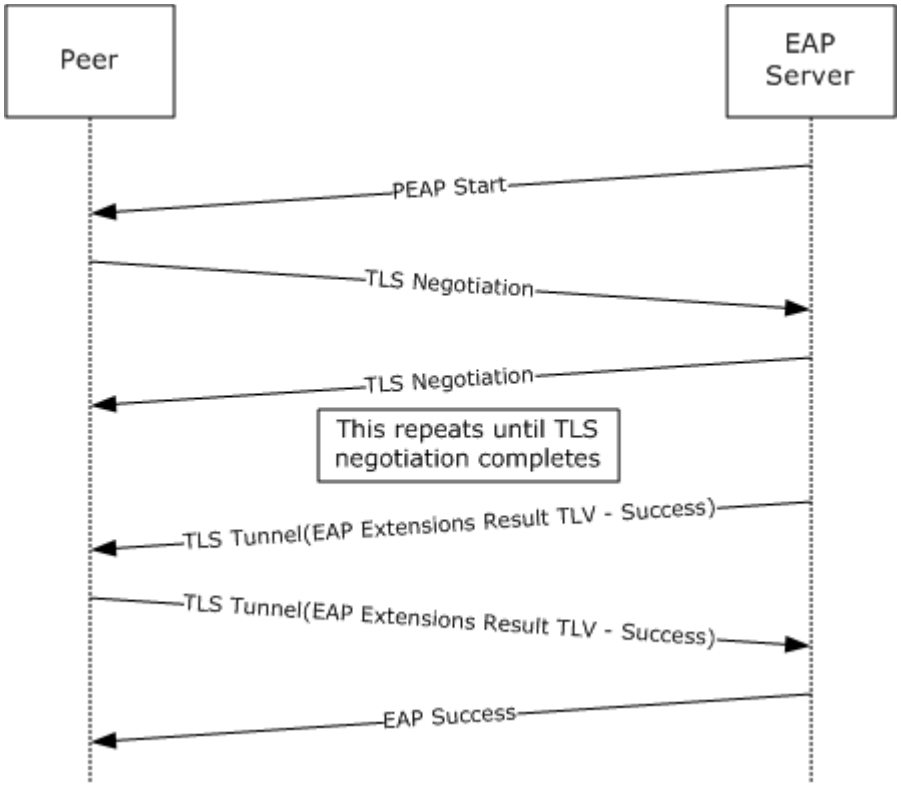


Figure 7: Successful CEAP TLS Tunnel Establishment with fast reconnect

4.2 Cryptobinding and SoH Processing Supported on CEAP Server Only

This section provides examples of CEAP interactions when cryptobinding and SoH processing [MS-SOH] are supported by a CEAP server implementation.

4.2.1 Successful CEAP TLS Tunnel Establishment and 2 Negotiation

This is similar to the example in section 4.1.1, except that, after phase 1, a Capabilities request and a SoH request are sent by the CEAP server and the peer responds with a NAK for both the requests. The peer also ignores the cryptobinding TLV from the CEAP server.

The following figure shows the CEAP server implementation not enforcing cryptobinding; if it did, the last message would be an EAP-Failure instead of EAP-Success.

