

Network Working Group  
Internet-Draft  
Obsoletes: 3748 (if approved)  
Intended status: Standards Track  
Expires: August 26, 2021

B. Aboba  
Microsoft Corporation  
L. Blunk  
Merit Network, Inc  
J. Vollbrecht  
Vollbrecht Consulting LLC  
J. Carlson  
Sun Microsystems, Inc  
H. Levkowetz  
ipUnplugged AB  
J. Arkko (Ed.)  
J. Mattsson (Ed.)  
Ericsson  
February 22, 2021

Extensible Authentication Protocol (EAP)  
draft-arkko-emu-rfc3748bis-00

Abstract

This document defines the Extensible Authentication Protocol (EAP), an authentication framework which supports multiple authentication methods. EAP typically runs directly over data link layers such as Point-to-Point Protocol (PPP), IEEE 802, or 3GPP 5G without requiring IP. EAP provides its own support for duplicate elimination and retransmission, but is reliant on lower layer ordering guarantees. Fragmentation is not supported within EAP itself; however, individual EAP methods may support this.

This document obsoletes RFC 3748, which in turn obsoleted RFC 2284. This document updates some of the security considerations, terms, references, the IANA considerations, and few other minor updates. A summary of the changes between this document and RFC 3748 is in Appendix A, and the changes from RFC 2284 were listed in RFC 3748.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 26, 2021.

#### Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Specification of Requirements . . . . .	5
1.2. Terminology . . . . .	5
1.3. Applicability . . . . .	7
2. Extensible Authentication Protocol (EAP) . . . . .	8
2.1. Support for Sequences . . . . .	10
2.2. EAP Multiplexing Model . . . . .	11
2.3. Pass-Through Behavior . . . . .	13
2.4. Peer-to-Peer Operation . . . . .	15
3. Lower Layer Behavior . . . . .	16
3.1. Lower Layer Requirements . . . . .	16
3.2. EAP Usage Within PPP . . . . .	19
3.2.1. PPP Configuration Option Format . . . . .	19
3.3. EAP Usage Within IEEE 802 . . . . .	20
3.4. Lower Layer Indications . . . . .	20
4. EAP Packet Format . . . . .	21
4.1. Request and Response . . . . .	22
4.2. Success and Failure . . . . .	24
4.3. Retransmission Behavior . . . . .	27
5. Initial EAP Request/Response Types . . . . .	28
5.1. Identity . . . . .	29
5.2. Notification . . . . .	30
5.3. Nak . . . . .	32
5.3.1. Legacy Nak . . . . .	32
5.3.2. Expanded Nak . . . . .	33
5.4. MD5-Challenge . . . . .	36

5.5.	One-Time Password (OTP)	38
5.6.	Generic Token Card (GTC)	39
5.7.	Expanded Types	40
5.8.	Experimental	42
6.	IANA Considerations	42
6.1.	Packet Codes	43
6.2.	Method Types	43
7.	Security Considerations	43
7.1.	Threat Model	44
7.2.	Security Claims	45
7.2.1.	Security Claims Terminology for EAP Methods	46
7.3.	Identity Protection	48
7.4.	Man-in-the-Middle Attacks	49
7.5.	Packet Modification Attacks	50
7.6.	Dictionary Attacks	51
7.7.	Connection to an Untrusted Network	51
7.8.	Negotiation Attacks	52
7.9.	Implementation Idiosyncrasies	52
7.10.	Key Derivation	53
7.11.	Weak Ciphersuites	55
7.11.1.	Legacy Authentication Methods	55
7.12.	Link Layer	56
7.13.	Separation of Authenticator and Backend Authentication Server	56
7.14.	Cleartext Passwords	57
7.15.	Channel Binding	58
7.16.	Protected Result Indications	58
8.	References	61
8.1.	Normative References	61
8.2.	Informative References	62
Appendix A.	Changes from RFC 3748	68
Appendix B.	Rationale	69
Appendix C.	Acknowledgements	70
Authors' Addresses		71

## 1. Introduction

This document defines the Extensible Authentication Protocol (EAP), an authentication framework which supports multiple authentication methods. EAP typically runs directly over data link layers such as Point-to-Point Protocol (PPP), IEEE 802, or 3GPP 5G without requiring IP. EAP provides its own support for duplicate elimination and retransmission, but is reliant on lower layer ordering guarantees. Fragmentation is not supported within EAP itself; however, individual EAP methods may support this.

EAP may be used on dedicated links, as well as switched circuits, and wired as well as wireless links. To date, EAP has been implemented

with hosts and routers that connect via switched circuits or dial-up lines using PPP [RFC1661]. It has also been implemented with switches and access points using IEEE 802 [IEEE-802]. EAP encapsulation on IEEE 802 wired media is described in [IEEE-802.1X], and encapsulation on IEEE wireless LANs in [IEEE-802.11i]. EAP can be used for authentication in all types of accesses in 3GPP 5G [TS.33.501].

One of the advantages of the EAP architecture is its flexibility. EAP is used to select a specific authentication mechanism, typically after the authenticator requests more information in order to determine the specific authentication method to be used. Rather than requiring the authenticator to be updated to support each new authentication method, EAP permits the use of a backend authentication server, which may implement some or all authentication methods, with the authenticator acting as a pass-through for some or all methods and peers.

Within this document, authenticator requirements apply regardless of whether the authenticator is operating as a pass-through or not. Where the requirement is meant to apply to either the authenticator or backend authentication server, depending on where the EAP authentication is terminated, the term "EAP server" will be used.

Other aspects of the EAP framework are discussed in companion documents, [RFC4137] discusses a possible state machine, [RFC5113] defines the network discovery and selection problem, [RFC5247] specifies the EAP key hierarchy, [RFC6677] and [RFC7029] explores man-in-the-middle attacks as well as defining how to implement channel bindings.

While the authors believe that the update from RFC 3748 is useful, it is by no means something that absolute has to be done, but has been provided for the community's consideration as part of an overall interest in maintaining the technology and its documentation. If we care about a technology we should keep it up to date. The authors believe that it is preferable to have ongoing maintenance that addresses issues when they are identified, rather than waiting for a larger but more infrequent update. The specific changes are discussed in Appendix A, and the rationale for the terminology-related parts of the change is discussed in more detail in Appendix B.

This update proposal is brought forward for discussion. Discussion may find that the update is considered useful or unnecessary, or perhaps even a distraction or flawed in some of its definitions. All feedback is welcome!

## 1.1. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 1.2. Terminology

This document frequently uses the following terms:

### authenticator

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

### peer

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

### Supplicant

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

### backend authentication server

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

### AAA

Authentication, Authorization, and Accounting. AAA protocols with EAP support include RADIUS [RFC3579] and Diameter [RFC4072]. In this document, the terms "AAA server" and "backend authentication server" are used interchangeably.

### Displayable Message

This is interpreted to be a human readable string of characters. The message encoding MUST follow the UTF-8 transformation format [RFC3629].

### EAP server

The entity that terminates the EAP authentication method with the peer. In the case where no backend authentication server is used, the EAP server is part of the authenticator. In the case where the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.

### Silently Discard

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

### Successful Authentication

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

### Message Integrity Check (MIC)

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

### Cryptographic Separation

Two keys (x and y) are "cryptographically separate" if an adversary that knows all messages exchanged in the protocol cannot compute x from y or y from x without "breaking" some cryptographic assumption. In particular, this definition allows that the adversary has the knowledge of all nonces sent in cleartext, as well as all predictable counter values used in the protocol. Breaking a cryptographic assumption would typically require inverting a one-way function or predicting the outcome of a cryptographic pseudo-random number generator without knowledge of the secret state. In other words, if the keys are cryptographically separate, there is no shortcut to compute x from y or y from x, but the work an adversary must do to perform this

computation is equivalent to performing an exhaustive search for the secret state value.

#### Main Session Key (MSK)

Keying material that is derived between the EAP peer and server and exported by the EAP method. The MSK is at least 64 octets in length. In existing implementations, a AAA server acting as an EAP server transports the MSK to the authenticator.

#### Extended Main Session Key (EMSK)

Additional keying material derived between the EAP client and server that is exported by the EAP method. The EMSK is at least 64 octets in length. The EMSK is not shared with the authenticator or any other third party. The EMSK is reserved for future uses that are not defined yet.

#### Result indications

A method provides result indications if after the method's last message is sent and received:

1. The peer is aware of whether it has authenticated the server, as well as whether the server has authenticated it.
2. The server is aware of whether it has authenticated the peer, as well as whether the peer has authenticated it.

In the case where successful authentication is sufficient to authorize access, then the peer and authenticator will also know if the other party is willing to provide or accept access. This may not always be the case. An authenticated peer may be denied access due to lack of authorization (e.g., session limit) or other reasons. Since the EAP exchange is run between the peer and the server, other nodes (such as AAA proxies) may also affect the authorization decision. This is discussed in more detail in Section 7.16.

### 1.3. Applicability

EAP was designed for use in network access authentication, where IP layer connectivity may not be available. Use of EAP for other purposes, such as bulk data transport, is NOT RECOMMENDED.

Since EAP does not require IP connectivity, it provides just enough support for the reliable transport of authentication protocols, and no more.

EAP is a lock-step protocol which only supports a single packet in flight. As a result, EAP cannot efficiently transport bulk data, unlike transport protocols such as TCP [RFC0793] or SCTP [RFC4960].

While EAP provides support for retransmission, it assumes ordering guarantees provided by the lower layer, so out of order reception is not supported.

Since EAP does not support fragmentation and reassembly, EAP authentication methods generating payloads larger than the minimum EAP MTU need to provide fragmentation support.

While authentication methods such as EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13] provide support for fragmentation and reassembly, the EAP methods defined in this document do not. As a result, if the EAP packet size exceeds the EAP MTU of the link, these methods will encounter difficulties.

EAP authentication is initiated by the server (authenticator), whereas many authentication protocols are initiated by the client (peer). As a result, it may be necessary for an authentication algorithm to add one or two additional messages (at most one roundtrip) in order to run over EAP.

Where certificate-based authentication is supported, the number of additional roundtrips may be much larger due to fragmentation of certificate chains. In general, a fragmented EAP packet will require as many round-trips to send as there are fragments. For example, a certificate chain 14960 octets in size would require ten round-trips to send with a 1496 octet EAP MTU.

Where EAP runs over a lower layer in which significant packet loss is experienced, or where the connection between the authenticator and authentication server experiences significant packet loss, EAP methods requiring many round-trips can experience difficulties. In these situations, use of EAP methods with fewer roundtrips is advisable.

## 2. Extensible Authentication Protocol (EAP)

The EAP authentication exchange proceeds as follows:

1. The authenticator sends a Request to authenticate the peer. The Request has a Type field to indicate what is being requested. Examples of Request Types include Identity, MD5-challenge, etc. The MD5-challenge Type corresponds closely to the CHAP authentication protocol [RFC1994]. Typically, the authenticator will send an initial Identity Request; however, an initial



Identity Request is not required, and MAY be bypassed. For example, the identity may not be required where it is determined by the port to which the peer has connected (leased lines, dedicated switch or dial-up ports), or where the identity is obtained in another fashion (via calling station identity or MAC address, in the Name field of the MD5-Challenge Response, etc.).

2. The peer sends a Response packet in reply to a valid Request. As with the Request packet, the Response packet contains a Type field, which corresponds to the Type field of the Request.
3. The authenticator sends an additional Request packet, and the peer replies with a Response. The sequence of Requests and Responses continues as long as needed. EAP is a 'lock step' protocol, so that other than the initial Request, a new Request cannot be sent prior to receiving a valid Response. The authenticator is responsible for retransmitting requests as described in Section 4.1. After a suitable number of retransmissions, the authenticator SHOULD end the EAP conversation. The authenticator MUST NOT send a Success or Failure packet when retransmitting or when it fails to get a response from the peer.
4. The conversation continues until the authenticator cannot authenticate the peer (unacceptable Responses to one or more Requests), in which case the authenticator implementation MUST transmit an EAP Failure (Code 4). Alternatively, the authentication conversation can continue until the authenticator determines that successful authentication has occurred, in which case the authenticator MUST transmit an EAP Success (Code 3).

Advantages:

- o The EAP protocol can support multiple authentication mechanisms without having to pre-negotiate a particular one.
- o Network Access Server (NAS) devices (e.g., a switch or access point) do not have to understand each authentication method and MAY act as a pass-through agent for a backend authentication server. Support for pass-through is optional. An authenticator MAY authenticate local peers, while at the same time acting as a pass-through for non-local peers and authentication methods it does not implement locally.
- o Separation of the authenticator from the backend authentication server simplifies credentials management and policy decision making.

#### Disadvantages:

- o For use in PPP, EAP requires the addition of a new authentication Type to PPP LCP and thus PPP implementations will need to be modified to use it. It also strays from the previous PPP authentication model of negotiating a specific authentication mechanism during LCP. Similarly, switch or access point implementations need to support [IEEE-802.1X] in order to use EAP.
- o Where the authenticator is separate from the backend authentication server, this complicates the security analysis and, if needed, key distribution.

#### 2.1. Support for Sequences

An EAP conversation MAY utilize a sequence of methods. A common example of this is an Identity request followed by a single EAP authentication method such as an MD5-Challenge. However, the peer and authenticator MUST utilize only one authentication method (Type 4 or greater) within an EAP conversation, after which the authenticator MUST send a Success or Failure packet.

Once a peer has sent a Response of the same Type as the initial Request, an authenticator MUST NOT send a Request of a different Type prior to completion of the final round of a given method (with the exception of a Notification-Request) and MUST NOT send a Request for an additional method of any Type after completion of the initial authentication method; a peer receiving such Requests MUST treat them as invalid, and silently discard them. As a result, Identity Requery is not supported.

A peer MUST NOT send a Nak (legacy or expanded) in reply to a Request after an initial non-Nak Response has been sent. Since spoofed EAP Request packets may be sent by an attacker, an authenticator receiving an unexpected Nak SHOULD discard it and log the event.

Multiple authentication methods within an EAP conversation are not supported due to their vulnerability to man-in-the-middle attacks (see Section 7.4) and incompatibility with existing implementations.

Where a single EAP authentication method is utilized, but other methods are run within it (a "tunneled" method), the prohibition against multiple authentication methods does not apply. Such "tunneled" methods appear as a single authentication method to EAP. Backward compatibility can be provided, since a peer not supporting a "tunneled" method can reply to the initial EAP-Request with a Nak (legacy or expanded). To address security vulnerabilities,

"tunneled" methods MUST support protection against man-in-the-middle attacks.

## 2.2. EAP Multiplexing Model

Conceptually, EAP implementations consist of the following components:

1. Lower layer. The lower layer is responsible for transmitting and receiving EAP frames between the peer and authenticator. EAP has been run over a variety of lower layers including PPP, wired IEEE 802 LANs [IEEE-802.1X], IEEE 802.11 wireless LANs [IEEE-802.11], UDP (L2TP [RFC2661] and IKEv2 [RFC7296]), TCP [I-D.ietf-ipsra-pic], and 3GPP 5G [TS.33.501]. Lower layer behavior is discussed in Section 3.
2. EAP layer. The EAP layer receives and transmits EAP packets via the lower layer, implements duplicate detection and retransmission, and delivers and receives EAP messages to and from the EAP peer and authenticator layers.
3. EAP peer and authenticator layers. Based on the Code field, the EAP layer demultiplexes incoming EAP packets to the EAP peer and authenticator layers. Typically, an EAP implementation on a given host will support either peer or authenticator functionality, but it is possible for a host to act as both an EAP peer and authenticator. In such an implementation both EAP peer and authenticator layers will be present.
4. EAP method layers. EAP methods implement the authentication algorithms and receive and transmit EAP messages via the EAP peer and authenticator layers. Since fragmentation support is not provided by EAP itself, this is the responsibility of EAP methods, which are discussed in Section 5.

The EAP multiplexing model is illustrated in Figure 1 below. Note that there is no requirement that an implementation conform to this model, as long as the on-the-wire behavior is consistent with it.

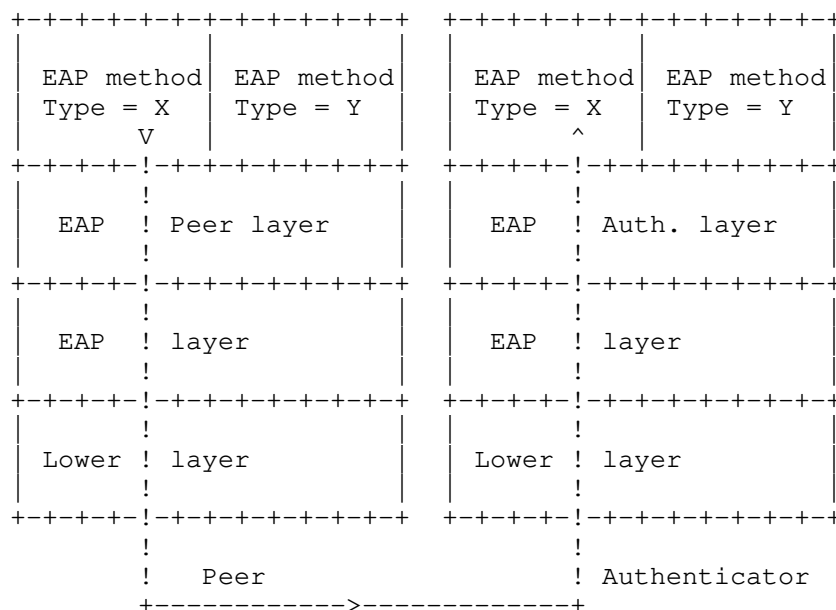


Figure 1: EAP Multiplexing Model

Within EAP, the Code field functions much like a protocol number in IP. It is assumed that the EAP layer demultiplexes incoming EAP packets according to the Code field. Received EAP packets with Code=1 (Request), 3 (Success), and 4 (Failure) are delivered by the EAP layer to the EAP peer layer, if implemented. EAP packets with Code=2 (Response) are delivered to the EAP authenticator layer, if implemented.

Within EAP, the Type field functions much like a port number in UDP or TCP. It is assumed that the EAP peer and authenticator layers demultiplex incoming EAP packets according to their Type, and deliver them only to the EAP method corresponding to that Type. An EAP method implementation on a host may register to receive packets from the peer or authenticator layers, or both, depending on which role(s) it supports.

Since EAP authentication methods may wish to access the Identity, implementations SHOULD make the Identity Request and Response accessible to authentication methods (Types 4 or greater), in addition to the Identity method. The Identity Type is discussed in Section 5.1.

A Notification Response is only used as confirmation that the peer received the Notification Request, not that it has processed it, or

displayed the message to the user. It cannot be assumed that the contents of the Notification Request or Response are available to another method. The Notification Type is discussed in Section 5.2.

Nak (Type 3) or Expanded Nak (Type 254) are utilized for the purposes of method negotiation. Peers respond to an initial EAP Request for an unacceptable Type with a Nak Response (Type 3) or Expanded Nak Response (Type 254). It cannot be assumed that the contents of the Nak Response(s) are available to another method. The Nak Type(s) are discussed in Section 5.3.

EAP packets with Codes of Success or Failure do not include a Type field, and are not delivered to an EAP method. Success and Failure are discussed in Section 4.2.

Given these considerations, the Success, Failure, Nak Response(s), and Notification Request/Response messages MUST NOT be used to carry data destined for delivery to other EAP methods.