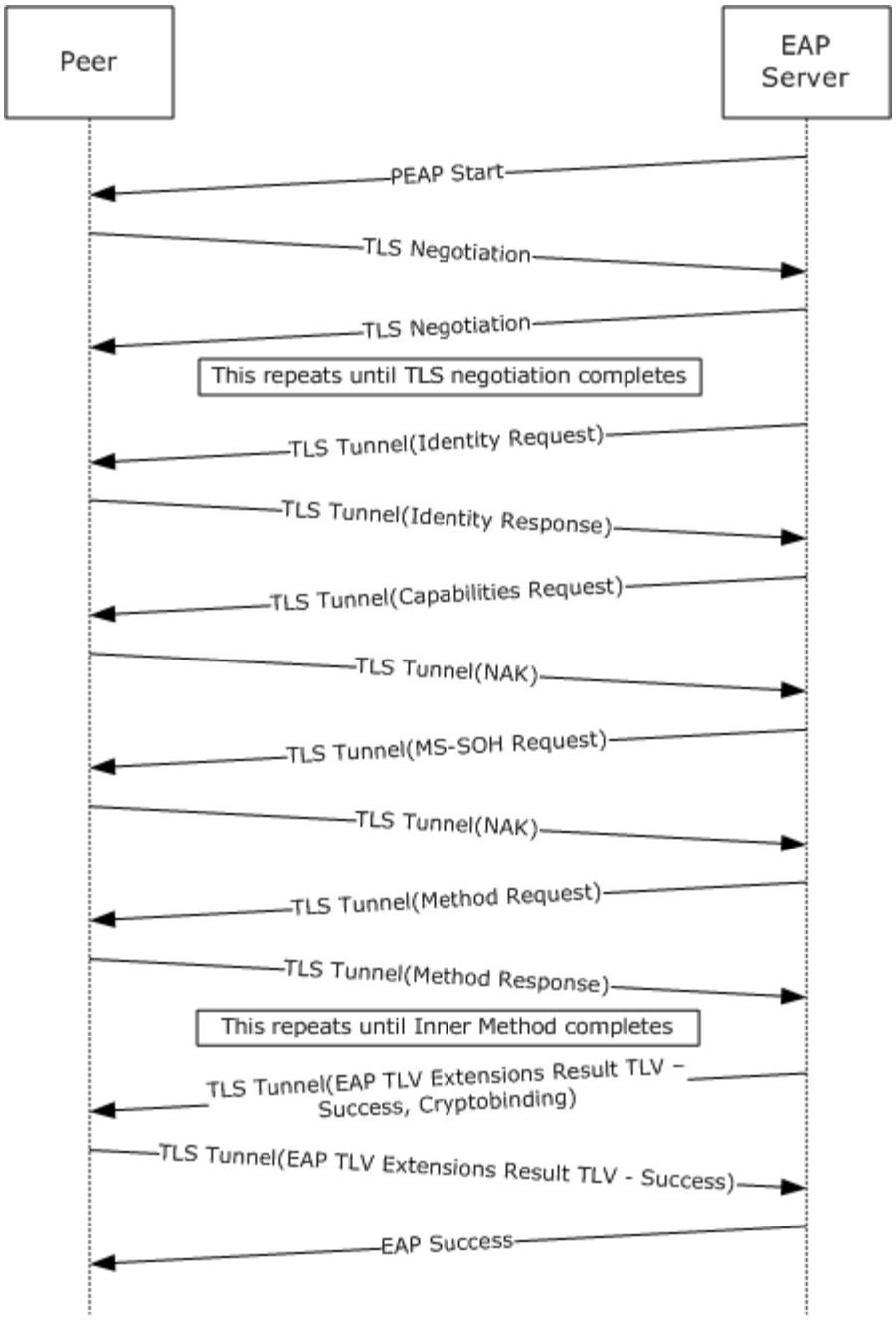


Figure 8: Successful CEAP TLS Tunnel Establishment and 2 negotiation

4.3 Cryptobinding and SoH Processing on CEAP Server and CEAP Peer

This section provides examples of CEAP interactions when cryptobinding and SoH processing are supported by a CEAP peer implementation, as well as a CEAP server implementation.

In the following example, cryptobinding and SoH processing is enforced on both the peer and CEAP server implementations.

4.3.1 Successful CEAP TLS Tunnel Establishment and 2 Negotiation

This is similar to the example in section 4.1.1, except that after phase 1, an SoH request is sent by the CEAP server and is positively acknowledged by the peer, which sends an SoH TLV (section 2.2.4.1.1.2). The peer also responds to the server's cryptobinding TLV by sending its own cryptobinding TLV.