### 2.3. Pass-Through Behavior

When operating as a "pass-through authenticator", an authenticator performs checks on the Code, Identifier, and Length fields as described in Section 4.1. It forwards EAP packets received from the peer and destined to its authenticator layer to the backend authentication server; packets received from the backend authentication server destined to the peer are forwarded to it.

A host receiving an EAP packet may only do one of three things with it: act on it, drop it, or forward it. The forwarding decision is typically based only on examination of the Code, Identifier, and Length fields. A pass-through authenticator implementation MUST be capable of forwarding EAP packets received from the peer with Code=2 (Response) to the backend authentication server. It also MUST be capable of receiving EAP packets from the backend authentication server and forwarding EAP packets of Code=1 (Request), Code=3 (Success), and Code=4 (Failure) to the peer.

Unless the authenticator implements one or more authentication methods locally which support the authenticator role, the EAP method layer header fields (Type, Type-Data) are not examined as part of the forwarding decision. Where the authenticator supports local authentication methods, it MAY examine the Type field to determine whether to act on the packet itself or forward it. Compliant pass-through authenticator implementations MUST by default forward EAP packets of any Type.

EAP packets received with Code=1 (Request), Code=3 (Success), and Code=4 (Failure) are demultiplexed by the EAP layer and delivered to the peer layer. Therefore, unless a host implements an EAP peer layer, these packets will be silently discarded. Similarly, EAP packets received with Code=2 (Response) are demultiplexed by the EAP layer and delivered to the authenticator layer. Therefore, unless a host implements an EAP authenticator layer, these packets will be silently discarded. The behavior of a "pass-through peer" is undefined within this specification, and is unsupported by AAA protocols such as RADIUS [RFC3579] and Diameter [RFC4072].

The forwarding model is illustrated in Figure 2.

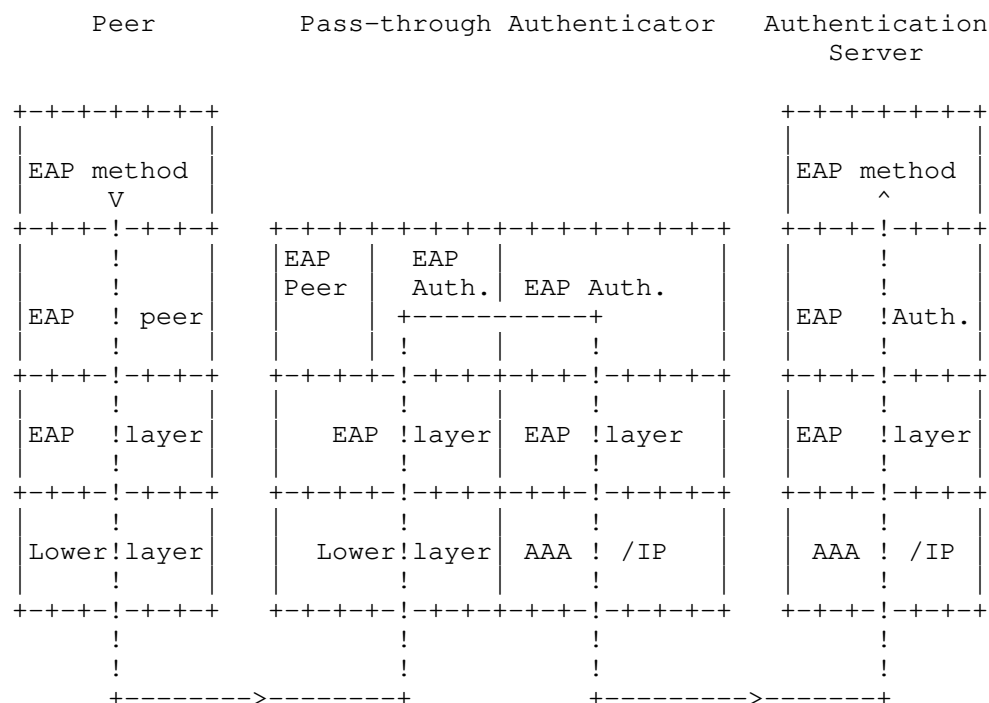


Figure 2: Pass-through Authenticator

For sessions in which the authenticator acts as a pass-through, it MUST determine the outcome of the authentication solely based on the Accept/Reject indication sent by the backend authentication server; the outcome MUST NOT be determined by the contents of an EAP packet sent along with the Accept/Reject indication, or the absence of such an encapsulated EAP packet.

## 2.4. Peer-to-Peer Operation

Since EAP is a peer-to-peer protocol, an independent and simultaneous authentication may take place in the reverse direction (depending on the capabilities of the lower layer). Both ends of the link may act as authenticators and peers at the same time. In this case, it is necessary for both ends to implement EAP authenticator and peer layers. In addition, the EAP method implementations on both peers must support both authenticator and peer functionality.

Although EAP supports peer-to-peer operation, some EAP implementations, methods, AAA protocols, and link layers may not support this. Some EAP methods may support asymmetric authentication, with one type of credential being required for the peer and another type for the authenticator. Hosts supporting peer-to-peer operation with such a method would need to be provisioned with both types of credentials.

For example, EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13] is a client-server protocol in which distinct certificate profiles are typically utilized for the client and server. This implies that a host supporting peer-to-peer authentication with EAP-TLS would need to implement both the EAP peer and authenticator layers, support both peer and authenticator roles in the EAP-TLS implementation, and provision certificates appropriate for each role.

AAA protocols such as RADIUS/EAP [RFC3579] and Diameter EAP [RFC4072] only support "pass-through authenticator" operation. As noted in [RFC3579] Section 2.6.2, a RADIUS server responds to an Access-Request encapsulating an EAP-Request, Success, or Failure packet with an Access-Reject. There is therefore no support for "pass-through peer" operation.

Even where a method is used which supports mutual authentication and result indications, several considerations may dictate that two EAP authentications (one in each direction) are required. These include:

1. Support for bi-directional session key derivation in the lower layer. Lower layers such as IEEE 802.11 may only support uni-directional derivation and transport of transient session keys. For example, the group-key handshake defined in [IEEE-802.11i] is uni-directional, since in IEEE 802.11 infrastructure mode, only the Access Point (AP) sends multicast/broadcast traffic. In IEEE 802.11 ad hoc mode, where either peer may send multicast/broadcast traffic, two uni-directional group-key exchanges are required. Due to limitations of the design, this also implies the need for unicast key derivations and EAP method exchanges to occur in each direction.

2. Support for tie-breaking in the lower layer. Lower layers such as IEEE 802.11 ad hoc do not support "tie breaking" wherein two hosts initiating authentication with each other will only go forward with a single authentication. This implies that even if 802.11 were to support a bi-directional group-key handshake, then two authentications, one in each direction, might still occur.
3. Peer policy satisfaction. EAP methods may support result indications, enabling the peer to indicate to the EAP server within the method that it successfully authenticated the EAP server, as well as for the server to indicate that it has authenticated the peer. However, a pass-through authenticator will not be aware that the peer has accepted the credentials offered by the EAP server, unless this information is provided to the authenticator via the AAA protocol. The authenticator SHOULD interpret the receipt of a key attribute within an Accept packet as an indication that the peer has successfully authenticated the server.

However, it is possible that the EAP peer's access policy was not satisfied during the initial EAP exchange, even though mutual authentication occurred. For example, the EAP authenticator may not have demonstrated authorization to act in both peer and authenticator roles. As a result, the peer may require an additional authentication in the reverse direction, even if the peer provided an indication that the EAP server had successfully authenticated to it.

### 3. Lower Layer Behavior

#### 3.1. Lower Layer Requirements

EAP makes the following assumptions about lower layers:

1. Unreliable transport. In EAP, the authenticator retransmits Requests that have not yet received Responses so that EAP does not assume that lower layers are reliable. Since EAP defines its own retransmission behavior, it is possible (though undesirable) for retransmission to occur both in the lower layer and the EAP layer when EAP is run over a reliable lower layer.

Note that EAP Success and Failure packets are not retransmitted. Without a reliable lower layer, and with a non-negligible error rate, these packets can be lost, resulting in timeouts. It is therefore desirable for implementations to improve their resilience to loss of EAP Success or Failure packets, as described in Section 4.2.



2. Lower layer error detection. While EAP does not assume that the lower layer is reliable, it does rely on lower layer error detection (e.g., CRC, Checksum, MIC, etc.). EAP methods may not include a MIC, or if they do, it may not be computed over all the fields in the EAP packet, such as the Code, Identifier, Length, or Type fields. As a result, without lower layer error detection, undetected errors could creep into the EAP layer or EAP method layer header fields, resulting in authentication failures.

For example, EAP TLS [RFC5216][I-D.ietf-emu-eap-tls13], which computes its MIC over the Type-Data field only, regards MIC validation failures as a fatal error. Without lower layer error detection, this method, and others like it, will not perform reliably.

3. Lower layer security. EAP does not require lower layers to provide security services such as per-packet confidentiality, authentication, integrity, and replay protection. However, where these security services are available, EAP methods supporting Key Derivation (see Section 7.2.1) can be used to provide dynamic keying material. This makes it possible to bind the EAP authentication to subsequent data and protect against data modification, spoofing, or replay. See Section 7.1 for details.
4. Minimum MTU. EAP is capable of functioning on lower layers that provide an EAP MTU size of 1020 octets or greater.

EAP does not support path MTU discovery, and fragmentation and reassembly is not supported by EAP, nor by the methods defined in this specification: Identity (1), Notification (2), Nak Response (3), MD5-Challenge (4), One Time Password (5), Generic Token Card (6), and expanded Nak Response (254) Types.

Typically, the EAP peer obtains information on the EAP MTU from the lower layers and sets the EAP frame size to an appropriate value. Where the authenticator operates in pass-through mode, the authentication server does not have a direct way of determining the EAP MTU, and therefore relies on the authenticator to provide it with this information, such as via the Framed-MTU attribute, as described in [RFC3579], Section 2.4.

While methods such as EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13] support fragmentation and reassembly, EAP methods originally designed for use within PPP where a 1500 octet MTU is guaranteed for control frames (see [RFC1661], Section 6.1) may lack fragmentation and reassembly features.

EAP methods can assume a minimum EAP MTU of 1020 octets in the absence of other information. EAP methods SHOULD include support for fragmentation and reassembly if their payloads can be larger than this minimum EAP MTU.

EAP is a lock-step protocol, which implies a certain inefficiency when handling fragmentation and reassembly. Therefore, if the lower layer supports fragmentation and reassembly (such as where EAP is transported over IP), it may be preferable for fragmentation and reassembly to occur in the lower layer rather than in EAP. This can be accomplished by providing an artificially large EAP MTU to EAP, causing fragmentation and reassembly to be handled within the lower layer.

5. Possible duplication. Where the lower layer is reliable, it will provide the EAP layer with a non-duplicated stream of packets. However, while it is desirable that lower layers provide for non-duplication, this is not a requirement. The Identifier field provides both the peer and authenticator with the ability to detect duplicates.
6. Ordering guarantees. EAP does not require the Identifier to be monotonically increasing, and so is reliant on lower layer ordering guarantees for correct operation. EAP was originally defined to run on PPP, and [RFC1661] Section 1 has an ordering requirement:

"The Point-to-Point Protocol is designed for simple links which transport packets between two peers. These links provide full-duplex simultaneous bi-directional operation, and are assumed to deliver packets in order."

Lower layer transports for EAP MUST preserve ordering between a source and destination at a given priority level (the ordering guarantee provided by [IEEE-802]).

Reordering, if it occurs, will typically result in an EAP authentication failure, causing EAP authentication to be re-run. In an environment in which reordering is likely, it is therefore expected that EAP authentication failures will be common. It is RECOMMENDED that EAP only be run over lower layers that provide ordering guarantees; running EAP over raw IP or UDP transport is

NOT RECOMMENDED. Encapsulation of EAP within RADIUS [RFC3579] satisfies ordering requirements, since RADIUS is a "lockstep" protocol that delivers packets in order.

### 3.2. EAP Usage Within PPP

In order to establish communications over a point-to-point link, each end of the PPP link first sends LCP packets to configure the data link during the Link Establishment phase. After the link has been established, PPP provides for an optional Authentication phase before proceeding to the Network-Layer Protocol phase.

By default, authentication is not mandatory. If authentication of the link is desired, an implementation **MUST** specify the Authentication Protocol Configuration Option during the Link Establishment phase.

If the identity of the peer has been established in the Authentication phase, the server can use that identity in the selection of options for the following network layer negotiations.

When implemented within PPP, EAP does not select a specific authentication mechanism at the PPP Link Control Phase, but rather postpones this until the Authentication Phase. This allows the authenticator to request more information before determining the specific authentication mechanism. This also permits the use of a "backend" server which actually implements the various mechanisms while the PPP authenticator merely passes through the authentication exchange. The PPP Link Establishment and Authentication phases, and the Authentication Protocol Configuration Option, are defined in The Point-to-Point Protocol (PPP) [RFC1661].

#### 3.2.1. PPP Configuration Option Format

A summary of the PPP Authentication Protocol Configuration Option format to negotiate EAP follows. The fields are transmitted from left to right.

Exactly one EAP packet is encapsulated in the Information field of a PPP Data Link Layer frame where the protocol field indicates type hex C227 (PPP EAP).

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type										Length										Authentication Protocol																			

Type

3

Length

4

Authentication Protocol

C227 (Hex) for Extensible Authentication Protocol (EAP)

### 3.3. EAP Usage Within IEEE 802

The encapsulation of EAP over IEEE 802 is defined in [IEEE-802.1X]. The IEEE 802 encapsulation of EAP does not involve PPP, and IEEE 802.1X does not include support for link or network layer negotiations. As a result, within IEEE 802.1X, it is not possible to negotiate non-EAP authentication mechanisms, such as PAP or CHAP [RFC1994].

### 3.4. Lower Layer Indications

The reliability and security of lower layer indications is dependent on the lower layer. Since EAP is media independent, the presence or absence of lower layer security is not taken into account in the processing of EAP messages.

To improve reliability, if a peer receives a lower layer success indication as defined in Section 7.12, it MAY conclude that a Success packet has been lost, and behave as if it had actually received a Success packet. This includes choosing to ignore the Success in some circumstances as described in Section 4.2. See also protected result indications in Section 7.16.

A discussion of some reliability and security issues with lower layer indications in PPP, IEEE 802 wired networks, and IEEE 802.11 wireless LANs can be found in the Security Considerations, Section 7.12.

After EAP authentication is complete, the peer will typically transmit and receive data via the authenticator. It is desirable to provide assurance that the entities transmitting data are the same ones that successfully completed EAP authentication. To accomplish this, it is necessary for the lower layer to provide per-packet integrity, authentication and replay protection, and to bind these per-packet services to the keys derived during EAP authentication. Otherwise, it is possible for subsequent data traffic to be modified, spoofed, or replayed.

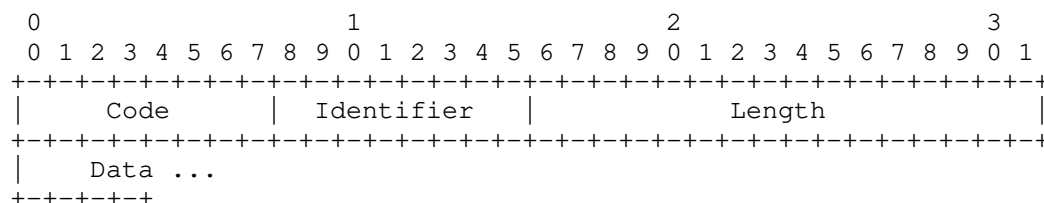
Where keying material for the lower layer ciphersuite is itself provided by EAP, ciphersuite negotiation and key activation are controlled by the lower layer. In PPP, ciphersuites are negotiated

within ECP so that it is not possible to use keys derived from EAP authentication until the completion of ECP. Therefore, an initial EAP exchange cannot be protected by a PPP ciphersuite, although EAP re-authentication can be protected.

In IEEE 802 media, initial key activation also typically occurs after completion of EAP authentication. Therefore an initial EAP exchange typically cannot be protected by the lower layer ciphersuite, although an EAP re-authentication or pre-authentication exchange can be protected.

#### 4. EAP Packet Format

A summary of the EAP packet format is shown below. The fields are transmitted from left to right.



##### Code

The Code field is one octet and identifies the Type of EAP packet. EAP Codes are assigned as follows:

- |   |          |
|---|----------|
| 1 | Request  |
| 2 | Response |
| 3 | Success  |
| 4 | Failure  |

Since EAP only defines Codes 1-4, EAP packets with other codes MUST be silently discarded by both authenticators and peers.

##### Identifier

The Identifier field is one octet and aids in matching Responses with Requests.

##### Length

The Length field is two octets and indicates the length, in octets, of the EAP packet including the Code, Identifier, Length, and Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and MUST be ignored upon reception. A message with the Length field set to a value larger than the number of received octets MUST be silently discarded.

#### Data

The Data field is zero or more octets. The format of the Data field is determined by the Code field.

### 4.1. Request and Response

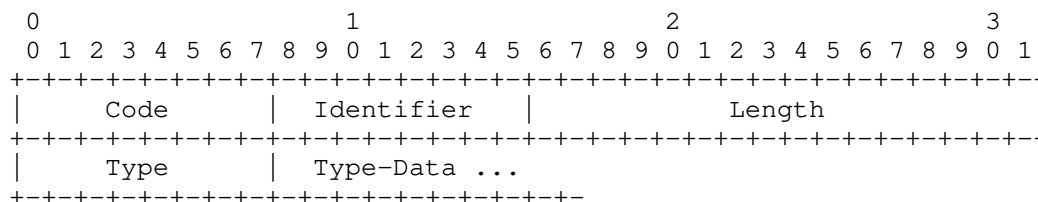
#### Description

The Request packet (Code field set to 1) is sent by the authenticator to the peer. Each Request has a Type field which serves to indicate what is being requested. Additional Request packets MUST be sent until a valid Response packet is received, an optional retry counter expires, or a lower layer failure indication is received.

Retransmitted Requests MUST be sent with the same Identifier value in order to distinguish them from new Requests. The content of the data field is dependent on the Request Type. The peer MUST send a Response packet in reply to a valid Request packet. Responses MUST only be sent in reply to a valid Request and never be retransmitted on a timer.

If a peer receives a valid duplicate Request for which it has already sent a Response, it MUST resend its original Response without reprocessing the Request. Requests MUST be processed in the order that they are received, and MUST be processed to their completion before inspecting the next Request.

A summary of the Request and Response packet format follows. The fields are transmitted from left to right.



#### Code

- 1 for Request
- 2 for Response

#### Identifier

The Identifier field is one octet. The Identifier field MUST be the same if a Request packet is retransmitted due to a timeout while waiting for a Response. Any new (non-retransmission) Requests MUST modify the Identifier field.

The Identifier field of the Response MUST match that of the currently outstanding Request. An authenticator receiving a Response whose Identifier value does not match that of the currently outstanding Request MUST silently discard the Response.

In order to avoid confusion between new Requests and retransmissions, the Identifier value chosen for each new Request need only be different from the previous Request, but need not be unique within the conversation. One way to achieve this is to start the Identifier at an initial value and increment it for each new Request. Initializing the first Identifier with a random number rather than starting from zero is recommended, since it makes sequence attacks somewhat more difficult.

Since the Identifier space is unique to each session, authenticators are not restricted to only 256 simultaneous authentication conversations. Similarly, with re-authentication, an EAP conversation might continue over a long period of time, and is not limited to only 256 roundtrips.