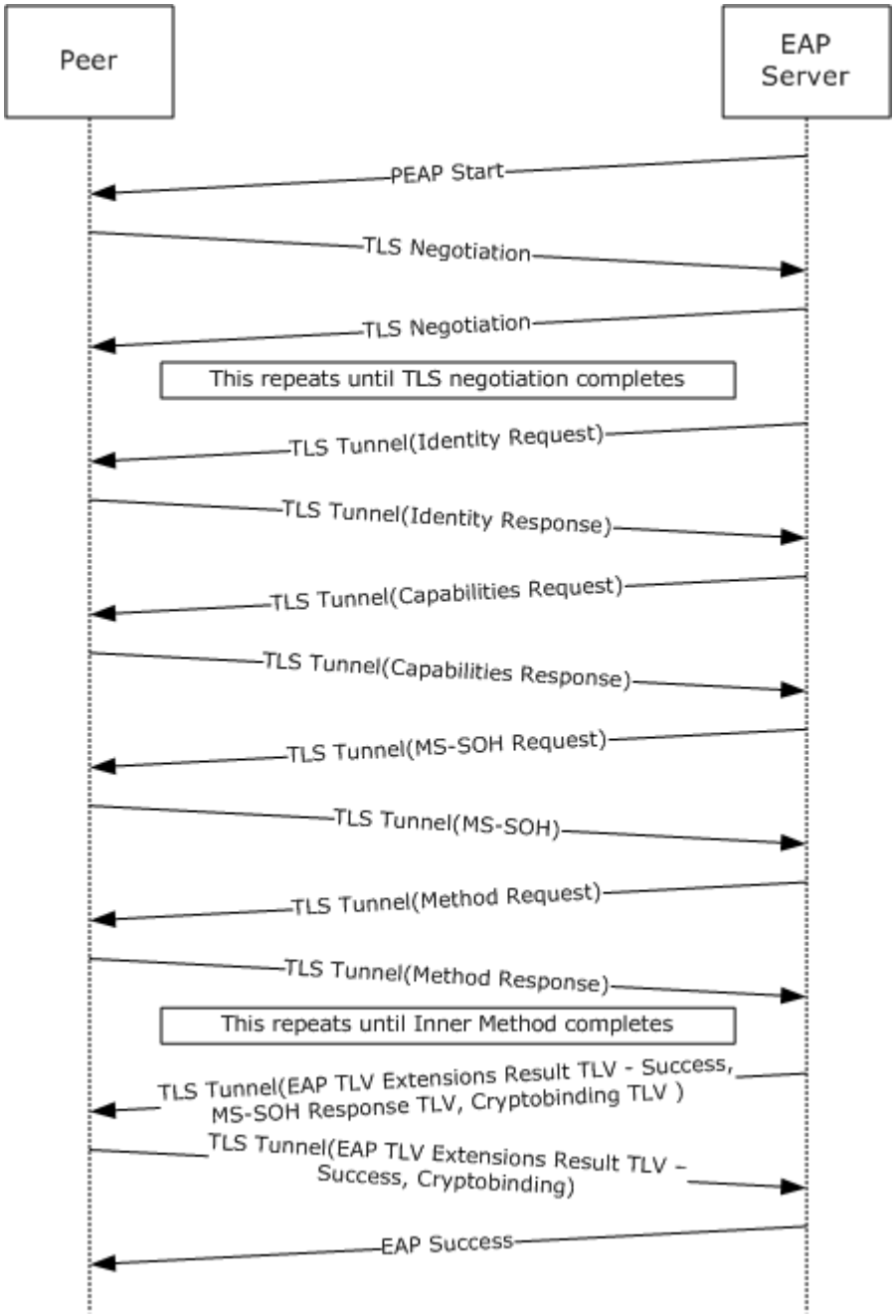


Figure 9: Successful CEAP TLS Tunnel Establishment and 2 negotiation

4.3.2 Successful CEAP TLS Tunnel Establishment with Fast Reconnect

The following diagram depicts a complete and successful CEAP authentication in which fast reconnect was used. Note that with fast reconnect, no CEAP authenticationor capabilities negotiation takes place.

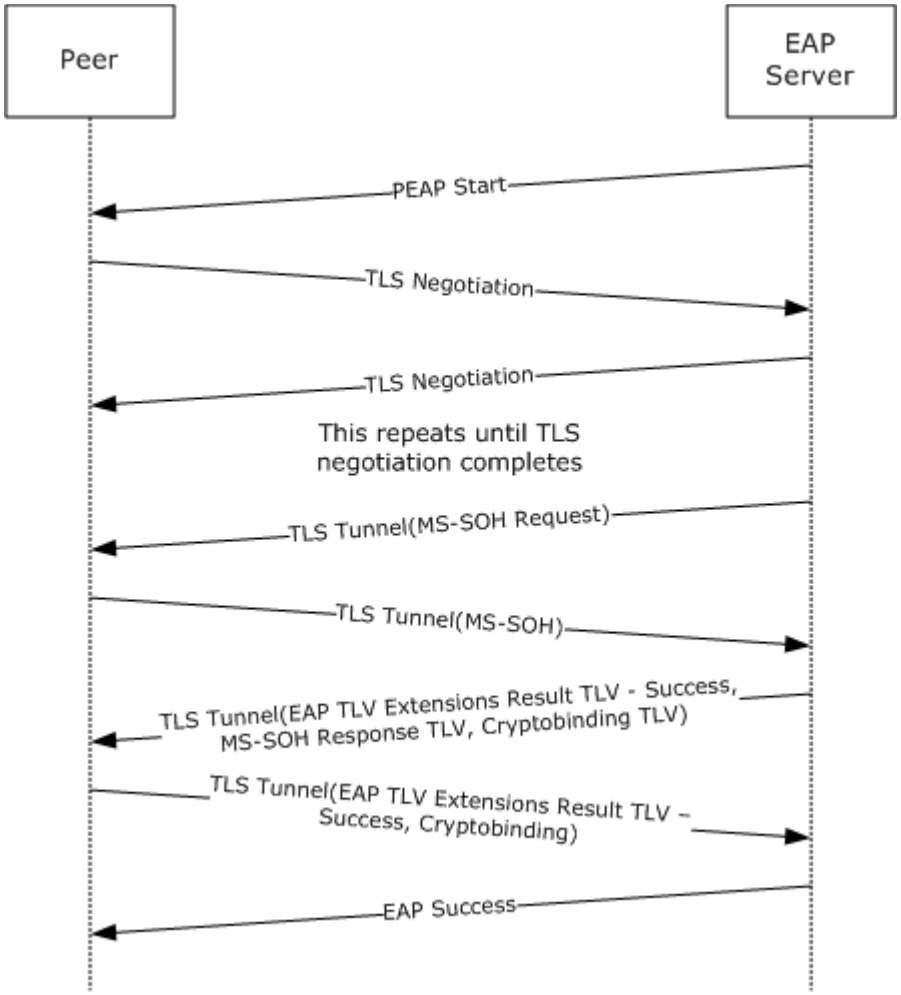


Figure 10: Successful CEAP TLS Tunnel Establishment with fast reconnect

4.3.3 Fallback to Full Authentication upon a Fast Reconnect Failure

The following diagram depicts a complete and successful CEAP authentication in which fast reconnect was attempted but failed (because, for example, fast reconnect was disabled on the peer). After the initial exchange of SoH packets, the peer indicated a failure, forcing full authentication, as in section 4.3.1.