Implementation Note: The authenticator is responsible for retransmitting Request messages. If the Request message is obtained from elsewhere (such as from a backend authentication server), then the authenticator will need to save a copy of the Request in order to accomplish this. The peer is responsible for

detecting and handling duplicate Request messages before processing them in any way, including passing them on to an outside party. The authenticator is also responsible for discarding Response messages with a non-matching Identifier value before acting on them in any way, including passing them on to the backend authentication server for verification. Since the authenticator can retransmit before receiving a Response from the peer, the authenticator can receive multiple Responses, each with a matching Identifier. Until a new Request is received by the authenticator, the Identifier value is not updated, so that the authenticator forwards Responses to the backend authentication server, one at a time.

#### Length

The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, and Type-Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and MUST be ignored upon reception. A message with the Length field set to a value larger than the number of received octets MUST be silently discarded.

#### Type

The Type field is one octet. This field indicates the Type of Request or Response. A single Type MUST be specified for each EAP Request or Response. An initial specification of Types follows in Section 5 of this document.

The Type field of a Response MUST either match that of the Request, or correspond to a legacy or Expanded Nak (see Section 5.3) indicating that a Request Type is unacceptable to the peer. A peer MUST NOT send a Nak (legacy or expanded) in response to a Request, after an initial non-Nak Response has been sent. An EAP server receiving a Response not meeting these requirements MUST silently discard it.

#### Type-Data

The Type-Data field varies with the Type of Request and the associated Response.

### 4.2. Success and Failure

The Success packet is sent by the authenticator to the peer after completion of an EAP authentication method (Type 4 or greater) to indicate that the peer has authenticated successfully to the authenticator. The authenticator MUST transmit an EAP packet with



the Code field set to 3 (Success). If the authenticator cannot authenticate the peer (unacceptable Responses to one or more Requests), then after unsuccessful completion of the EAP method in progress, the implementation MUST transmit an EAP packet with the Code field set to 4 (Failure). An authenticator MAY wish to issue multiple Requests before sending a Failure response in order to allow for human typing mistakes. Success and Failure packets MUST NOT contain additional data.

Success and Failure packets MUST NOT be sent by an EAP authenticator if the specification of the given method does not explicitly permit the method to finish at that point. A peer EAP implementation receiving a Success or Failure packet where sending one is not explicitly permitted MUST silently discard it. By default, an EAP peer MUST silently discard a "canned" Success packet (a Success packet sent immediately upon connection). This ensures that a rogue authenticator will not be able to bypass mutual authentication by sending a Success packet prior to conclusion of the EAP method conversation.

Implementation Note: Because the Success and Failure packets are not acknowledged, they are not retransmitted by the authenticator, and may be potentially lost. A peer MUST allow for this circumstance as described in this note. See also Section 3.4 for guidance on the processing of lower layer success and failure indications.

As described in Section 2.1, only a single EAP authentication method is allowed within an EAP conversation. EAP methods may implement result indications. After the authenticator sends a failure result indication to the peer, regardless of the response from the peer, it MUST subsequently send a Failure packet. After the authenticator sends a success result indication to the peer and receives a success result indication from the peer, it MUST subsequently send a Success packet.

On the peer, once the method completes unsuccessfully (that is, either the authenticator sends a failure result indication, or the peer decides that it does not want to continue the conversation, possibly after sending a failure result indication), the peer MUST terminate the conversation and indicate failure to the lower layer. The peer MUST silently discard Success packets and MAY silently discard Failure packets. As a result, loss of a Failure packet need not result in a timeout.

On the peer, after success result indications have been exchanged by both sides, a Failure packet MUST be silently discarded. The peer MAY, in the event that an EAP Success is not received, conclude that

the EAP Success packet was lost and that authentication concluded successfully.

If the authenticator has not sent a result indication, and the peer is willing to continue the conversation, the peer waits for a Success or Failure packet once the method completes, and MUST NOT silently discard either of them. In the event that neither a Success nor Failure packet is received, the peer SHOULD terminate the conversation to avoid lengthy timeouts in case the lost packet was an EAP Failure.

If the peer attempts to authenticate to the authenticator and fails to do so, the authenticator MUST send a Failure packet and MUST NOT grant access by sending a Success packet. However, an authenticator MAY omit having the peer authenticate to it in situations where limited access is offered (e.g., guest access). In this case, the authenticator MUST send a Success packet.

Where the peer authenticates successfully to the authenticator, but the authenticator does not send a result indication, the authenticator MAY deny access by sending a Failure packet where the peer is not currently authorized for network access.

A summary of the Success and Failure packet format is shown below. The fields are transmitted from left to right.

0									1									2									3								
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
Code									Identifier									Length																	

Code

3 for Success  
4 for Failure

Identifier

The Identifier field is one octet and aids in matching replies to Responses. The Identifier field MUST match the Identifier field of the Response packet that it is sent in response to.

Length

#### 4.3. Retransmission Behavior

Because the authentication process will often involve user input, some care must be taken when deciding upon retransmission strategies and authentication timeouts. By default, where EAP is run over an unreliable lower layer, the EAP retransmission timer SHOULD be dynamically estimated. A maximum of 3-5 retransmissions is suggested.

When run over a reliable lower layer (e.g., EAP over ISAKMP/TCP, as within [I-D.ietf-ipsra-pic]), the authenticator retransmission timer SHOULD be set to an infinite value, so that retransmissions do not occur at the EAP layer. The peer may still maintain a timeout value so as to avoid waiting indefinitely for a Request.

Where the authentication process requires user input, the measured round trip times may be determined by user responsiveness rather than network characteristics, so that dynamic RTO estimation may not be helpful. Instead, the retransmission timer SHOULD be set so as to provide sufficient time for the user to respond, with longer timeouts required in certain cases, such as where Token Cards (see Section 5.6) are involved.

In order to provide the EAP authenticator with guidance as to the appropriate timeout value, a hint can be communicated to the authenticator by the backend authentication server (such as via the RADIUS Session-Timeout attribute).

In order to dynamically estimate the EAP retransmission timer, the algorithms for the estimation of SRTT, RTTVAR, and RTO described in [RFC6298] are RECOMMENDED, including use of Karn's algorithm, with the following potential modifications:

- o In order to avoid synchronization behaviors that can occur with fixed timers among distributed systems, the retransmission timer is calculated with a jitter by using the RTO value and randomly adding a value drawn between  $-RTO_{min}/2$  and  $RTO_{min}/2$ . Alternative calculations to create jitter MAY be used. These MUST be pseudo-random. For a discussion of pseudo-random number generation, see [RFC1750].
- o When EAP is transported over a single link (as opposed to over the Internet), smaller values of  $RTO_{initial}$ ,  $RTO_{min}$ , and  $RTO_{max}$  MAY be used. Recommended values are  $RTO_{initial}=1$  second,  $RTO_{min}=200ms$ , and  $RTO_{max}=20$  seconds.

- o When EAP is transported over a single link (as opposed to over the Internet), estimates MAY be done on a per-authenticator basis, rather than a per-session basis. This enables the retransmission estimate to make the most use of information on link-layer behavior.
- o An EAP implementation MAY clear SRTT and RTTVAR after backing off the timer multiple times, as it is likely that the current SRTT and RTTVAR are bogus in this situation. Once SRTT and RTTVAR are cleared, they should be initialized with the next RTT sample taken as described in [RFC6298] equation 2.2.

## 5. Initial EAP Request/Response Types

This section defines the initial set of EAP Types used in Request/Response exchanges. More Types may be defined in future documents. The Type field is one octet and identifies the structure of an EAP Request or Response packet. The first 3 Types are considered special case Types.

The remaining Types define authentication exchanges. Nak (Type 3) or Expanded Nak (Type 254) are valid only for Response packets, they MUST NOT be sent in a Request.

All EAP implementations MUST support Types 1-4, which are defined in this document, and SHOULD support Type 254. Implementations MAY support other Types defined here or in future RFCs.

1	Identity
2	Notification
3	Nak (Response only)
4	MD5-Challenge
5	One Time Password (OTP)
6	Generic Token Card (GTC)
254	Expanded Types
255	Experimental use

EAP methods MAY support authentication based on shared secrets. If the shared secret is a passphrase entered by the user, implementations MAY support entering passphrases with non-ASCII characters. In this case, the input should be processed using an appropriate stringprep [RFC3454] profile, and encoded in octets using UTF-8 encoding [RFC3629]. A preliminary version of a possible stringprep profile is described in [RFC8265].

## 5.1. Identity

### Description

The Identity Type is used to query the identity of the peer. Generally, the authenticator will issue this as the initial Request. An optional displayable message MAY be included to prompt the peer in the case where there is an expectation of interaction with a user. A Response of Type 1 (Identity) SHOULD be sent in Response to a Request with a Type of 1 (Identity).

Some EAP implementations piggy-back various options into the Identity Request after a NUL-character. By default, an EAP implementation SHOULD NOT assume that an Identity Request or Response can be larger than 1020 octets.

It is RECOMMENDED that the Identity Response be used primarily for routing purposes and selecting which EAP method to use. EAP Methods SHOULD include a method-specific mechanism for obtaining the identity, so that they do not have to rely on the Identity Response. Identity Requests and Responses are sent in cleartext, so an attacker may snoop on the identity, or even modify or spoof identity exchanges. To address these threats, it is preferable for an EAP method to include an identity exchange that supports per-packet authentication, integrity and replay protection, and confidentiality. The Identity Response may not be the appropriate identity for the method; it may have been truncated or obfuscated so as to provide privacy, or it may have been decorated for routing purposes. Where the peer is configured to only accept authentication methods supporting protected identity exchanges, the peer MAY provide an abbreviated Identity Response (such as omitting the peer-name portion of the NAI [RFC2486]). For further discussion of identity protection, see Section 7.3.

Implementation Note: The peer MAY obtain the Identity via user input. It is suggested that the authenticator retry the Identity Request in the case of an invalid Identity or authentication failure to allow for potential typos on the part of the user. It is suggested that the Identity Request be retried a minimum of 3 times before terminating the authentication. The Notification Request MAY be used to indicate an invalid authentication attempt prior to transmitting a new Identity Request (optionally, the failure MAY be indicated within the message of the new Identity Request itself).

### Type

1

## Type-Data

This field MAY contain a displayable message in the Request, containing UTF-8 encoded ISO 10646 characters [RFC3629]. Where the Request contains a null, only the portion of the field prior to the null is displayed. If the Identity is unknown, the Identity Response field should be zero bytes in length. The Identity Response field MUST NOT be null terminated. In all cases, the length of the Type-Data field is derived from the Length field of the Request/Response packet.

Security Claims (see Section 7.2):

Auth. mechanism:	None
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	N/A
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

## 5.2. Notification

### Description

The Notification Type is optionally used to convey a displayable message from the authenticator to the peer. An authenticator MAY send a Notification Request to the peer at any time when there is no outstanding Request, prior to completion of an EAP authentication method. The peer MUST respond to a Notification Request with a Notification Response unless the EAP authentication method specification prohibits the use of Notification messages. In any case, a Nak Response MUST NOT be sent in response to a Notification Request. Note that the default maximum length of a Notification Request is 1020 octets. By default, this leaves at most 1015 octets for the human readable message.

An EAP method MAY indicate within its specification that Notification messages must not be sent during that method. In this case, the peer MUST silently discard Notification Requests from the point where an initial Request for that Type is answered with a Response of the same Type.

The peer SHOULD display this message to the user or log it if it cannot be displayed. The Notification Type is intended to provide an acknowledged notification of some imperative nature, but it is not an error indication, and therefore does not change the state of the peer. Examples include a password with an expiration time that is about to expire, an OTP sequence integer which is nearing 0, an authentication failure warning, etc. In most circumstances, Notification should not be required.

#### Type

2

#### Type-Data

The Type-Data field in the Request contains a displayable message greater than zero octets in length, containing UTF-8 encoded ISO 10646 characters [RFC3629]. The length of the message is determined by the Length field of the Request packet. The message MUST NOT be null terminated. A Response MUST be sent in reply to the Request with a Type field of 2 (Notification). The Type-Data field of the Response is zero octets in length. The Response should be sent immediately (independent of how the message is displayed or logged).

Security Claims (see Section 7.2):

Auth. mechanism:	None
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	N/A
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

### 5.3. Nak

#### 5.3.1. Legacy Nak

##### Description

The legacy Nak Type is valid only in Response messages. It is sent in reply to a Request where the desired authentication Type is unacceptable. Authentication Types are numbered 4 and above. The Response contains one or more authentication Types desired by the Peer. Type zero (0) is used to indicate that the sender has no viable alternatives, and therefore the authenticator SHOULD NOT send another Request after receiving a Nak Response containing a zero value.

Since the legacy Nak Type is valid only in Responses and has very limited functionality, it MUST NOT be used as a general purpose error indication, such as for communication of error messages, or negotiation of parameters specific to a particular EAP method.

##### Code

2 for Response.

##### Identifier

The Identifier field is one octet and aids in matching Responses with Requests. The Identifier field of a legacy Nak Response MUST match the Identifier field of the Request packet that it is sent in response to.

##### Length



>=6

Type

3

Type-Data

Where a peer receives a Request for an unacceptable authentication Type (4-253,255), or a peer lacking support for Expanded Types receives a Request for Type 254, a Nak Response (Type 3) MUST be sent. The Type-Data field of the Nak Response (Type 3) MUST contain one or more octets indicating the desired authentication Type(s), one octet per Type, or the value zero (0) to indicate no proposed alternative. A peer supporting Expanded Types that receives a Request for an unacceptable authentication Type (4-253, 255) MAY include the value 254 in the Nak Response (Type 3) to indicate the desire for an Expanded authentication Type. If the authenticator can accommodate this preference, it will respond with an Expanded Type Request (Type 254).

Security Claims (see Section 7.2):

Auth. mechanism:	None
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	N/A
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

### 5.3.2. Expanded Nak

Description

The Expanded Nak Type is valid only in Response messages. It MUST be sent only in reply to a Request of Type 254 (Expanded Type) where the authentication Type is unacceptable. The Expanded Nak

Type uses the Expanded Type format itself, and the Response contains one or more authentication Types desired by the peer, all in Expanded Type format. Type zero (0) is used to indicate that the sender has no viable alternatives. The general format of the Expanded Type is described in Section 5.7.

Since the Expanded Nak Type is valid only in Responses and has very limited functionality, it MUST NOT be used as a general purpose error indication, such as for communication of error messages, or negotiation of parameters specific to a particular EAP method.

#### Code

2 for Response.

#### Identifier

The Identifier field is one octet and aids in matching Responses with Requests. The Identifier field of an Expanded Nak Response MUST match the Identifier field of the Request packet that it is sent in response to.

#### Length

>=20

#### Type

254

#### Vendor-Id

0 (IETF)

#### Vendor-Type

3 (Nak)

#### Vendor-Data

The Expanded Nak Type is only sent when the Request contains an Expanded Type (254) as defined in Section 5.7. The Vendor-Data field of the Nak Response MUST contain one or more authentication Types (4 or greater), all in expanded format, 8 octets per Type, or the value zero (0), also in Expanded Type format, to indicate no proposed alternative. The desired authentication Types may include a mixture of Vendor-Specific and IETF Types. For example,

an Expanded Nak Response indicating a preference for OTP (Type 5), and an MIT (Vendor-Id=20) Expanded Type of 6 would appear as follows:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
2										Identifier										Length=28																			
Type=254										0 (IETF)																													
										3 (Nak)																													
Type=254										0 (IETF)																													
										5 (OTP)																													
Type=254										20 (MIT)																													
										6																													

An Expanded Nak Response indicating a no desired alternative would appear as follows:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
2										Identifier										Length=20																			
Type=254										0 (IETF)																													
										3 (Nak)																													
Type=254										0 (IETF)																													
										0 (No alternative)																													

Security Claims (see Section 7.2):

Auth. mechanism:	None
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	N/A
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

#### 5.4. MD5-Challenge

##### Description

The MD5-Challenge Type is analogous to the PPP CHAP protocol [RFC1994] (with MD5 as the specified algorithm). The Request contains a "challenge" message to the peer. A Response MUST be sent in reply to the Request. The Response MAY be either of Type 4 (MD5-Challenge), Nak (Type 3), or Expanded Nak (Type 254). The Nak reply indicates the peer's desired authentication Type(s). EAP peer and EAP server implementations MUST support the MD5-Challenge mechanism. An authenticator that supports only pass-through MUST allow communication with a backend authentication server that is capable of supporting MD5-Challenge, although the EAP authenticator implementation need not support MD5-Challenge itself. However, if the EAP authenticator can be configured to authenticate peers locally (e.g., not operate in pass-through), then the requirement for support of the MD5-Challenge mechanism applies.

Note that the use of the Identifier field in the MD5-Challenge Type is different from that described in [RFC1994]. EAP allows for retransmission of MD5-Challenge Request packets, while [RFC1994] states that both the Identifier and Challenge fields MUST change each time a Challenge (the CHAP equivalent of the MD5-Challenge Request packet) is sent.

Note 1. MD5 algorithm has severe issues, particularly when used without HMAC (which is not used by CHAP or EAP-MD5). For more information, refer to Section 7.11.1.

Note 2: [RFC1994] treats the shared secret as an octet string, and does not specify how it is entered into the system (or if it is handled by the user at all). EAP MD5-Challenge implementations MAY support entering passphrases with non-ASCII characters. See Section 5 for instructions how the input should be processed and encoded into octets.

Type

4

Type-Data

The contents of the Type-Data field is summarized below. For reference on the use of these fields, see the PPP Challenge Handshake Authentication Protocol [RFC1994].

[illegible]

Security Claims (see Section 7.2):

Auth. mechanism:	Password or pre-shared key.
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	No
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

### 5.5. One-Time Password (OTP)

#### Description

The One-Time Password system is defined in "A One-Time Password System" [RFC2289] and "OTP Extended Responses" [RFC2243]. The Request contains an OTP challenge in the format described in [RFC2289]. A Response MUST be sent in reply to the Request. The Response MUST be of Type 5 (OTP), Nak (Type 3), or Expanded Nak (Type 254). The Nak Response indicates the peer's desired authentication Type(s). The EAP OTP method is intended for use with the One-Time Password system only, and MUST NOT be used to provide support for cleartext passwords.

#### Type

5

#### Type-Data

The Type-Data field contains the OTP "challenge" as a displayable message in the Request. In the Response, this field is used for the 6 words from the OTP dictionary [RFC2289]. The messages MUST NOT be null terminated. The length of the field is derived from the Length field of the Request/Reply packet.

Note: [RFC2289] does not specify how the secret pass-phrase is entered by the user, or how the pass-phrase is converted into octets. EAP OTP implementations MAY support entering passphrases with non-ASCII characters. See Section 5 for instructions on how the input should be processed and encoded into octets.

Security Claims (see Section 7.2):

Auth. mechanism:	One-Time Password
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	Yes
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	No
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

## 5.6. Generic Token Card (GTC)

### Description

The Generic Token Card Type is defined for use with various Token Card implementations which require user input. The Request contains a displayable message and the Response contains the Token Card information necessary for authentication. Typically, this would be information read by a user from the Token card device and entered as ASCII text. A Response MUST be sent in reply to the Request. The Response MUST be of Type 6 (GTC), Nak (Type 3), or Expanded Nak (Type 254). The Nak Response indicates the peer's desired authentication Type(s). The EAP GTC method is intended for use with the Token Cards supporting challenge/response authentication and MUST NOT be used to provide support for cleartext passwords in the absence of a protected tunnel with server authentication.