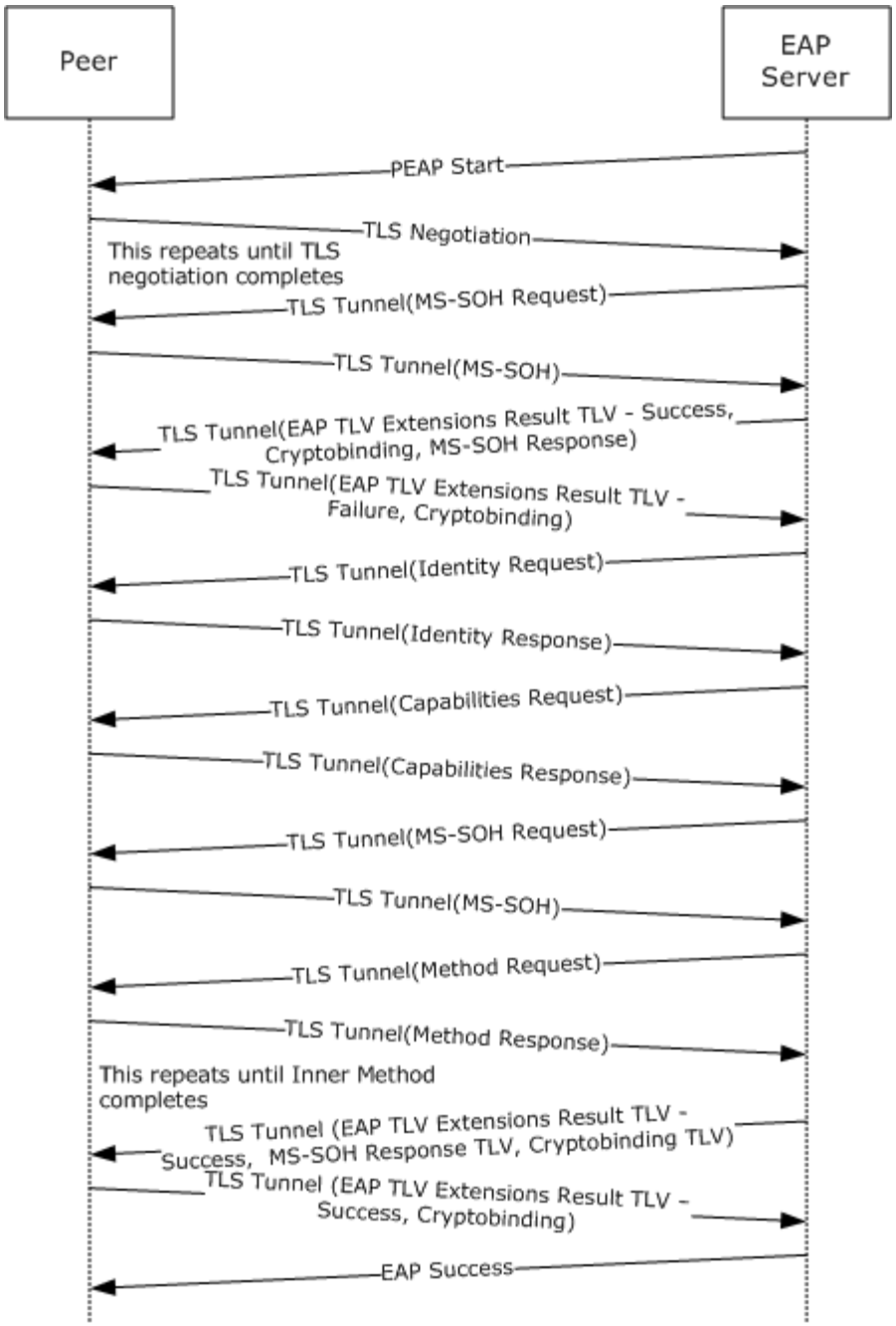


Figure 11: Fallback to full authentication upon a fast reconnect failure

4.4 Sample Cryptobinding TLV Data

The format of the Cryptobinding TLV packet is shown in section 2.2.4.2.1.

4.4.1 Cryptobinding TLV Request from Server to Client

4.4.1.1 Header

As per the description given in section 2.2.4.2.1, the first 8 octets of the cryptobinding TLV header appear as below:

```
00  0C  00 38  00  00  00  00
```

4.4.1.2 Nonce

The next field in the TLV is **nonce**, which is a 32 octet field generated by a random function. In our case let us assume that the following nonce is generated on server machine.

```
BD  A7  A5  99  FA  81  65  21  AD  30  64  C2  BD  DB  D1  6E
AA  94  9E  7D  98  A8  D7  94  31  47  CF   42  5D  85  DA  7B
```

4.4.1.3 Compound MAC

The 20 octet Compound MAC is generated as described in section 3.1.5.5. This field is generated from an HMAC-SHA1-160 operation. This operation requires two fields: data and key.

4.4.1.3.1 Data for HMAC-SHA1-160 Operation

The data required for HMAC-SHA1-160 operation is generated as per section 3.1.5.5.1. The generated data is as below:

```
00   0C 00  38  00   00    00   00  BD  A7  A5  99  FA  81  65  21
AD 30  64  C2 BD DB  D1  6E  AA  94  9E  7D  98  A8  D7  94
31   47  CF 42 5D   85  DA  7B  00   00   00   00  00  00   00  00
00   00  00  00 00   00   00    00  00   00   00   00  19
```

4.4.1.3.2 Key for HMAC-SHA1-160 Operation

The key required for HMAC-SHA1-160 operation is called the Compound MAC Key (CMK) and is generated by the formulae described in section 3.1.5.5.2. Inputs required for this operation are the TempKey(K) and IPMK Seed(S).

4.4.1.3.2.1 Temp Key

The most significant 40 octets of the Tunnel Key (TK) are considered as Temp Key (K). The TK is a 64-octet key generated in CEAP phase 1. Let us assume that the following TK is generated in the CEAP phase 1:

```
73  8B  B5  F4  62  D5  8E  7E  D8  44  E1  F0  0D  0E  BE  50
C5  0A  20  50  DE  11  99  77  10  D6  5F  45  FB  5F  BA  B7
E3  18  1E  92  4F  42  97  38  DE  40  C8  46  CD  F5  0B  CB
F9  CE  DB  1E  85  1D  22  52  45  3B  DF  63
```

Only the most significant 40 octets of the above data are relevant here.

4.4.1.3.2.2 IPMK Seed

IPMK seed is defined as follows:

```
IPMK Seed = "Inner Methods Compound Keys" | ISK
```

The ASCII representation of the string "Inner Methods Compound Keys" is (in hex):

```
49  6E  6E  65  72  20  4D  65  74  68  6F  64  73  20  43  6F
6D  70  6F  75  6E  64  20  4B  65  79  73
```

ISK is the Inner Session Key which would be obtained from the Inner method MPPE keys as described in section 3.1.5.5.2.2. Let us say that the generated ISK is as below:

```
67  3E  96  14  01  BE  FB  A5  60  71  7B  3B  5D  DD  40  38
65  67  F9  F4  16  FD  3E  9D  FC  71  16  3B  DF  F2  FA  95
```

4.4.1.3.2.3 IPMK and CMK

The PRF+ function generates 60 octet output out of which the most significant 40 octets denote the IPMK and the rest (20 octet) denote the CMK. With all the required information as described above for PRF+ function the computed T1, T2 and T3 appear as follows:

```
T1 = 3A 91 1C 25 54 73 E8 3E 9A 0C C3 33 AE 1F 8A 35 CD  C7  41  63
T2  =  E7  F6  0F  6C  65  EF  71  C2  64  42  AA  AC  A2  B6  F1  EB 4F  25  EC  A3
T3   =   33  55  35  3B  69  20  D0  74  C7  82  E4  75  DF  B0  99  9D 4D  B4  67  EB
IPMK = T1 | T2
CMK = T3
```

The generated CMK and the HMAC data are passed through the HMAC-SHA1-160 operation to generate the Compound MAC. The Compound MAC obtained from HMAC-SHA1-160 operation is as follows:

```
0C  BF  10  5E   91  75  57  48  22  4F  BB  83  00  06  26   91  1C  FB   1B  0F
```

After all the above computations the Cryptobinding TLV request from server appears as follows:

```
00  0C  00  38  00  00  00  00  BD  A7  A5  99  FA  81  65  21
AD 30  64 C2  BD  DB D1 6E  AA  94  9E  7D 98  A8  D7  94
31  47  CF  42  5D  85  DA  7B 0C  BF  10  5E   91  75  57  48
22  4F  BB  83  00  06  26   91  1C  FB  1B  0F
```

4.4.2 Cryptobinding TLV Response from Client to Server

4.4.2.1 Header

As per the description given in section 2.2.4.2.1, the first 8 octets of the cryptobinding TLV header appear as below:

```
00  0C  00 38  00  00  00  01
```

4.4.2.2 Nonce

The next field in the TLV is **nonce**, which is a 32 octet field generated by a random function. In our case let us assume that the following nonce is generated on client machine.

```
6C  6B  A3  87  84  23  74  57  CC  C9  0B  1A  90  8C  BD  F4
71  1B  69  99  4D  0C  FE  8D  3D  B4  4E  CB  CD  AD  37  E9
```

4.4.2.3 Compound MAC

The 20 octet Compound MAC is generated as described in section 3.1.5.5.1. This field is generated from an HMAC-SHA1-160 operation. This operation requires two fields: data and key.

4.4.2.3.1 Data for HMAC-SHA1-160 Operation

The data required for HMAC-SHA1-160 operation is generated as per section 3.1.5.5.1. The generated data is as below:

```
00  0C  00 38   00  00   00  01  6C  6B  A3  87  84  23  74  57
CC  C9  0B  1A  90  8C  BD  F4  71  1B  69  99  4D  0C  FE  8D
3D  B4  4E  CB  CD  AD  37  E9  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00   00   00  00  00  00  19
```

4.4.2.3.2 Key for HMAC-SHA1-160 Operation

The key required for HMAC-SHA1-160 operation is called the Compound MAC Key (CMK) and is generated by the formulae described in section 3.1.5.5.2. Inputs required for this operation are the TempKey(K) and IPMK Seed(S).

4.4.2.3.2.1 Temp Key

Because the Tunnel Key is same for both client and server, the TempKey remains the same as well.

4.4.2.3.2.2 IPMK Seed

Because the ISK for both client and server are same, the IPMK seed remains the same as well.

4.4.2.3.2.3 IPMK and CMK

Because all the inputs to PRF+ function are same, it generates the same IPMK and CMK as the server. The generated CMK and the HMAC data are passed through the HMAC-SHA1-160 operation to generate the Compound MAC.

The Compound MAC obtained from HMAC-SHA1-160 operation is as follows:

```
42  E0  86  07  1D  1C  8B  8C  8E  45  8F  70  21  F0  6A  6E  AB  16  B6  46
```

After all the above computations the Cryptobinding TLV response from client appears as follows:

```
00  0C  00 38  00  00  00  01  6C  6B  A3  87  84  23  74  57
CC  C9  0B 1A  90 8C  BD  F4  71  1B  69  99  4D  0C  FE  8D
3D  B4  4E CB  CD AD 37  E9  42  E0  86  07  1D  1C  8B  8C
8E   45  8F  70  21  F0 6A  6E AB  16  B6  46
```

4.4.3 MPPE Keys Generation

The MPPE keys generation is performed as per section 3.1.5.7. It requires both the IPMK and seed (S) as inputs. The IPMK generated by both client and server are as follows:

```
3A 91 1C 25 54 73 E8 3E 9A 0C C3 33 AE 1F 8A 35 CD C7 41 63 E7 F6 0F 6C 65 EF 71 C2 64 42 AA
AC A2 B6 F1 EB 4F 25 EC A3
```

Seed is the ASCII encoding of the string "Session Key Generating Function" appended with byte 0x00:

```
Seed = 53 65 73 73 69 6F 6E 20 4B 65 79 20 47 65 6E 65 72 61 74 69 6E 67 20 46 75 6E 63 74 69
6F 6E 00
```