### Type

6

### Type-Data

The Type-Data field in the Request contains a displayable message greater than zero octets in length. The length of the message is determined by the Length field of the Request packet. The message MUST NOT be null terminated. A Response MUST be sent in reply to the Request with a Type field of 6 (Generic Token Card). The Response contains data from the Token Card required for

authentication. The length of the data is determined by the Length field of the Response packet.

EAP GTC implementations MAY support entering a response with non-ASCII characters. See Section 5 for instructions how the input should be processed and encoded into octets.

Security Claims (see Section 7.2):

Auth. mechanism:	Hardware token.
Ciphersuite negotiation:	No
Mutual authentication:	No
Integrity protection:	No
Replay protection:	No
Confidentiality:	No
Key derivation:	No
Key strength:	N/A
Dictionary attack prot.:	No
Fast reconnect:	No
Crypt. binding:	N/A
Session independence:	N/A
Fragmentation:	No
Channel binding:	No
Perfect Forward Secrecy:	N/A

## 5.7. Expanded Types

### Description

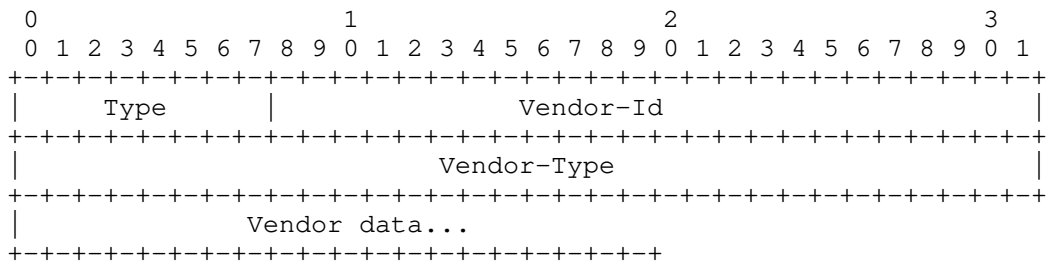
Since many of the existing uses of EAP are vendor-specific, the Expanded method Type is available to allow vendors to support their own Expanded Types not suitable for general usage.

The Expanded Type is also used to expand the global Method Type space beyond the original 255 values. A Vendor-Id of 0 maps the original 255 possible Types onto a space of  $2^{32}-1$  possible Types. (Type 0 is only used in a Nak Response to indicate no acceptable alternative).

An implementation that supports the Expanded attribute MUST treat EAP Types that are less than 256 equivalently, whether they appear as a single octet or as the 32-bit Vendor-Type within an Expanded Type where Vendor-Id is 0. Peers not equipped to interpret the Expanded Type MUST send a Nak as described in Section 5.3.1, and negotiate a more suitable authentication method.

A summary of the Expanded Type format is shown below. The fields are transmitted from left to right.





Type

254 for Expanded Type

Vendor-Id

The Vendor-Id is 3 octets and represents the SMI Network Management Private Enterprise Code of the Vendor in network byte order, as allocated by IANA. A Vendor-Id of zero is reserved for use by the IETF in providing an expanded global EAP Type space.

Vendor-Type

The Vendor-Type field is four octets and represents the vendor-specific method Type.

If the Vendor-Id is zero, the Vendor-Type field is an extension and superset of the existing namespace for EAP Types. The first 256 Types are reserved for compatibility with single-octet EAP Types that have already been assigned or may be assigned in the future. Thus, EAP Types from 0 through 255 are semantically identical, whether they appear as single octet EAP Types or as Vendor-Types when Vendor-Id is zero. There is one exception to this rule: Expanded Nak and Legacy Nak packets share the same Type, but must be treated differently because they have a different format.

Vendor-Data

The Vendor-Data field is defined by the vendor. Where a Vendor-Id of zero is present, the Vendor-Data field will be used for transporting the contents of EAP methods of Types defined by the IETF.

## 5.8. Experimental

### Description

The Experimental Type has no fixed format or content. It is intended for use when experimenting with new EAP Types. This Type is intended for experimental and testing purposes. No guarantee is made for interoperability between peers using this Type, as outlined in [RFC3692].

### Type

255

### Type-Data

### Undefined

## 6. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP protocol, in accordance with BCP 26, [RFC8126].

There are two name spaces in EAP that require registration: Packet Codes and method Types.

EAP is not intended as a general-purpose protocol, and allocations SHOULD NOT be made for purposes unrelated to authentication.

The following terms are used here with the meanings defined in BCP 26: "name space", "assigned value", "registration".

The following policies are used here with the meanings defined in BCP 26: "Private Use", "First Come First Served", "Expert Review", "Specification Required", "IETF Review", "Standards Action".

For registration requests where a Designated Expert should be consulted, the responsible IESG area director should appoint the Designated Expert. The intention is that any allocation will be accompanied by a published RFC. But in order to allow for the allocation of values prior to the RFC being approved for publication, the Designated Expert can approve allocations once it seems clear that an RFC will be published. The Designated expert will post a

request to the EAP WG mailing list (or a successor designated by the Area Director) for comment and review, including an Internet-Draft. Before a period of 30 days has passed, the Designated Expert will either approve or deny the registration request and publish a notice of the decision to the EAP WG mailing list or its successor, as well as informing IANA. A denial notice must be justified by an explanation, and in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable should be provided.

#### 6.1. Packet Codes

Packet Codes have a range from 1 to 255, of which 1-4 have been allocated by this document and 5-6 by [RFC6696]. Because a new Packet Code has considerable impact on interoperability, a new Packet Code requires Standards Action, and should be allocated starting at 5.

#### 6.2. Method Types

The original EAP method Type space has a range from 1 to 255, and is the scarcest resource in EAP, and thus must be allocated with care. Method Type 0 is reserved. Method Types 1-55 have been allocated, with 20 available for re-use. Method Types 20 and 56-191 may be allocated through Expert Review, on the advice of a Designated Expert, with Specification Required.

Allocation of blocks of method Types (more than one for a given purpose) should require IETF Review. EAP Type Values 192-253 are reserved and allocation requires Standards Action.

Method Type 254 is allocated for the Expanded Type. Where the Vendor-Id field is non-zero, the Expanded Type is used for functions specific only to one vendor's implementation of EAP, where no interoperability is deemed useful. When used with a Vendor-Id of zero, method Type 254 can also be used to provide for an expanded IETF method Type space. Method Type values 256-4294967295 may be allocated after Type values 1-191 have been allocated, on the advice of a Designated Expert, with Specification Required.

Method Type 255 is allocated for Experimental use, such as testing of new EAP methods before a permanent Type is allocated.

### 7. Security Considerations

This section defines a generic threat model as well as the EAP method security claims mitigating those threats.

It is expected that the generic threat model and corresponding security claims will be used to define EAP method requirements for use in specific environments. An example of such a requirements analysis is provided in [IEEE-802.11i-req]. A security claims section is required in EAP method specifications, so that EAP methods can be evaluated against the requirements.

### 7.1. Threat Model

EAP was developed for use with PPP [RFC1661] and was later adapted for use in wired IEEE 802 networks [IEEE-802] in [IEEE-802.1X] and 3GPP 5G [TS.33.501]. Subsequently, EAP has been proposed for use on wireless LAN networks and over the Internet. In all these situations, it is possible for an attacker to gain access to links over which EAP packets are transmitted. For example, attacks on telephone infrastructure are documented in [DECEPTION].

An attacker with access to the link may carry out a number of attacks, including:

1. An attacker may try to discover user identities by snooping authentication traffic.
2. An attacker may try to modify or spoof EAP packets.
3. An attacker may launch denial of service attacks by spoofing lower layer indications or Success/Failure packets, by replaying EAP packets, or by generating packets with overlapping Identifiers.
4. An attacker may attempt to recover the pass-phrase by mounting an offline dictionary attack.
5. An attacker may attempt to convince the peer to connect to an untrusted network by mounting a man-in-the-middle attack.
6. An attacker may attempt to disrupt the EAP negotiation in order to cause a weak authentication method to be selected.
7. An attacker may attempt to recover keys by taking advantage of weak key derivation techniques used within EAP methods.
8. An attacker may attempt to take advantage of weak ciphersuites subsequently used after the EAP conversation is complete.
9. An attacker may attempt to perform downgrading attacks on lower layer cipher suite negotiation in order to ensure that a weaker cipher suite is used subsequently to EAP authentication.

10. An attacker acting as an authenticator may provide incorrect information to the EAP peer and/or server via out-of-band mechanisms (such as via a AAA or lower layer protocol). This includes impersonating another authenticator, or providing inconsistent information to the peer and EAP server.

Depending on the lower layer, these attacks may be carried out without requiring physical proximity. Where EAP is used over wireless networks, EAP packets may be forwarded by authenticators (e.g., pre-authentication) so that the attacker need not be within the coverage area of an authenticator in order to carry out an attack on it or its peers. Where EAP is used over the Internet, attacks may be carried out at an even greater distance.

## 7.2. Security Claims

In order to clearly articulate the security provided by an EAP method, EAP method specifications MUST include a Security Claims section, including the following declarations:

- o Mechanism. This is a statement of the authentication technology: certificates, pre-shared keys, passwords, token cards, etc.
- o Security claims. This is a statement of the claimed security properties of the method, using terms defined in Section 7.2.1: mutual authentication, integrity protection, replay protection, confidentiality, key derivation, key strength, dictionary attack resistance, fast reconnect, cryptographic binding, session independence, fragmentation, channel binding, perfect forward secrecy. The Security Claims section of an EAP method specification SHOULD provide justification for the claims that are made. This can be accomplished by including a proof in an Appendix, or including a reference to a proof.
- o Key strength. If the method derives keys, then the effective key strength MUST be estimated. This estimate is meant for potential users of the method to determine if the keys produced are strong enough for the intended application.

The effective key strength SHOULD be stated as a number of bits, defined as follows: If the effective key strength is N bits, the best currently known methods to recover the key (with non-negligible probability) require, on average, an effort comparable to  $2^{(N-1)}$  operations of a typical block cipher. The statement SHOULD be accompanied by a short rationale, explaining how this number was derived. This explanation SHOULD include the parameters required to achieve the stated key strength based on current knowledge of the algorithms.

(Note: Although it is difficult to define what "comparable effort" and "typical block cipher" exactly mean, reasonable approximations are sufficient here. Refer to e.g. [SILVERMAN] for more discussion.)

The key strength depends on the methods used to derive the keys. For instance, if keys are derived from a shared secret (such as a password or a long-term secret), and possibly some public information such as nonces, the effective key strength is limited by the strength of the long-term secret (assuming that the derivation procedure is computationally simple). To take another example, when using public key algorithms, the strength of the symmetric key depends on the strength of the public keys used.

- o Description of key hierarchy. EAP methods deriving keys MUST either provide a reference to a key hierarchy specification, or describe how Master Session Keys (MSKs) and Extended Master Session Keys (EMSKs) are to be derived.
- o Indication of vulnerabilities. In addition to the security claims that are made, the specification MUST indicate which of the security claims detailed in Section 7.2.1 are NOT being made.

#### 7.2.1. Security Claims Terminology for EAP Methods

These terms are used to describe the security properties of EAP methods:

##### Protected ciphersuite negotiation

This refers to the ability of an EAP method to negotiate the ciphersuite used to protect the EAP conversation, as well as to integrity protect the negotiation. It does not refer to the ability to negotiate the ciphersuite used to protect data.

##### Mutual authentication

This refers to an EAP method in which, within an interlocked exchange, the authenticator authenticates the peer and the peer authenticates the authenticator. Two independent one-way methods, running in opposite directions do not provide mutual authentication as defined here.

##### Integrity protection

This refers to providing data origin authentication and protection against unauthorized modification of information for EAP packets (including EAP Requests and Responses). When making this claim, a

method specification MUST describe the EAP packets and fields within the EAP packet that are protected.

#### Replay protection

This refers to protection against replay of an EAP method or its messages, including success and failure result indications.

#### Confidentiality

This refers to encryption of EAP messages, including EAP Requests and Responses, and success and failure result indications. A method making this claim MUST support identity protection (see Section 7.3).

#### Key derivation

This refers to the ability of the EAP method to derive exportable keying material, such as the Master Session Key (MSK), and Extended Master Session Key (EMSK). The MSK is used only for further key derivation, not directly for protection of the EAP conversation or subsequent data. Use of the EMSK is reserved.

#### Key strength

If the effective key strength is N bits, the best currently known methods to recover the key (with non-negligible probability) require, on average, an effort comparable to  $2^{(N-1)}$  operations of a typical block cipher.

#### Dictionary attack resistance

Where password authentication is used, passwords are commonly selected from a small set (as compared to a set of N-bit keys), which raises a concern about dictionary attacks. A method may be said to provide protection against dictionary attacks if, when it uses a password as a secret, the method does not allow an offline attack that has a work factor based on the number of passwords in an attacker's dictionary.

#### Fast reconnect

The ability, in the case where a security association has been previously established, to create a new or refreshed security association more efficiently or in a smaller number of round-trips.

#### Cryptographic binding

The demonstration of the EAP peer to the EAP server that a single entity has acted as the EAP peer for all methods executed within a tunnel method. Binding MAY also imply that the EAP server demonstrates to the peer that a single entity has acted as the EAP server for all methods executed within a tunnel method. If executed correctly, binding serves to mitigate man-in-the-middle vulnerabilities.

#### Session independence

The demonstration that passive attacks (such as capture of the EAP conversation) or active attacks (including compromise of the MSK or EMSK) does not enable compromise of subsequent or prior MSKs or EMSKs.

#### Fragmentation

This refers to whether an EAP method supports fragmentation and reassembly. As noted in Section 3.1, EAP methods should support fragmentation and reassembly if EAP packets can exceed the minimum MTU of 1020 octets.

#### Channel binding

The communication within an EAP method of integrity-protected channel properties such as endpoint identifiers which can be compared to values communicated via out of band mechanisms (such as via a AAA or lower layer protocol).

#### Perfect Forward Secrecy

The demonstration that the derived keying material, such as the MSK and EMSK will not be compromised even if long-term secrets used in EAP conversation are compromised.

Note: This list of security claims is not exhaustive. Additional properties, such as additional denial-of-service protection, may be relevant as well.

### 7.3. Identity Protection

An Identity exchange is optional within the EAP conversation. Therefore, it is possible to omit the Identity exchange entirely, or to use a method-specific identity exchange once a protected channel has been established.

However, where roaming is supported as described in [RFC2607], it may be necessary to locate the appropriate backend authentication server



before the authentication conversation can proceed. The realm portion of the Network Access Identifier (NAI) [RFC2486] is typically included within the EAP-Response/Identity in order to enable the authentication exchange to be routed to the appropriate backend authentication server. Therefore, while the peer-name portion of the NAI SHOULD be omitted in the EAP-Response/Identity where proxies or relays are present, the realm portion may be required.

It is possible for the identity in the identity response to be different from the identity authenticated by the EAP method. This may be intentional in the case of identity privacy. An EAP method SHOULD use the authenticated identity when making access control decisions.

#### 7.4. Man-in-the-Middle Attacks

Where EAP is tunneled within another protocol that omits peer authentication, there exists a potential vulnerability to a man-in-the-middle attack. For details, see [I-D.puthenkulam-eap-binding] and [MITM].

As noted in Section 2.1, EAP does not permit untunneled sequences of authentication methods. Were a sequence of EAP authentication methods to be permitted, the peer might not have proof that a single entity has acted as the authenticator for all EAP methods within the sequence. For example, an authenticator might terminate one EAP method, then forward the next method in the sequence to another party without the peer's knowledge or consent. Similarly, the authenticator might not have proof that a single entity has acted as the peer for all EAP methods within the sequence.

Tunneling EAP within another protocol enables an attack by a rogue EAP authenticator tunneling EAP to a legitimate server. Where the tunneling protocol is used for key establishment but does not require peer authentication, an attacker convincing a legitimate peer to connect to it will be able to tunnel EAP packets to a legitimate server, successfully authenticating and obtaining the key. This allows the attacker to successfully establish itself as a man-in-the-middle, gaining access to the network, as well as the ability to decrypt data traffic between the legitimate peer and server.

This attack may be mitigated by the following measures:

1. Requiring mutual authentication within EAP tunneling mechanisms.
2. Requiring cryptographic binding between the EAP tunneling protocol and the tunneled EAP methods. Where cryptographic binding is supported, a mechanism is also needed to protect

against downgrade attacks that would bypass it. For further details on cryptographic binding, see [I-D.puthenkulam-eap-binding].

3. Limiting the EAP methods authorized for use without protection, based on peer and authenticator policy.
4. Avoiding the use of tunnels when a single, strong method is available.

#### 7.5. Packet Modification Attacks

While EAP methods may support per-packet data origin authentication, integrity, and replay protection, support is not provided within the EAP layer.

Since the Identifier is only a single octet, it is easy to guess, allowing an attacker to successfully inject or replay EAP packets. An attacker may also modify EAP headers (Code, Identifier, Length, Type) within EAP packets where the header is unprotected. This could cause packets to be inappropriately discarded or misinterpreted.

To protect EAP packets against modification, spoofing, or replay, methods supporting protected ciphersuite negotiation, mutual authentication, and key derivation, as well as integrity and replay protection, are recommended. See Section 7.2.1 for definitions of these security claims.

Method-specific MICs may be used to provide protection. If a per-packet MIC is employed within an EAP method, then peers, authentication servers, and authenticators not operating in pass-through mode MUST validate the MIC. MIC validation failures SHOULD be logged. Whether a MIC validation failure is considered a fatal error or not is determined by the EAP method specification.

It is RECOMMENDED that methods providing integrity protection of EAP packets include coverage of all the EAP header fields, including the Code, Identifier, Length, Type, and Type-Data fields.

Since EAP messages of Types Identity, Notification, and Nak do not include their own MIC, it may be desirable for the EAP method MIC to cover information contained within these messages, as well as the header of each EAP message.

To provide protection, EAP also may be encapsulated within a protected channel created by protocols such as ISAKMP [RFC2408], as is done in [RFC7296] or within TLS [RFC2246]. However, as noted in

Section 7.4, EAP tunneling may result in a man-in-the-middle vulnerability.

Existing EAP methods define message integrity checks (MICs) that cover more than one EAP packet. For example, EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13] defines a MIC over a TLS record that could be split into multiple fragments; within the FINISHED message, the MIC is computed over previous messages. Where the MIC covers more than one EAP packet, a MIC validation failure is typically considered a fatal error.

Within EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13], a MIC validation failure is treated as a fatal error, since that is what is specified in TLS [RFC2246]. However, it is also possible to develop EAP methods that support per-packet MICs, and respond to verification failures by silently discarding the offending packet.

In this document, descriptions of EAP message handling assume that per-packet MIC validation, where it occurs, is effectively performed as though it occurs before sending any responses or changing the state of the host which received the packet.

#### 7.6. Dictionary Attacks

Password authentication algorithms such as EAP-MD5, MS-CHAPv1 [RFC2433], and Kerberos V [RFC1510] are known to be vulnerable to dictionary attacks. MS-CHAPv1 vulnerabilities are documented in [PPTPv1]; MS-CHAPv2 vulnerabilities are documented in [PPTPv2]; Kerberos vulnerabilities are described in [KRBATTACK], [KRBLIM], and [KERB4WEAK].

In order to protect against dictionary attacks, authentication methods resistant to dictionary attacks (as defined in Section 7.2.1) are recommended.

If an authentication algorithm is used that is known to be vulnerable to dictionary attacks, then the conversation may be tunneled within a protected channel in order to provide additional protection. However, as noted in Section 7.4, EAP tunneling may result in a man-in-the-middle vulnerability, and therefore dictionary attack resistant methods are preferred.

#### 7.7. Connection to an Untrusted Network

With EAP methods supporting one-way authentication, such as EAP-MD5, the peer does not authenticate the authenticator, making the peer vulnerable to attack by a rogue authenticator. Methods supporting

mutual authentication (as defined in Section 7.2.1) address this vulnerability.

In EAP there is no requirement that authentication be full duplex or that the same protocol be used in both directions. It is perfectly acceptable for different protocols to be used in each direction. This will, of course, depend on the specific protocols negotiated. However, in general, completing a single unitary mutual authentication is preferable to two one-way authentications, one in each direction. This is because separate authentications that are not bound cryptographically so as to demonstrate they are part of the same session are subject to man-in-the-middle attacks, as discussed in Section 7.4.

#### 7.8. Negotiation Attacks

In a negotiation attack, the attacker attempts to convince the peer and authenticator to negotiate a less secure EAP method. EAP does not provide protection for Nak Response packets, although it is possible for a method to include coverage of Nak Responses within a method-specific MIC.

Within or associated with each authenticator, it is not anticipated that a particular named peer will support a choice of methods. This would make the peer vulnerable to attacks that negotiate the least secure method from among a set. Instead, for each named peer, there SHOULD be an indication of exactly one method used to authenticate that peer name. If a peer needs to make use of different authentication methods under different circumstances, then distinct identities SHOULD be employed, each of which identifies exactly one authentication method.

#### 7.9. Implementation Idiosyncrasies

The interaction of EAP with lower layers such as PPP and IEEE 802 are highly implementation dependent.

For example, upon failure of authentication, some PPP implementations do not terminate the link, instead limiting traffic in Network-Layer Protocols to a filtered subset, which in turn allows the peer the opportunity to update secrets or send mail to the network administrator indicating a problem. Similarly, while an authentication failure will result in denied access to the controlled port in [IEEE-802.1X], limited traffic may be permitted on the uncontrolled port.

In EAP there is no provision for retries of failed authentication. However, in PPP the LCP state machine can renegotiate the

authentication protocol at any time, thus allowing a new attempt. Similarly, in IEEE 802.1X the Supplicant or Authenticator can re-authenticate at any time. It is recommended that any counters used for authentication failure not be reset until after successful authentication, or subsequent termination of the failed link.

#### 7.10. Key Derivation

It is possible for the peer and EAP server to mutually authenticate and derive keys. In order to provide keying material for use in a subsequently negotiated ciphersuite, an EAP method supporting key derivation MUST export a Master Session Key (MSK) of at least 64 octets, and an Extended Master Session Key (EMSK) of at least 64 octets. EAP Methods deriving keys MUST provide for mutual authentication between the EAP peer and the EAP Server.

The MSK and EMSK MUST NOT be used directly to protect data; however, they are of sufficient size to enable derivation of a AAA-Key subsequently used to derive Transient Session Keys (TSKs) for use with the selected ciphersuite. Each ciphersuite is responsible for specifying how to derive the TSKs from the AAA-Key.

The AAA-Key is derived from the keying material exported by the EAP method (MSK and EMSK). This derivation occurs on the AAA server. In many existing protocols that use EAP, the AAA-Key and MSK are equivalent, but more complicated mechanisms are possible (see [RFC5247] for details).

EAP methods SHOULD ensure the freshness of the MSK and EMSK, even in cases where one party may not have a high quality random number generator. A RECOMMENDED method is for each party to provide a nonce of at least 128 bits, used in the derivation of the MSK and EMSK.

EAP methods export the MSK and EMSK, but not Transient Session Keys so as to allow EAP methods to be ciphersuite and media independent. Keying material exported by EAP methods MUST be independent of the ciphersuite negotiated to protect data.

Depending on the lower layer, EAP methods may run before or after ciphersuite negotiation, so that the selected ciphersuite may not be known to the EAP method. By providing keying material usable with any ciphersuite, EAP methods can be used with a wide range of ciphersuites and media.

In order to preserve algorithm independence, EAP methods deriving keys SHOULD support (and document) the protected negotiation of the ciphersuite used to protect the EAP conversation between the peer and

server. This is distinct from the ciphersuite negotiated between the peer and authenticator, used to protect data.

The strength of Transient Session Keys (TSKs) used to protect data is ultimately dependent on the strength of keys generated by the EAP method. If an EAP method cannot produce keying material of sufficient strength, then the TSKs may be subject to a brute force attack. In order to enable deployments requiring strong keys, EAP methods supporting key derivation SHOULD be capable of generating an MSK and EMSK, each with an effective key strength of at least 128 bits.

Methods supporting key derivation MUST demonstrate cryptographic separation between the MSK and EMSK branches of the EAP key hierarchy. Without violating a fundamental cryptographic assumption (such as the non-invertibility of a one-way function), an attacker recovering the MSK or EMSK MUST NOT be able to recover the other quantity with a level of effort less than brute force.

Non-overlapping substrings of the MSK MUST be cryptographically separate from each other, as defined in Section 7.2.1. That is, knowledge of one substring MUST NOT help in recovering some other substring without breaking some hard cryptographic assumption. This is required because some existing ciphersuites form TSKs by simply splitting the AAA-Key to pieces of appropriate length. Likewise, non-overlapping substrings of the EMSK MUST be cryptographically separate from each other, and from substrings of the MSK.

The EMSK is reserved for future use and MUST remain on the EAP peer and EAP server where it is derived; it MUST NOT be transported to, or shared with, additional parties, or used to derive any other keys. (This restriction will be relaxed in a future document that specifies how the EMSK can be used.)

Since EAP does not provide for explicit key lifetime negotiation, EAP peers, authenticators, and authentication servers MUST be prepared for situations in which one of the parties discards the key state, which remains valid on another party.

This specification does not provide detailed guidance on how EAP methods derive the MSK and EMSK, how the AAA-Key is derived from the MSK and/or EMSK, or how the TSKs are derived from the AAA-Key.

The development and validation of key derivation algorithms is difficult, and as a result, EAP methods SHOULD re-use well established and analyzed mechanisms for key derivation (such as those specified in IKE [RFC2409] or TLS [RFC2246]), rather than inventing new ones. EAP methods SHOULD also utilize well established and

analyzed mechanisms for MSK and EMSK derivation. Further details on EAP Key Derivation are provided within [RFC5247].

#### 7.11. Weak Ciphersuites

If after the initial EAP authentication, data packets are sent without per-packet authentication, integrity, and replay protection, an attacker with access to the media can inject packets, "flip bits" within existing packets, replay packets, or even hijack the session completely. Without per-packet confidentiality, it is possible to snoop data packets.

To protect against data modification, spoofing, or snooping, it is recommended that EAP methods supporting mutual authentication and key derivation (as defined by Section 7.2.1) be used, along with lower layers providing per-packet confidentiality, authentication, integrity, and replay protection.

Additionally, if the lower layer performs ciphersuite negotiation, it should be understood that EAP does not provide by itself integrity protection of that negotiation. Therefore, in order to avoid downgrading attacks which would lead to weaker ciphersuites being used, clients implementing lower layer ciphersuite negotiation SHOULD protect against negotiation downgrading.

This can be done by enabling users to configure which ciphersuites are acceptable as a matter of security policy, or the ciphersuite negotiation MAY be authenticated using keying material derived from the EAP authentication and a MIC algorithm agreed upon in advance by lower-layer peers.

##### 7.11.1. Legacy Authentication Methods

EAP has a long history, and the early authentication methods have severe issues. For instance, the MD5-Challenge method uses an algorithm that has problems described in [RFC6151]. These problems are particularly pressing, given that MD5-Challenge does not employ a HMAC construction. The use of MD5-Challenge is NOT RECOMMENDED, at least not outside an external, tunneled authentication method.

Users and network administrators must be aware of the security issues in the authentication methods they choose to allow and use. Modern use of EAP employs typically newer authentication methods such as Transport Layer Security (EAP-TLS) [I-D.ietf-emu-eap-tls13], Tunnel Extensible Authentication Protocol (TEAP) [RFC7170], or 3rd Generation Authentication and Key Agreement (EAP-AKA') [I-D.ietf-emu-rfc5448bis].

## 7.12. Link Layer

There are reliability and security issues with link layer indications in PPP, IEEE 802 LANs, and IEEE 802.11 wireless LANs:

1. PPP. In PPP, link layer indications such as LCP-Terminate (a link failure indication) and NCP (a link success indication) are not authenticated or integrity protected. They can therefore be spoofed by an attacker with access to the link.
2. IEEE 802. IEEE 802.1X EAPOL-Start and EAPOL-Logoff frames are not authenticated or integrity protected. They can therefore be spoofed by an attacker with access to the link.
3. IEEE 802.11. In IEEE 802.11, link layer indications include Disassociate and Deauthenticate frames (link failure indications), and the first message of the 4-way handshake (link success indication). These messages are not authenticated or integrity protected, and although they are not forwardable, they are spoofable by an attacker within range.

In IEEE 802.11, IEEE 802.1X data frames may be sent as Class 3 unicast data frames, and are therefore forwardable. This implies that while EAPOL-Start and EAPOL-Logoff messages may be authenticated and integrity protected, they can be spoofed by an authenticated attacker far from the target when "pre-authentication" is enabled.

In IEEE 802.11, a "link down" indication is an unreliable indication of link failure, since wireless signal strength can come and go and may be influenced by radio frequency interference generated by an attacker. To avoid unnecessary resets, it is advisable to damp these indications, rather than passing them directly to the EAP. Since EAP supports retransmission, it is robust against transient connectivity losses.

## 7.13. Separation of Authenticator and Backend Authentication Server

It is possible for the EAP peer and EAP server to mutually authenticate and derive a AAA-Key for a ciphersuite used to protect subsequent data traffic. This does not present an issue on the peer, since the peer and EAP client reside on the same machine; all that is required is for the client to derive the AAA-Key from the MSK and EMSK exported by the EAP method, and to subsequently pass a Transient Session Key (TSK) to the ciphersuite module.

However, in the case where the authenticator and authentication server reside on different machines, there are several implications for security.



1. Authentication will occur between the peer and the authentication server, not between the peer and the authenticator. This means that it is not possible for the peer to validate the identity of the authenticator that it is speaking to, using EAP alone.
2. As discussed in [RFC3579], the authenticator is dependent on the AAA protocol in order to know the outcome of an authentication conversation, and does not look at the encapsulated EAP packet (if one is present) to determine the outcome. In practice, this implies that the AAA protocol spoken between the authenticator and authentication server MUST support per-packet authentication, integrity, and replay protection.
3. After completion of the EAP conversation, where lower layer security services such as per-packet confidentiality, authentication, integrity, and replay protection will be enabled, a secure association protocol SHOULD be run between the peer and authenticator in order to provide mutual authentication between the peer and authenticator, guarantee liveness of transient session keys, provide protected ciphersuite and capabilities negotiation for subsequent data, and synchronize key usage.
4. A AAA-Key derived from the MSK and/or EMSK negotiated between the peer and authentication server MAY be transmitted to the authenticator. Therefore, a mechanism needs to be provided to transmit the AAA-Key from the authentication server to the authenticator that needs it. The specification of the AAA-key derivation, transport, and wrapping mechanisms is outside the scope of this document. Further details on AAA-Key Derivation are provided within [RFC5247].

#### 7.14. Cleartext Passwords

This specification does not define a mechanism for cleartext password authentication. The omission is intentional. Use of cleartext passwords would allow the password to be captured by an attacker with access to a link over which EAP packets are transmitted.

Since protocols encapsulating EAP, such as RADIUS [RFC3579], may not provide confidentiality, EAP packets may be subsequently encapsulated for transport over the Internet where they may be captured by an attacker.

As a result, cleartext passwords cannot be securely used within EAP, except where encapsulated within a protected tunnel with server authentication. Some of the same risks apply to EAP methods without dictionary attack resistance, as defined in Section 7.2.1. For details, see Section 7.6.

### 7.15. Channel Binding

It is possible for a compromised or poorly implemented EAP authenticator to communicate incorrect information to the EAP peer and/or server. This may enable an authenticator to impersonate another authenticator or communicate incorrect information via out-of-band mechanisms (such as via a AAA or lower layer protocol).

Where EAP is used in pass-through mode, the EAP peer typically does not verify the identity of the pass-through authenticator, it only verifies that the pass-through authenticator is trusted by the EAP server. This creates a potential security vulnerability.

Section 4.3.7 of [RFC3579] describes how an EAP pass-through authenticator acting as a AAA client can be detected if it attempts to impersonate another authenticator (such by sending incorrect NAS-Identifier [RFC2865], NAS-IP-Address [RFC2865] or NAS-IPv6-Address [RFC3162] attributes via the AAA protocol). However, it is possible for a pass-through authenticator acting as a AAA client to provide correct information to the AAA server while communicating misleading information to the EAP peer via a lower layer protocol.

For example, it is possible for a compromised authenticator to utilize another authenticator's Called-Station-Id or NAS-Identifier in communicating with the EAP peer via a lower layer protocol, or for a pass-through authenticator acting as a AAA client to provide an incorrect peer Calling-Station-Id [RFC2865][RFC3580] to the AAA server via the AAA protocol.

In order to address this vulnerability, EAP methods may support a protected exchange of channel properties such as endpoint identifiers, including (but not limited to): Called-Station-Id [RFC2865][RFC3580], Calling-Station-Id [RFC2865][RFC3580], NAS-Identifier [RFC2865], NAS-IP-Address [RFC2865], and NAS-IPv6-Address [RFC3162].

Using such a protected exchange, it is possible to match the channel properties provided by the authenticator via out-of-band mechanisms against those exchanged within the EAP method. Where discrepancies are found, these SHOULD be logged; additional actions MAY also be taken, such as denying access.

### 7.16. Protected Result Indications

Within EAP, Success and Failure packets are neither acknowledged nor integrity protected. Result indications improve resilience to loss of Success and Failure packets when EAP is run over lower layers which do not support retransmission or synchronization of the

authentication state. In media such as IEEE 802.11, which provides for retransmission, as well as synchronization of authentication state via the 4-way handshake defined in [IEEE-802.11i], additional resilience is typically of marginal benefit.

Depending on the method and circumstances, result indications can be spoofable by an attacker. A method is said to provide protected result indications if it supports result indications, as well as the "integrity protection" and "replay protection" claims. A method supporting protected result indications MUST indicate which result indications are protected, and which are not.

Protected result indications are not required to protect against rogue authenticators. Within a mutually authenticating method, requiring that the server authenticate to the peer before the peer will accept a Success packet prevents an attacker from acting as a rogue authenticator.

However, it is possible for an attacker to forge a Success packet after the server has authenticated to the peer, but before the peer has authenticated to the server. If the peer were to accept the forged Success packet and attempt to access the network when it had not yet successfully authenticated to the server, a denial of service attack could be mounted against the peer. After such an attack, if the lower layer supports failure indications, the authenticator can synchronize state with the peer by providing a lower layer failure indication. See Section 7.12 for details.

If a server were to authenticate the peer and send a Success packet prior to determining whether the peer has authenticated the authenticator, an idle timeout can occur if the authenticator is not authenticated by the peer. Where supported by the lower layer, an authenticator sensing the absence of the peer can free resources.

In a method supporting result indications, a peer that has authenticated the server does not consider the authentication successful until it receives an indication that the server successfully authenticated it. Similarly, a server that has successfully authenticated the peer does not consider the authentication successful until it receives an indication that the peer has authenticated the server.

In order to avoid synchronization problems, prior to sending a success result indication, it is desirable for the sender to verify that sufficient authorization exists for granting access, though, as discussed below, this is not always possible.

While result indications may enable synchronization of the authentication result between the peer and server, this does not guarantee that the peer and authenticator will be synchronized in terms of their authorization or that timeouts will not occur. For example, the EAP server may not be aware of an authorization decision made by a AAA proxy; the AAA server may check authorization only after authentication has completed successfully, to discover that authorization cannot be granted, or the AAA server may grant access but the authenticator may be unable to provide it due to a temporary lack of resources. In these situations, synchronization may only be achieved via lower layer result indications.

Success indications may be explicit or implicit. For example, where a method supports error messages, an implicit success indication may be defined as the reception of a specific message without a preceding error message. Failures are typically indicated explicitly. As described in Section 4.2, a peer silently discards a Failure packet received at a point where the method does not explicitly permit this to be sent. For example, a method providing its own error messages might require the peer to receive an error message prior to accepting a Failure packet.

Per-packet authentication, integrity, and replay protection of result indications protects against spoofing. Since protected result indications require use of a key for per-packet authentication and integrity protection, methods supporting protected result indications MUST also support the "key derivation", "mutual authentication", "integrity protection", and "replay protection" claims.

Protected result indications address some denial-of-service vulnerabilities due to spoofing of Success and Failure packets, though not all. EAP methods can typically provide protected result indications only in some circumstances. For example, errors can occur prior to key derivation, and so it may not be possible to protect all failure indications. It is also possible that result indications may not be supported in both directions or that synchronization may not be achieved in all modes of operation.

For example, within EAP-TLS [RFC5216][I-D.ietf-emu-eap-tls13], in the client authentication handshake, the server authenticates the peer, but does not receive a protected indication of whether the peer has authenticated it. In contrast, the peer authenticates the server and is aware of whether the server has authenticated it. In the session resumption handshake, the peer authenticates the server, but does not receive a protected indication of whether the server has authenticated it. In this mode, the server authenticates the peer and is aware of whether the peer has authenticated it.

## 8. References

### 8.1. Normative References

- [RFC1661] Simpson, W., Ed., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, DOI 10.17487/RFC1661, July 1994, <<https://www.rfc-editor.org/info/rfc1661>>.
- [RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, DOI 10.17487/RFC1994, August 1996, <<https://www.rfc-editor.org/info/rfc1994>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2243] Metz, C., "OTP Extended Responses", RFC 2243, DOI 10.17487/RFC2243, November 1997, <<https://www.rfc-editor.org/info/rfc2243>>.
- [RFC2289] Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time Password System", STD 61, RFC 2289, DOI 10.17487/RFC2289, February 1998, <<https://www.rfc-editor.org/info/rfc2289>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [IEEE-802] Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Overview and Architecture", IEEE Standard 802, 1990.

[IEEE-802.1X]

Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X, January 2020.

[TS.33.501]

3GPP, "Security architecture and procedures for 5G System", 3GPP TS 33.501 17.0.0, December 2020.

## 8.2. Informative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1510] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, DOI 10.17487/RFC1510, September 1993, <<https://www.rfc-editor.org/info/rfc1510>>.
- [RFC1750] Eastlake 3rd, D., Crocker, S., and J. Schiller, "Randomness Recommendations for Security", RFC 1750, DOI 10.17487/RFC1750, December 1994, <<https://www.rfc-editor.org/info/rfc1750>>.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <<https://www.rfc-editor.org/info/rfc2246>>.
- [RFC2284] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, DOI 10.17487/RFC2284, March 1998, <<https://www.rfc-editor.org/info/rfc2284>>.
- [RFC2408] Maughan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, DOI 10.17487/RFC2408, November 1998, <<https://www.rfc-editor.org/info/rfc2408>>.
- [RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, DOI 10.17487/RFC2409, November 1998, <<https://www.rfc-editor.org/info/rfc2409>>.
- [RFC2433] Zorn, G. and S. Cobb, "Microsoft PPP CHAP Extensions", RFC 2433, DOI 10.17487/RFC2433, October 1998, <<https://www.rfc-editor.org/info/rfc2433>>.