Because the length of the keys is 128 octets, it requires 7 iterations of PRF+ function to generate 128 octets of data. The data after each iteration is as follows:

```
T1 =  6A   02   D7   82   20   1B   C7   13   8B   F8   EF   F7   33   B4   96   97   0D   7C
AB   30
T2 =  0A   C9   57   72   78   E1   DD   D5   AE   F7   66   97   17   52   D4   E5   84   A1
C8   95
T3 =  03   9B   4D   05   E3   BC   9A   84   84   DD   C2   AA   6E   2C   E1   62   76   5C
40   68
T4 =  BF   F6   5A   45   10   E3   05   74   85   DB   98   B7   99   D8   6E   66   76   3C
64   D4
T5 =  98   89   B4   DD   1B   27   3D   C8   A2   CA   73   D6   0D   11   AF   B2   2C   52
BA   AD
T6 =  D3   51   E0   CB   7B   B2   E7   2C   7D   93   73   85   7E   03   C1   4A   32   C8
F7   E5
T7 =  95   9F   46   68   0E   86   E6   5C   89   F8   80   C8   A6   DA   00   56   3A   FB
19   C0
```

Based on the above data, the keys on the server side are as follows:

```
RecvKey =  6A 02 D7 82 20 1B C7 13 8B F8 EF F7 33 B4 96 97 0D 7C AB 30 0A C9 57 72 78 E1 DD
D5 AE F7 66 97
SendKey = 17 52 D4 E5 84 A1 C8 95 03 9B 4D 05 E3 BC 9A 84 84 DD C2 AA 6E 2C E1 62 76 5C 40 68
BF F6 5A 45
Client RecvKey = server SendKey
Client SendKey = server RecvKey
```

Only the most significant 64 octets are used though we generate 128 octets. The least significant 64 octets are reserved for future use.

4.4.4 Example NAP Health Evaluation Scenario using CEAP / EAP

The NAP health policy server and NAP enforcement points exchange system health information via the CEAP SoH and SoHR TLVs and restricted access instructions with the Remote Authentication Dial-In User service (RADIUS) server and proxy messages.

- The NPS service compares the incoming request message to its configured set of connection request policies. For NAP, the request message should match a connection request policy that specifies that the NPS service perform the authentication and authorization locally.
- The connection request policy can also impose connection requirements. For example, for 802.1X and VPN enforcement, the connection request policy requires the use of a Protected Extensible Authentication Protocol (CEAP)-based authentication method. If the connecting client does not use CEAP, the connection request is rejected.
- The NPS service evaluates the health information in the request message, which consists of a system statement of health (SSoH) containing health status information. The NPS service passes the health status information to its installed SHVs, which return health evaluation information to the NPS service.
- The NPS service compares the request message and the health evaluation information from the SHVs to the appropriate set of network policies. The health evaluation information is compared to the Health Policy condition of NAP-based network policies. The Health Policy condition specifies the conditions for health compliance or noncompliance. The NPS service applies the first matching network policy to the request message.
- Based on the matching NAP-based network policy and its associated NAP settings, the NPS service creates a system statement of health response (SSoHR). This response includes the health evaluation information from the SHVs and indicates whether the NAP client has unlimited or restricted access and whether the NAP client should automatically attempt to remediate its health state.
- The NPS service sends a RADIUS response message with the SSoHR to the NAP enforcement point. If the client has been granted restricted access, the response message can also contain instructions that specify how the NAP client's access is restricted.
- The NAP enforcement point sends the SSoHR to the NAP client.

5 Security

The following sections specify security considerations for implementers of CEAP.

5.1 Security Considerations for Implementers

5.1.1 Fast Reconnect

CEAP fast reconnect is desirable in applications such as wireless roaming. This feature allows sessions to be resumed without completing a full authentication.

However, some issues that should be considered to avoid introducing security vulnerabilities include:

- In cases where no identity is proved with an CEAP method, implementers should ensure that the appropriate authorization checks are still performed for the session.
- To protect against risks associated with incorrectly assigning identity on fast reconnection scenarios, implementations should strongly tie identity information to the TLS session. That is, the CEAP implementation must determine the user identity even with a session resume. If it cannot do so, then it must not authorize access. The reason is that because no CEAP authentication takes place during fast reconnect; proof of identity is based exclusively on the TLS session.

5.1.2 Identity Verification

Because the TLS session has not yet been negotiated, the initial identity request/response occurs in the clear, without integrity protection or authentication. It is therefore vulnerable to snooping and packet modification.

If the initial EAP **cleartext** identity request/response has been tampered with, then, after the TLS session is established, it is conceivable that the CEAP server will discover that it cannot verify the peer's claim of identity. For example, the peer's user ID may not be valid or may not be within a **realm** handled by the CEAP server. In a case where the CEAP server is unable to validate the peer's identity claims, the CEAP server must abort the authentication.