- [RFC2486] Aboba, B. and M. Beadles, "The Network Access Identifier", RFC 2486, DOI 10.17487/RFC2486, January 1999, <<https://www.rfc-editor.org/info/rfc2486>>.
- [RFC2607] Aboba, B. and J. Vollbrecht, "Proxy Chaining and Policy Implementation in Roaming", RFC 2607, DOI 10.17487/RFC2607, June 1999, <<https://www.rfc-editor.org/info/rfc2607>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <<https://www.rfc-editor.org/info/rfc2865>>.
- [RFC3162] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, DOI 10.17487/RFC3162, August 2001, <<https://www.rfc-editor.org/info/rfc3162>>.
- [RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, DOI 10.17487/RFC3454, December 2002, <<https://www.rfc-editor.org/info/rfc3454>>.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <<https://www.rfc-editor.org/info/rfc3579>>.
- [RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G., and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, DOI 10.17487/RFC3580, September 2003, <<https://www.rfc-editor.org/info/rfc3580>>.
- [RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, DOI 10.17487/RFC3692, January 2004, <<https://www.rfc-editor.org/info/rfc3692>>.

- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC4072] Eronen, P., Ed., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, DOI 10.17487/RFC4072, August 2005, <<https://www.rfc-editor.org/info/rfc4072>>.
- [RFC4137] Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", RFC 4137, DOI 10.17487/RFC4137, August 2005, <<https://www.rfc-editor.org/info/rfc4137>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5113] Arkko, J., Aboba, B., Korhonen, J., Ed., and F. Bari, "Network Discovery and Selection Problem", RFC 5113, DOI 10.17487/RFC5113, January 2008, <<https://www.rfc-editor.org/info/rfc5113>>.
- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <<https://www.rfc-editor.org/info/rfc5216>>.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <<https://www.rfc-editor.org/info/rfc5247>>.
- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, DOI 10.17487/RFC4013, February 2005, <<https://www.rfc-editor.org/info/rfc4013>>.
- [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <<https://www.rfc-editor.org/info/rfc6151>>.
- [RFC6677] Hartman, S., Ed., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, DOI 10.17487/RFC6677, July 2012, <<https://www.rfc-editor.org/info/rfc6677>>.



- [RFC6696] Cao, Z., He, B., Shi, Y., Wu, Q., Ed., and G. Zorn, Ed., "EAP Extensions for the EAP Re-authentication Protocol (ERP)", RFC 6696, DOI 10.17487/RFC6696, July 2012, <<https://www.rfc-editor.org/info/rfc6696>>.
- [RFC7029] Hartman, S., Wasserman, M., and D. Zhang, "Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding", RFC 7029, DOI 10.17487/RFC7029, October 2013, <<https://www.rfc-editor.org/info/rfc7029>>.
- [RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <<https://www.rfc-editor.org/info/rfc7170>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC7613] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 7613, DOI 10.17487/RFC7613, August 2015, <<https://www.rfc-editor.org/info/rfc7613>>.
- [RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <<https://www.rfc-editor.org/info/rfc8265>>.
- [DECEPTION] Slatalla, M. and J. Quittner, "Masters of Deception", Harper-Collins New York, 1995.
- [KRBATTACK] Wu, T., "A Real-World Analysis of Kerberos Password Security", Proceedings of the 1999 ISOC Network and Distributed System Security Symposium <http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/wu.pdf>.
- [KRBLIM] Bellovin, S. and M. Merrit, "Limitations of the Kerberos authentication system", Proceedings of the 1991 Winter USENIX Conference pp. 253-267, 1991.

## [KERB4WEAK]

Dole, B., Lodin, S., and E. Spafford, "Misplaced trust: Kerberos 4 session keys", Proceedings of the Internet Society Network and Distributed System Security Symposium pp. 60-70, March 1997.

## [I-D.ietf-ipsra-pic]

Sheffer, Y., Krawczyk, H., and B. Aboba, "PIC, A Pre-IKE Credential Provisioning Protocol", draft-ietf-ipsra-pic-05 (work in progress), February 2002.

## [PPTPv1]

Schneier, B. and Mudge, "Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol", Proceedings of the 5th ACM Conference on Communications and Computer Security ACM Press, November 1998.

## [IEEE-802.11]

Institute of Electrical and Electronics Engineers, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11, 1999.

## [SILVERMAN]

Silverman, R., "A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths", RSA Laboratories Bulletin 13 (Revised November 2001)  
<http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html>, April 2000.

## [IEEE-802.11i]

Institute of Electrical and Electronics Engineers, "Unapproved Draft Supplement to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security", IEEE Draft 802.11i (work in progress), 2003.

## [I-D.zorn-eap-eval]

Zorn, G., "Specifying Security Claims for EAP Authentication Types", draft-zorn-eap-eval-00 (work in progress), October 2002.

## [I-D.puthenkulam-eap-binding]

Puthenkulam, J., "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04 (work in progress), October 2003.

- [MITM] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols", IACR ePrint Archive Report 2002/163 <http://eprint.iacr.org/2002/163>, October 2002.
- [IEEE-802.11i-req] Stanley, D., "EAP Method Requirements for Wireless LANs", February 2004.
- [PPTPv2] Schneier, B. and Mudge, "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)", CQRE 99, Springer-Verlag pp. 192-203, 1999.
- [Terminology] Alissa Cooper et al., , "Inclusive terminology in IETF Documents", Contribution under the IETF GitHub <https://github.com/ietf/terminology>, October 2020.
- [W3C] Le Hegaret, P. and C. Mercier, "W3C Manual of Style", W3C Document <https://w3c.github.io/manual-of-style/>, January 2021.
- [RedHat] Wright, C., "Making open source more inclusive by eradicating problematic language", RedHat Blog <https://www.redhat.com/en/blog/making-open-source-more-inclusive-eradicating-problematic-language>, January 2021.
- [GitLab] Ramsay, J., "Change the default initial branch name for new projects on GitLab", GitLab issue 221164 <https://gitlab.com/gitlab-org/gitlab/-/issues/221164>, June 2020.
- [Mozilla] Davidson, J., "Replace all user-facing instances that refer to "master" password", Mozilla Bug 1644807 [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1644807](https://bugzilla.mozilla.org/show_bug.cgi?id=1644807), November 2016.
- [IESG] IESG, , "IESG Statement On Oppressive or Exclusionary Language", IESG Statement <https://www.ietf.org/about/groups/iesg/statements/statement-on-oppressive-exclusionary-language/>, July 2020.
- [I-D.ietf-emu-eap-tls13] Mattsson, J. and M. Sethi, "Using EAP-TLS with TLS 1.3", draft-ietf-emu-eap-tls13-13 (work in progress), November 2020.

[I-D.ietf-emu-rfc5448bis]

Arkko, J., Lehtovirta, V., Torvinen, V., and P. Eronen,  
"Improved Extensible Authentication Protocol Method for  
3GPP Mobile Network Authentication and Key Agreement (EAP-  
AKA')", draft-ietf-emu-rfc5448bis-09 (work in progress),  
January 2021.

#### Appendix A. Changes from RFC 3748

There are no changes with related to interoperability. Minor changes, including style, grammar, spelling, and editorial changes are not mentioned here. The only changes are the following:

- o The names of the MSK and EMSK terms used to discuss and specify the protocol have been changed.
- o The security considerations note the deficiencies in legacy EAP methods such as MD5-Challenge in Section 7.11.1, and recommend the use of more modern authentication methods.
- o Ivo Sedlacek's errata on a reference to Section 7.12 rather than Section 7.2 from Section 3.4 has been adopted.
- o IANA rules have been updated to comply with RFC 8126 and current allocations.
- o References have been updated to their most recent versions.
- o The security claim perfect forward secrecy has been added.
- o References to 3GPP 5G has been added.
- o The peer-name portion of the NAI SHOULD be omitted in the EAP-Response/Identity.
- o Since the publication of RFC3748, several documents related to the core EAP document have been published: [RFC4137] offers a proposed state machine [RFC5113] defines the network discovery and selection problem, [RFC5247] specifies the EAP key hierarchy, [RFC6677] [RFC7029] explores man-in-the-middle attacks and defines how to implement channel bindings. References to RFC 4137, RFC 5113, RFC 5247, RFC 6677, and RFC 7029 3GPP have been added.

There are still some open questions, however:

- o RFC 3748 referred to an early version of the SASLPREP document, which turned into [RFC4013], then [RFC7613], and is currently

[RFC8265]. Does this still apply? Has something been learned in the meanwhile about internationalization and passwords?

- o Is there a need to update security considerations beyond what was done already? There is likely more to say about privacy, identity protection, pervasive monitoring and perfect forward secrecy.
- o IEEE references need to be updated to newer ones. Some aspects of IEEE have changed since 2004
- o IEEE links are discussed a lot in the document, and some of 3GPP link technologies and related EAP methods. Should the document say something more about 3GPP and 5G?
- o Could some sections be replaced by links to RFC 4137, RFC 5113, RFC 5247, RFC 6677, and RFC 7029? Should the document say more about RFC 4137, RFC 5113, RFC 5247, RFC 6677, and RFC 7029?
- o What other issues have been discussed since 2004, but not recorded in errata?

A summary of the changes between RFC 3748 and RFC 2284 were listed in Appendix A of RFC 3748 [RFC3748] [RFC2284].

## Appendix B. Rationale

In 2020, the Internet Engineering Steering Group (IESG) noted that terminology used in IETF documents is important [IESG]. When the objective of an organization is to be inclusive and respectful, terminology can also have an effect. There are obvious challenges for creating good terminology for the parts of Internet technology currently under development, both in a technical sense and in our ability to agree what terms are inclusive. There are also difficult tradeoffs related to changing terminology for existing technology, or for spending valuable effort on terminology vs. other things.

This update is both a refresh of the RFC in general, bringing in the noted errata, updates to referred documents, but also an update of the terminology.

With regards to terminology, the authors have worked for a long time with EAP technology, and continue to make contributions in this space. In the authors' view, while there is no need for a change, some of the terms that are used when referring to various parts of the overall EAP technology could be improved. As a result, the authors wanted to make a modest proposal for a change that would improve the terms without changing the associated acronyms, and enable better use of the terms in future documents.

It should be noted that the issues with EAP terms are minor, compared many other terminology or other problems with Internet technology. The authors do not wish to start a big debate; if the WG finds this useful, we can perhaps make an update and move on. If not, we can simply move on without making a change.

The specific change that is suggested in this document relates to the use of the word "master" in various EAP terms. This word is rather benign when compared to the use of master/slave or black/whitelists, and other similar terms. Indeed, "master" is commonly used in a large number of everyday terms. Given this, some authors and organizations have chosen to make updates only with the most egregious terms, such as master/slave.

Nevertheless, at least the authors of this document feel that he would use another word if a different word or term was available. It should be noted that:

- o The slavery-related meaning comes up in any dictionary search for the word "master".
- o The word "master" and some suggested alternatives (such as "main") are listed in [Terminology].
- o Several organisations have recommended changing the word "master" in various aspects of their documentation or software. Others are considering changes. See, for instance, [W3C] [RedHat] [GitLab] [Mozilla].

In any case, as noted, this proposal is for the working group to discuss. Discussion may find that the proposal is considered useful, unnecessary, or flawed in some fashion.

#### Appendix C. Acknowledgements

This version of the document is a minor update with respect to RFC 3748. The acknowledgements from RFC 3748 apply:

This protocol derives much of its inspiration from Dave Carrel's AHA document, as well as the PPP CHAP protocol [RFC1994]. Valuable feedback was provided by Yoshihiro Ohba of Toshiba America Research, Jari Arkko of Ericsson, Sachin Seth of Microsoft, Glen Zorn of Cisco Systems, Jesse Walker of Intel, Bill Arbaugh, Nick Petroni and Bryan Payne of the University of Maryland, Steve Bellovin of AT&T Research, Paul Funk of Funk Software, Pasi Eronen of Nokia, Joseph Salowey of Cisco, Paul Congdon of HP, and members of the EAP working group.

The use of Security Claims sections for EAP methods, as required by Section 7.2 and specified for each EAP method described in this document, was inspired by Glen Zorn through [I-D.zorn-eap-eval].

The authors of the most recent version of this document would like to thank Stephen Hayes, Lars Eggert, Mohit Sethi, Alissa Cooper, and Ivo Sedlacek for enlightening discussions and general contributions in this area.

#### Authors' Addresses

Bernard Aboba  
Microsoft Corporation  
USA

Email: bernarda@microsoft.com

Larry J. Blunk  
Merit Network, Inc  
USA

Email: ljb@merit.edu

John R. Vollbrecht  
Vollbrecht Consulting LLC  
USA

Email: jrv@umich.edu

James Carlson  
Sun Microsystems, Inc  
USA

Email: james.d.carlson@sun.com

Henrik Levkowetz  
ipUnplugged AB  
Sweden

Email: henrik@levkowetz.com

Jari Arkko (Editor)  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net

John Mattsson (Editor)  
Ericsson  
Stockholm  
Sweden

Email: john.mattsson@ericsson.com



Network Working Group  
Internet-Draft  
Updates: 5216 (if approved)  
Intended status: Standards Track  
Expires: 23 April 2022

J. Preuß Mattsson  
M. Sethi  
Ericsson  
20 October 2021

Using EAP-TLS with TLS 1.3 (EAP-TLS 1.3)  
draft-ietf-emu-eap-tls13-21

## Abstract

The Extensible Authentication Protocol (EAP), defined in RFC 3748, provides a standard mechanism for support of multiple authentication methods. This document specifies the use of EAP-Transport Layer Security (EAP-TLS) with TLS 1.3 while remaining backwards compatible with existing implementations of EAP-TLS. TLS 1.3 provides significantly improved security and privacy, and reduced latency when compared to earlier versions of TLS. EAP-TLS with TLS 1.3 (EAP-TLS 1.3) further improves security and privacy by always providing forward secrecy, never disclosing the peer identity, and by mandating use of revocation checking, when compared to EAP-TLS with earlier versions of TLS. This document also provides guidance on authentication, authorization, and resumption for EAP-TLS in general (regardless of the underlying TLS version used). This document updates RFC 5216.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 April 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Requirements and Terminology . . . . .	4
2. Protocol Overview . . . . .	5
2.1. Overview of the EAP-TLS Conversation . . . . .	5
2.1.1. Authentication . . . . .	6
2.1.2. Ticket Establishment . . . . .	7
2.1.3. Resumption . . . . .	9
2.1.4. Termination . . . . .	11
2.1.5. No Peer Authentication . . . . .	14
2.1.6. Hello Retry Request . . . . .	15
2.1.7. Identity . . . . .	16
2.1.8. Privacy . . . . .	17
2.1.9. Fragmentation . . . . .	18
2.2. Identity Verification . . . . .	18
2.3. Key Hierarchy . . . . .	19
2.4. Parameter Negotiation and Compliance Requirements . . . . .	20
2.5. EAP State Machines . . . . .	21
3. Detailed Description of the EAP-TLS Protocol . . . . .	22
4. IANA considerations . . . . .	22
5. Security Considerations . . . . .	22
5.1. Security Claims . . . . .	22
5.2. Peer and Server Identities . . . . .	23
5.3. Certificate Validation . . . . .	23
5.4. Certificate Revocation . . . . .	23
5.5. Packet Modification Attacks . . . . .	24
5.6. Authorization . . . . .	25
5.7. Resumption . . . . .	26
5.8. Privacy Considerations . . . . .	28
5.9. Pervasive Monitoring . . . . .	29
5.10. Discovered Vulnerabilities . . . . .	29
5.11. Cross-Protocol Attacks . . . . .	30
6. References . . . . .	30
6.1. Normative References . . . . .	30
6.2. Informative references . . . . .	31
Appendix A. Updated References . . . . .	36
Acknowledgments . . . . .	36
Contributors . . . . .	36

Authors' Addresses	36
--------------------	----

## 1. Introduction

The Extensible Authentication Protocol (EAP), defined in [RFC3748], provides a standard mechanism for support of multiple authentication methods. EAP-Transport Layer Security (EAP-TLS) [RFC5216] specifies an EAP authentication method with certificate-based mutual authentication utilizing the TLS handshake protocol for cryptographic algorithms and protocol version negotiation and establishment of shared secret keying material. EAP-TLS is widely supported for authentication and key establishment in IEEE 802.11 [IEEE-802.11] (Wi-Fi) and IEEE 802.1AE [IEEE-802.1AE] (MACsec) networks using IEEE 802.1X [IEEE-802.1X] and it's the default mechanism for certificate based authentication in 3GPP 5G [TS.33.501] and MulteFire [MulteFire] networks. Many other EAP methods such as EAP-FAST [RFC4851], EAP-TTLS [RFC5281], TEAP [RFC7170], and PEAP [PEAP] depend on TLS and EAP-TLS.

EAP-TLS [RFC5216] references TLS 1.0 [RFC2246] and TLS 1.1 [RFC4346], but can also work with TLS 1.2 [RFC5246]. TLS 1.0 and 1.1 are formally deprecated and prohibited to negotiate and use [RFC8996]. Weaknesses found in TLS 1.2, as well as new requirements for security, privacy, and reduced latency have led to the specification of TLS 1.3 [RFC8446], which obsoletes TLS 1.2 [RFC5246]. TLS 1.3 is in large parts a complete remodeling of the TLS handshake protocol including a different message flow, different handshake messages, different key schedule, different cipher suites, different resumption mechanism, different privacy protection, and different record padding. This means that significant parts of the normative text in the previous EAP-TLS specification [RFC5216] are not applicable to EAP-TLS with TLS 1.3. Therefore, aspects such as resumption, privacy handling, and key derivation need to be appropriately addressed for EAP-TLS with TLS 1.3.

This document updates [RFC5216] to define how to use EAP-TLS with TLS 1.3. When older TLS versions are negotiated, RFC 5216 applies to maintain backwards compatibility. However, this document does provide additional guidance on authentication, authorization, and resumption for EAP-TLS regardless of the underlying TLS version used. This document only describes differences compared to [RFC5216]. When EAP-TLS is used with TLS 1.3, some references are updated as specified in Appendix A. All message flow are example message flows specific to TLS 1.3 and do not apply to TLS 1.2. Since EAP-TLS couples the TLS handshake state machine with the EAP state machine it is possible that new versions of TLS will cause incompatibilities that introduce failures or security issues if they are not carefully integrated into the EAP-TLS protocol. Therefore, implementations MUST limit the maximum TLS version they use to 1.3, unless later versions are explicitly enabled by the administrator.

This document specifies EAP-TLS 1.3 and does not specify how other TLS-based EAP methods use TLS 1.3. The specification for how other TLS-based EAP methods use TLS 1.3 is left to other documents such as [I-D.ietf-emu-tls-eap-types].

In addition to the improved security and privacy offered by TLS 1.3, there are other significant benefits of using EAP-TLS with TLS 1.3. Privacy, which in EAP-TLS means that no information about the underlying peer identity is disclosed, is mandatory and achieved without any additional round-trips. Revocation checking is mandatory and simplified with OCSP stapling, and TLS 1.3 introduces more possibilities to reduce fragmentation when compared to earlier versions of TLS.