Moreover, it cannot be assumed that the peer identities presented within multiple EAP-Response/Identity packets will be the same. For example, the initial EAP-Response/Identity might correspond to a machine identity, while subsequent identities might be those of the user. Thus, CEAP implementations should not abort the authentication just because the identities do not match. However, because the initial EAP-Response/Identity determines the EAP server handling the authentication, if this or any other identity is inappropriate for use with the destination EAP server, there is no alternative but to terminate the CEAP conversation.

5.1.3 Authentication Outcomes

Because the cleartext EAP success or failure messages can be tampered with, implementations should rely only on the EAP Extensions method with Result TLV's status messages to determine the outcome of a session.

5.2 Index of Security Parameters

Security parameter	Section
Allowable EAP CEAP method configuration	Sections 3.2.3 and 3.3.3

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows NT® operating system
- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 3.1.1: The Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 CEAP implementations support Cryptobinding TLVs (section 2.2.4.2.1). Other versions of Windows do not support Cryptobinding TLVs.

<2> Section 3.1.1: The ADM element is initialized with the value configured at the registry value HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\BypassNegotiation. It is only supported on Windows 7 and Windows Server 2008 R2.

<3> Section 3.1.1: The ADM element is initialized with the value configured at the registry value HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\AssumePhase2Fragmentation. It is only supported on Windows 7 and Windows Server 2008 R2.

<4> Section 3.1.1: The Windows 7 and Windows Server 2008 R2 CEAP implementations support Capabilities Negotiation Method (section 2.2.4.1.1.3) packets. Other versions of Windows do not support these packets - in these cases, the peer responds with an EAP NAK and the server never sends a Capabilities Negotiation Method packet.

<5> Section 3.1.1: The Windows 7 and Windows Server 2008 R2 CEAP implementations support CEAP Phase 2 packet fragmentation. The Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 CEAP implementations do not support CEAP Phase 2 packet fragmentation.

<6> Section 3.1.5.5: Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2 implement cryptobinding. Use of cryptobinding can be configured on both CEAP server and CEAP peer implementations.

Windows CEAP server implementations which support Cryptobinding always send cryptobinding TLVs. If a server implementation configured to enforce cryptobinding TLVs sends a cryptobinding TLV and does not receive one in response, it ends the conversation by sending an EAP-Failure. If the enforcement is not configured and the server does not receive a cryptobinding TLV, it is processed without cryptobinding support.

Windows CEAP peer implementations can be configured to enforce the exchange of a cryptobinding TLV. A peer receiving a cryptobinding TLV responds with a cryptobinding TLV irrespective of the configuration. If the peer is configured to expect a cryptobinding TLV and does not receive one, it ends the conversation by sending a Failure Result TLV (section 2.2.8.1.2). If the peer does not receive a cryptobinding TLV and is not configured to expect a cryptobinding TLV, the peer processes the packet without cryptobinding support.

<7> Section 3.2.1: Only supported on Windows 7, Windows Server 2008, and Windows Server 2008 R2 CEAP implementations.

<8> Section 3.2.1: Only supported on Windows 7, Windows Server 2008, and Windows Server 2008 R2 CEAP implementations.

<9> Section 3.2.3: Only supported on Windows 7 and Windows Server 2008 R2.

<10> Section 3.2.3: **BypassCapNegotiation** is initialized from "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\BypassNegotiation". **AssumePhase2Frag** is initialized from "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\AssumePhase2Fragmentation".

<11> Section 3.2.5.4.6: The Windows CEAP peer implementations never send the Capabilities Method Response (section 3.3.5.4.4) packet with the F flag set to zero.

<12> Section 3.2.7.1: Windows uses the certificates in the "machine trusted root CA store" to validate the trust anchor of the server certificate.

<13> Section 3.3.3: Only supported on Windows 7 and Windows Server 2008 R2.

<14> Section 3.3.3: **BypassCapNegotiation** is initialized from "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\BypassNegotiation". **AssumePhase2Frag** is initialized from "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\RasMan\PPP\EAP\25\AssumePhase2Fragmentation".

<15> Section 3.3.4.1: The Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 CEAP implementations support SoH [MS-SOH] TLV transmission and processing. Other versions of Windows do not support SoH TLV (section 2.2.4.1.1.2) transmission and processing.

<16> Section 3.3.5.4.3: The Windows CEAP server implementations never send a Capabilities Method Request (section 3.2.5.4.6) packet with the F flag set to zero.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index

L

M

N

O

P

R

S

T

V