### 1.1. Requirements and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts used in EAP-TLS [RFC5216] and TLS [RFC8446]. The term EAP-TLS peer is used for the entity acting as EAP peer and TLS client. The term EAP-TLS server is used for the entity acting as EAP server and TLS server.

This document follows the terminology from [I-D.ietf-tls-rfc8446bis] where the master secret is renamed to the main secret and the exporter\_master\_secret is renamed to the exporter\_secret.

## 2. Protocol Overview

### 2.1. Overview of the EAP-TLS Conversation

This section updates Section 2.1 of [RFC5216] by amending it in accordance with the following discussion.

If the TLS implementation correctly implements TLS version negotiation, EAP-TLS will automatically leverage that capability. The EAP-TLS implementation needs to know which version of TLS was negotiated to correctly support EAP-TLS 1.3 as well as to maintain backward compatibility with EAP-TLS 1.2.

TLS 1.3 changes both the message flow and the handshake messages compared to earlier versions of TLS. Therefore, much of Section 2.1 of [RFC5216] does not apply for TLS 1.3. Except for Sections 2.2 and 5.7, this update applies only when TLS 1.3 is negotiated. When TLS 1.2 is negotiated, then [RFC5216] applies.

TLS 1.3 introduces several new handshake messages including HelloRetryRequest, NewSessionTicket, and KeyUpdate. In general, these messages will be handled by the underlying TLS libraries and are not visible to EAP-TLS, however, there are a few things to note:

- \* The HelloRetryRequest is used by the server to reject the parameters offered in the ClientHello and suggest new parameters. When this message is encountered it will increase the number of round trips used by the protocol.
- \* The NewSessionTicket message is used to convey resumption information and is covered in Sections 2.1.2 and 2.1.3.
- \* The KeyUpdate message is used to update the traffic keys used on a TLS connection. EAP-TLS does not encrypt significant amounts of data so this functionality is not needed. Implementations SHOULD NOT send this message, however some TLS libraries may automatically generate and process this message.
- \* Early Data MUST NOT be used in EAP-TLS. EAP-TLS servers MUST NOT send an early\_data extension and clients MUST NOT send an EndOfEarlyData message.
- \* Post-handshake authentication MUST NOT be used in EAP-TLS. Clients MUST NOT send a "post\_handshake\_auth" extension and Servers MUST NOT request post-handshake client authentication.

After receiving an EAP-Request packet with EAP-Type=EAP-TLS as described in [RFC5216] the conversation will continue with the TLS handshake protocol encapsulated in the data fields of EAP-Response and EAP-Request packets. When EAP-TLS is used with TLS version 1.3, the formatting and processing of the TLS handshake SHALL be done as specified in version 1.3 of TLS. This update only lists additional and different requirements, restrictions, and processing compared to [RFC8446] and [RFC5216].

#### 2.1.1. Authentication

This section updates Section 2.1.1 of [RFC5216] by amending it in accordance with the following discussion.

The EAP-TLS server MUST authenticate with a certificate and SHOULD require the EAP-TLS peer to authenticate with a certificate. Certificates can be of any type supported by TLS including raw public keys. Pre-Shared Key (PSK) authentication SHALL NOT be used except for resumption. The full handshake in EAP-TLS with TLS 1.3 always provides forward secrecy by exchange of ephemeral "key\_share" extensions in the ClientHello and ServerHello (e.g., containing ephemeral ECDHE public keys). SessionID is deprecated in TLS 1.3, see Sections 4.1.2 and 4.1.3 of [RFC8446]. TLS 1.3 introduced early application data which like all application data (other than the protected success indication described below) is not used in EAP-TLS; see Section 4.2.10 of [RFC8446] for additional information on the "early\_data" extension. Resumption is handled as described in Section 2.1.3. As a protected success indication [RFC3748] the EAP-TLS server always sends TLS application data 0x00, see Section 2.5. Note that a TLS implementation MAY not allow the EAP-TLS layer to control in which order things are sent and the application data MAY therefore be sent before a NewSessionTicket. TLS application data 0x00 is therefore to be interpreted as success after the EAP-Request that contains TLS application data 0x00. After the EAP-TLS server has sent an EAP-Request containing the TLS application data 0x00 and received an EAP-Response packet of EAP-Type=EAP-TLS and no data, the EAP-TLS server sends EAP-Success.

Figure 1 shows an example message flow for a successful EAP-TLS full handshake with mutual authentication (and neither HelloRetryRequest nor post-handshake messages are sent).

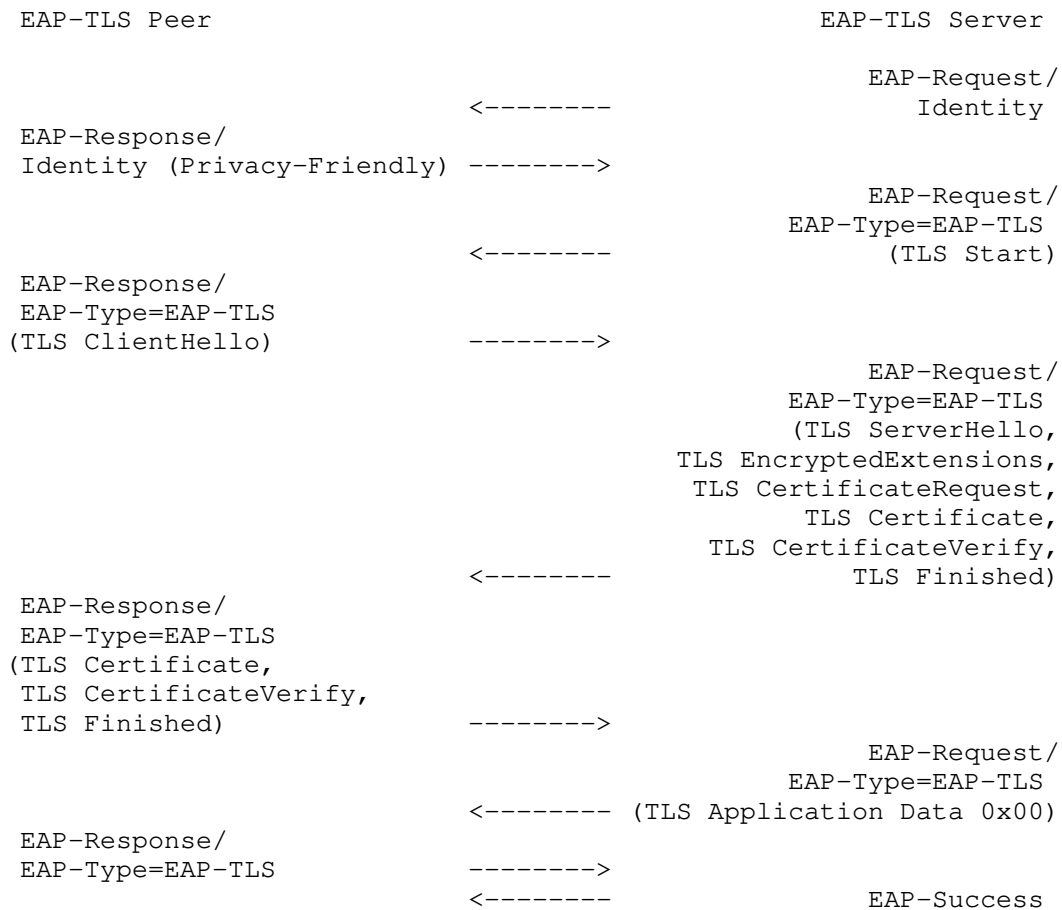


Figure 1: EAP-TLS mutual authentication

#### 2.1.2. Ticket Establishment

This is a new section when compared to [RFC5216].

To enable resumption when using EAP-TLS with TLS 1.3, the EAP-TLS server MUST send one or more post-handshake NewSessionTicket messages (each associated with a PSK, a PSK identity, a ticket lifetime, and other parameters) in the initial authentication. Note that TLS 1.3 [RFC8446] limits the ticket lifetime to a maximum of 604800 seconds (7 days) and EAP-TLS servers MUST respect this upper limit when issuing tickets. The NewSessionTicket is sent after the EAP-TLS server has received the client Finished message in the initial authentication. The NewSessionTicket can be sent in the same flight as the TLS server Finished or later. The PSK associated with the

ticket depends on the client Finished and cannot be pre-computed (so as to be sent in the same flight as the TLS server Finished) in handshakes with client authentication. The NewSessionTicket message MUST NOT include an "early\_data" extension. If the "early\_data" extension is received then it MUST be ignored. Servers should take into account that fewer NewSessionTickets will likely be needed in EAP-TLS than in the usual HTTPS connection scenario. In most cases a single NewSessionTicket will be sufficient. A mechanism by which clients can specify the desired number of tickets needed for future connections is defined in [I-D.ietf-tls-ticketrequests].

Figure 2 shows an example message flow for a successful EAP-TLS full handshake with mutual authentication and ticket establishment of a single ticket.

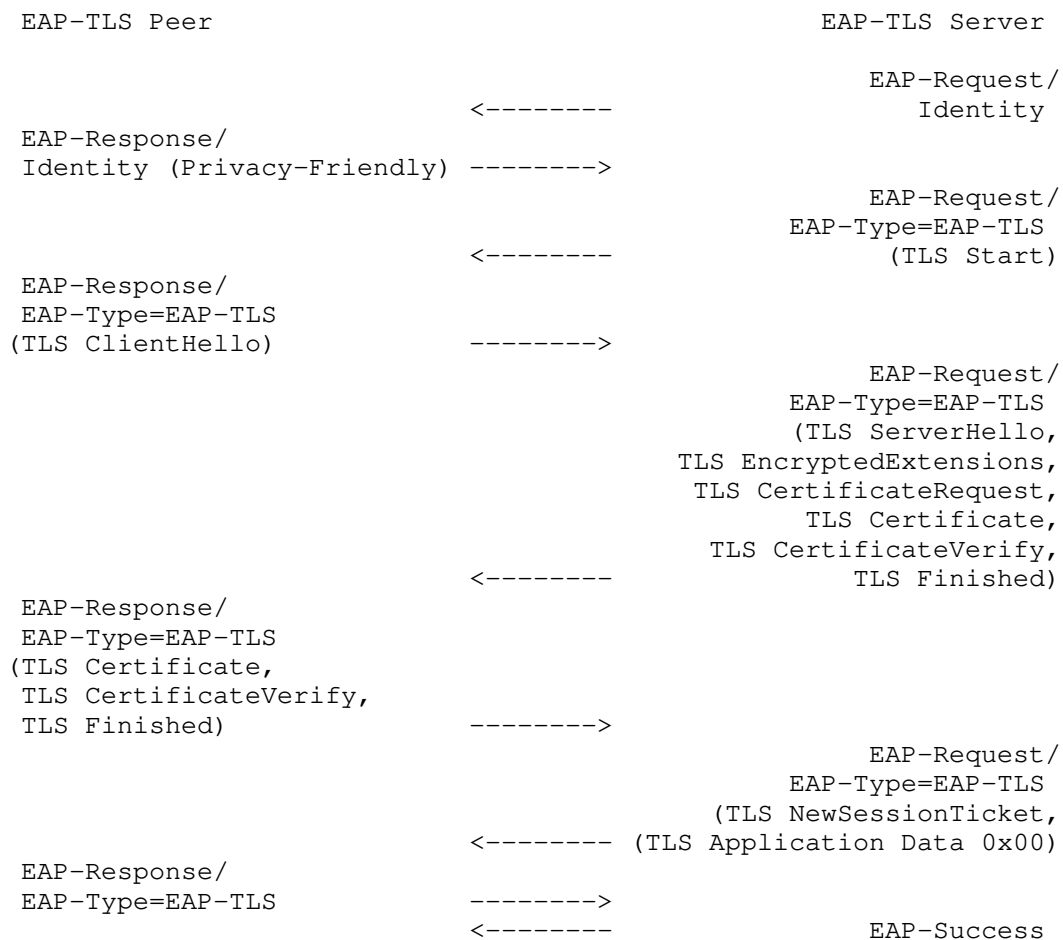




Figure 2: EAP-TLS ticket establishment

### 2.1.3. Resumption

This section updates Section 2.1.2 of [RFC5216] by amending it in accordance with the following discussion.

EAP-TLS is typically used with client authentication and typically fragments the TLS flights into a large number of EAP requests and EAP responses. Resumption significantly reduces the number of round-trips and enables the EAP-TLS server to omit database lookups needed during a full handshake with client authentication. TLS 1.3 replaces the session resumption mechanisms in earlier versions of TLS with a new PSK exchange. When EAP-TLS is used with TLS version 1.3, EAP-TLS SHALL use a resumption mechanism compatible with version 1.3 of TLS.

For TLS 1.3, resumption is described in Section 2.2 of [RFC8446]. If the client has received a NewSessionTicket message from the EAP-TLS server, the client can use the PSK identity associated with the ticket to negotiate the use of the associated PSK. If the EAP-TLS server accepts it, then the resumed session has been deemed to be authenticated, and securely associated with the prior authentication or resumption. It is up to the EAP-TLS peer to use resumption, but it is RECOMMENDED that the EAP-TLS peer use resumption if it has a valid ticket that has not been used before. It is left to the EAP-TLS server whether to accept resumption, but it is RECOMMENDED that the EAP-TLS server accept resumption if the ticket which was issued is still valid. However, the EAP-TLS server MAY choose to require a full handshake. In the case a full handshake is required, the negotiation proceeds as if the session was a new authentication, and the resumption attempt is ignored. The requirements of Sections 2.1.1 and 2.1.2 then apply in their entirety. As described in Appendix C.4 of [RFC8446], reuse of a ticket allows passive observers to correlate different connections. EAP-TLS peers and EAP-TLS servers SHOULD follow the client tracking preventions in Appendix C.4 of [RFC8446].

It is RECOMMENDED to use a Network Access Identifiers (NAIs) with the same realm during resumption and the original full handshake. This requirement allows EAP packets to be routed to the same destination as the original full handshake. If this recommendation is not followed, resumption is likely impossible. When NAI reuse can be done without privacy implications, it is RECOMMENDED to use the same NAI in the resumption, as was used in the original full handshake [RFC7542]. For example, the NAI @realm can safely be reused since it does not provide any specific information to associate a user's resumption attempt with the original full handshake. However, reusing the NAI P2ZIM2F+OEVAO21nNWg2bVpgNnU=@realm enables an on-path

attacker to associate a resumption attempt with the original full handshake. The TLS PSK identity is typically derived by the TLS implementation and may be an opaque blob without a routable realm. The TLS PSK identity on its own is therefore unsuitable as a NAI in the Identity Response.

Figure 3 shows an example message flow for a subsequent successful EAP-TLS resumption handshake where both sides authenticate via a PSK provisioned via an earlier NewSessionTicket and where the server provisions a single new ticket.

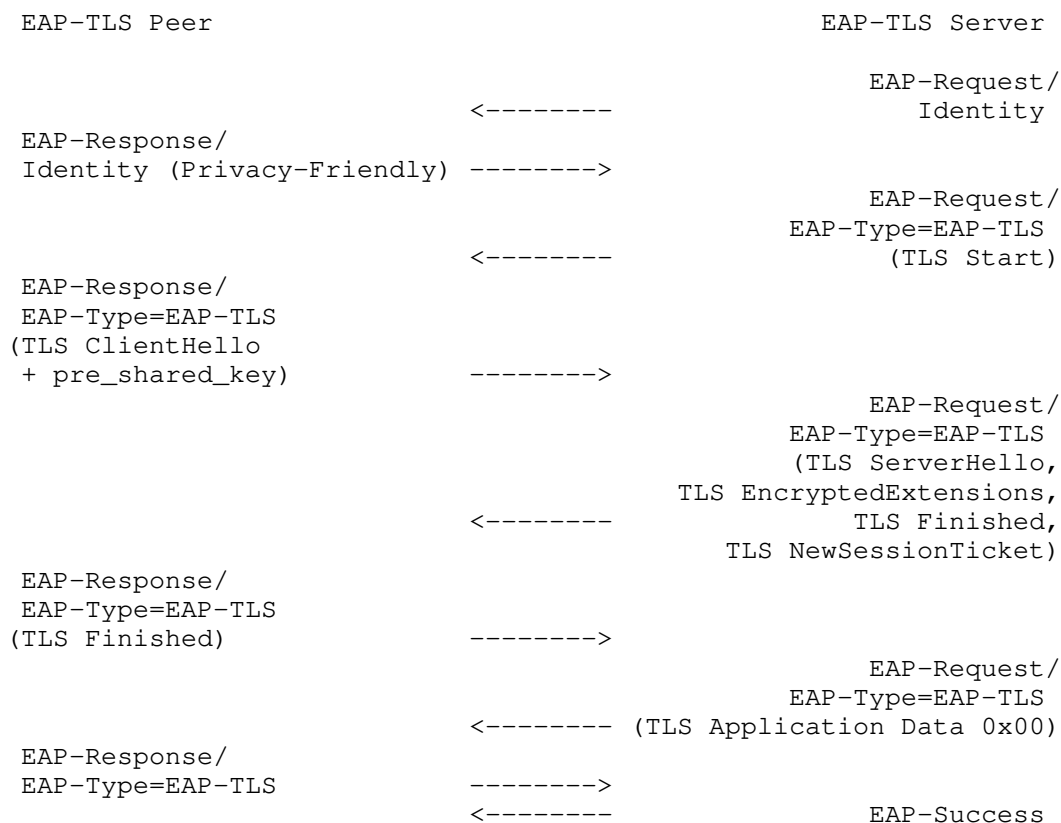


Figure 3: EAP-TLS resumption

As specified in Section 2.2 of [RFC8446], the EAP-TLS peer SHOULD supply a "key\_share" extension when attempting resumption, which allows the EAP-TLS server to potentially decline resumption and fall back to a full handshake. If the EAP-TLS peer did not supply a "key\_share" extension when attempting resumption, the EAP-TLS server needs to send HelloRetryRequest to signal that additional information

is needed to complete the handshake, and the EAP-TLS peer needs to send a second ClientHello containing that information. Providing a "key\_share" and using the "psk\_dhe\_ke" pre-shared key exchange mode is also important in order to limit the impact of a key compromise. When using "psk\_dhe\_ke", TLS 1.3 provides forward secrecy meaning that compromise of the PSK used for resumption does not compromise any earlier connections. The "psk\_dh\_ke" key-exchange mode **MUST** be used for resumption unless the deployment has a local requirement to allow configuration of other mechanisms.

#### 2.1.4. Termination

This section updates Section 2.1.3 of [RFC5216] by amending it in accordance with the following discussion.

TLS 1.3 changes both the message flow and the handshake messages compared to earlier versions of TLS. Therefore, some normative text in Section 2.1.3 of [RFC5216] does not apply for TLS 1.3. The two paragraphs below replace the corresponding paragraphs in Section 2.1.3 of [RFC5216] when EAP-TLS is used with TLS 1.3. The other paragraphs in Section 2.1.3 of [RFC5216] still apply with the exception that SessionID is deprecated.

If the EAP-TLS peer authenticates successfully, the EAP-TLS server **MUST** send an EAP-Request packet with EAP-Type=EAP-TLS containing TLS records conforming to the version of TLS used. The message flow ends with a protected success indication from the EAP-TLS server, followed by an EAP-Response packet of EAP-Type=EAP-TLS and no data from the EAP-TLS peer, followed by EAP-Success from the server.

If the EAP-TLS server authenticates successfully, the EAP-TLS peer **MUST** send an EAP-Response message with EAP-Type=EAP-TLS containing TLS records conforming to the version of TLS used.

Figures 4, 5, and 6 illustrate message flows in several cases where the EAP-TLS peer or EAP-TLS server sends a TLS Error alert message. In earlier versions of TLS, error alerts could be warnings or fatal. In TLS 1.3, error alerts are always fatal and the only alerts sent at warning level are "close\_notify" and "user\_canceled", both of which indicate that the connection is not going to continue normally, see [RFC8446].

In TLS 1.3 [RFC8446], error alerts are not mandatory to send after a fatal error condition. Failure to send TLS Error alerts means that the peer or server would have no way of determining what went wrong. EAP-TLS 1.3 strengthens this requirement. Whenever an implementation encounters a fatal error condition, it **MUST** send an appropriate TLS Error alert.

Figure 4 shows an example message flow where the EAP-TLS server rejects the ClientHello with an error alert. The EAP-TLS server can also partly reject the ClientHello with a HelloRetryRequest, see Section 2.1.6.

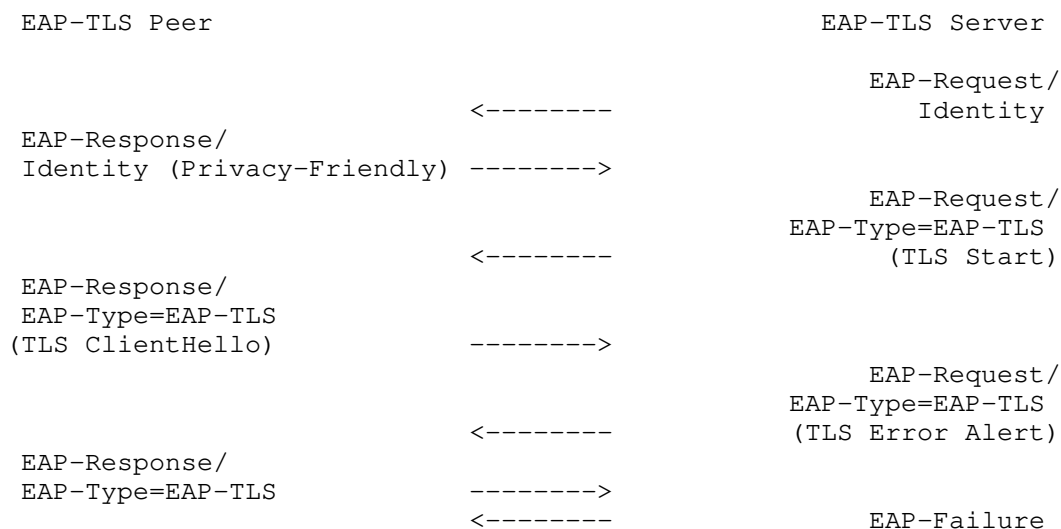


Figure 4: EAP-TLS server rejection of ClientHello

Figure 5 shows an example message flow where EAP-TLS server authentication is unsuccessful and the EAP-TLS peer sends a TLS Error alert.

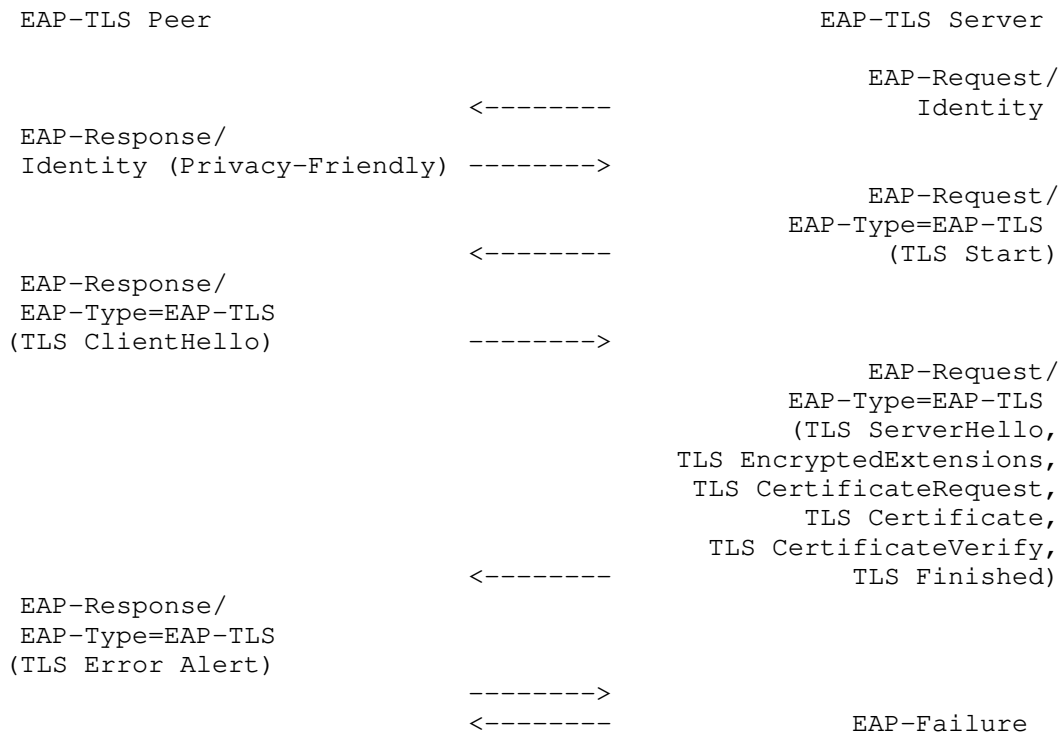


Figure 5: EAP-TLS unsuccessful EAP-TLS server authentication

Figure 6 shows an example message flow where the EAP-TLS server authenticates to the EAP-TLS peer successfully, but the EAP-TLS peer fails to authenticate to the EAP-TLS server and the server sends a TLS Error alert.

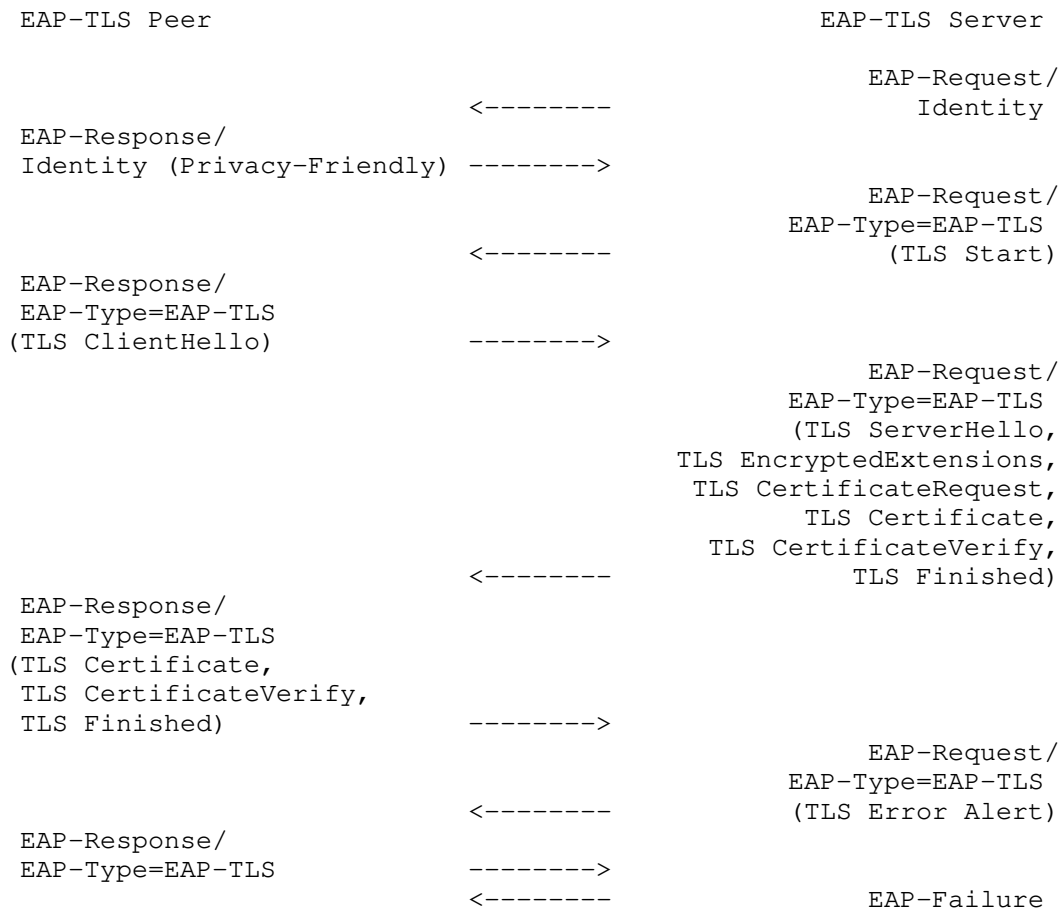


Figure 6: EAP-TLS unsuccessful client authentication

#### 2.1.5. No Peer Authentication

This is a new section when compared to [RFC5216].

Figure 7 shows an example message flow for a successful EAP-TLS full handshake without peer authentication (e.g., emergency services, as described in [RFC7406]).

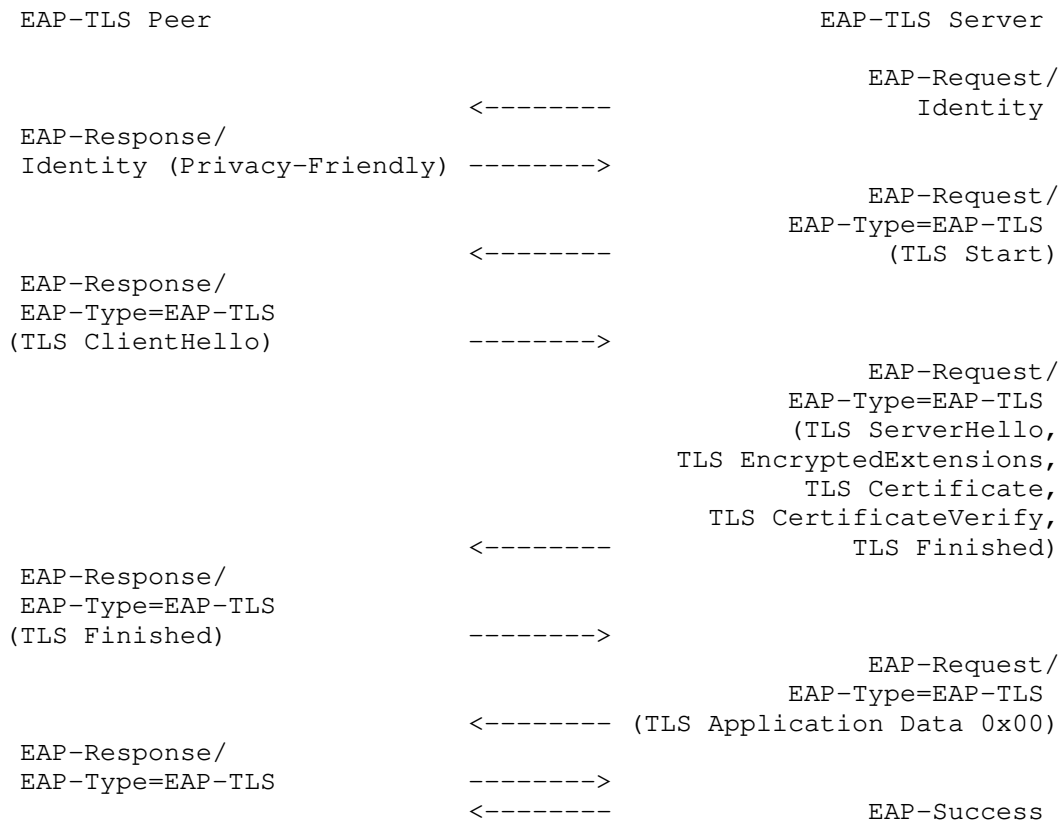


Figure 7: EAP-TLS without peer authentication

#### 2.1.6. Hello Retry Request

This is a new section when compared to [RFC5216].

As defined in TLS 1.3 [RFC8446], EAP-TLS servers can send a HelloRetryRequest message in response to a ClientHello if the EAP-TLS server finds an acceptable set of parameters but the initial ClientHello does not contain all the needed information to continue the handshake. One use case is if the EAP-TLS server does not support the groups in the "key\_share" extension (or there is no "key\_share" extension), but supports one of the groups in the "supported\_groups" extension. In this case the client should send a new ClientHello with a "key\_share" that the EAP-TLS server supports.

Figure 8 shows an example message flow for a successful EAP-TLS full handshake with mutual authentication and HelloRetryRequest. Note the extra round-trip as a result of the HelloRetryRequest.

```

EAP-TLS Peer                                EAP-TLS Server

                                           EAP-Request/
                                           Identity
EAP-Response/                               <-----
Identity (Privacy-Friendly) ----->

                                           EAP-Request/
                                           EAP-Type=EAP-TLS
                                           (TLS Start)
EAP-Response/                               <-----
EAP-Type=EAP-TLS                           ----->
(TLS ClientHello)

                                           EAP-Request/
                                           EAP-Type=EAP-TLS
                                           (TLS HelloRetryRequest)
EAP-Response/                               <-----
EAP-Type=EAP-TLS                           ----->
(TLS ClientHello)

                                           EAP-Request/
                                           EAP-Type=EAP-TLS
                                           (TLS ServerHello,
                                           TLS EncryptedExtensions,
                                           TLS CertificateRequest,
                                           TLS Certificate,
                                           TLS CertificateVerify,
                                           TLS Finished)
EAP-Response/                               ----->
EAP-Type=EAP-TLS                           (TLS Application Data 0x00)
(TLS Certificate,
TLS CertificateVerify,
TLS Finished)

                                           EAP-Request/
                                           EAP-Type=EAP-TLS
                                           (TLS Application Data 0x00)
EAP-Response/                               <-----
EAP-Type=EAP-TLS                           ----->
                                           <-----
                                           EAP-Success

```

Figure 8: EAP-TLS with Hello Retry Request

#### 2.1.7. Identity

This is a new section when compared to [RFC5216].

It is RECOMMENDED to use anonymous NAIs [RFC7542] in the Identity Response as such identities are routable and privacy-friendly. While opaque blobs are allowed by [RFC3748], such identities are NOT



RECOMMENDED as they are not routable and should only be considered in local deployments where the EAP-TLS peer, EAP authenticator, and EAP-TLS server all belong to the same network. Many client certificates contain an identity such as an email address, which is already in NAI format. When the client certificate contains a NAI as subject name or alternative subject name, an anonymous NAI SHOULD be derived from the NAI in the certificate, see Section 2.1.8. More details on identities are described in Sections 2.1.3, 2.1.8, 2.2, and 5.8.

#### 2.1.8. Privacy

This section updates Section 2.1.4 of [RFC5216] by amending it in accordance with the following discussion.

EAP-TLS 1.3 significantly improves privacy when compared to earlier versions of EAP-TLS. EAP-TLS 1.3 forbids cipher suites without confidentiality which means that TLS 1.3 is always encrypting large parts of the TLS handshake including the certificate messages.

EAP-TLS peer and server implementations supporting TLS 1.3 MUST support anonymous Network Access Identifiers (NAIs) (Section 2.4 in [RFC7542]) and a client supporting TLS 1.3 MUST NOT send its username in cleartext in the Identity Response. Following [RFC7542], it is RECOMMENDED to omit the username (i.e., the NAI is @realm), but other constructions such as a fixed username (e.g., anonymous@realm) or an encrypted username (e.g., xCZINCPTK5+7y81CrSYbPg+RKPE30TrYLn4AQc4AC2U=@realm) are allowed. Note that the NAI MUST be a UTF-8 string as defined by the grammar in Section 2.2 of [RFC7542].

The HelloRequest message used for privacy in EAP-TLS 1.2 does not exist in TLS 1.3 but as the certificate messages in TLS 1.3 are encrypted, there is no need to send an empty certificate\_list and perform a second handshake for privacy (as needed by EAP-TLS with earlier versions of TLS). When EAP-TLS is used with TLS version 1.3 the EAP-TLS peer and EAP-TLS server SHALL follow the processing specified by version 1.3 of TLS. This means that the EAP-TLS peer only sends an empty certificate\_list if it does not have an appropriate certificate to send, and the EAP-TLS server MAY treat an empty certificate\_list as a terminal condition.

EAP-TLS with TLS 1.3 is always used with privacy. This does not add any extra round-trips and the message flow with privacy is just the normal message flow as shown in Figure 1.

### 2.1.9. Fragmentation

This section updates Section 2.1.5 of [RFC5216] by amending it in accordance with the following discussion.

Including ContentType (1 byte), ProtocolVersion (2 bytes), and length (2 bytes) headers a single TLS record may be up to 16645 octets in length. EAP-TLS fragmentation support is provided through addition of a flags octet within the EAP-Response and EAP-Request packets, as well as a (conditional) TLS Message Length field of four octets. Implementations **MUST NOT** set the L bit in unfragmented messages, but **MUST** accept unfragmented messages with and without the L bit set.

Some EAP implementations and access networks may limit the number of EAP packet exchanges that can be handled. To avoid fragmentation, it is **RECOMMENDED** to keep the sizes of EAP-TLS peer, EAP-TLS server, and trust anchor certificates small and the length of the certificate chains short. In addition, it is **RECOMMENDED** to use mechanisms that reduce the sizes of Certificate messages. For a detailed discussion on reducing message sizes to prevent fragmentation, see [I-D.ietf-emu-eaptlscert].

### 2.2. Identity Verification

This section updates Section 2.2 of [RFC5216] by amending it in accordance with the following discussion. The guidance in this section is relevant for EAP-TLS in general (regardless of the underlying TLS version used).

The EAP peer identity provided in the EAP-Response/Identity is not authenticated by EAP-TLS. Unauthenticated information **MUST NOT** be used for accounting purposes or to give authorization. The authenticator and the EAP-TLS server **MAY** examine the identity presented in EAP-Response/Identity for purposes such as routing and EAP method selection. EAP-TLS servers **MAY** reject conversations if the identity does not match their policy. Note that this also applies to resumption, see Sections 2.1.3, 5.6, and 5.7.

The EAP server identity in the TLS server certificate is typically a fully qualified domain name (FQDN) in the SubjectAltName (SAN) extension. Since EAP-TLS deployments may use more than one EAP server, each with a different certificate, EAP peer implementations **SHOULD** allow for the configuration of one or more trusted root certificates (CA certificate) to authenticate the server certificate and one or more server names to match against the SubjectAltName (SAN) extension in the server certificate. If any of the configured names match any of the names in the SAN extension then the name check passes. To simplify name matching, an EAP-TLS deployment can assign

a name to represent an authorized EAP server and EAP Server certificates can include this name in the list of SANs for each certificate that represents an EAP-TLS server. If server name matching is not used, then it degrades the confidence that the EAP server with which it is interacting is authoritative for the given network. If name matching is not used with a public root CA, then effectively any server can obtain a certificate which will be trusted for EAP authentication by the Peer. While this guidance to verify domain names is new, and was not mentioned in [RFC5216], it has been widely implemented in EAP-TLS peers. As such, it is believed that this section contains minimal new interoperability or implementation requirements on EAP-TLS peers and can be applied to earlier versions of TLS.

The process of configuring a root CA certificate and a server name is non-trivial and therefore automated methods of provisioning are RECOMMENDED. For example, the eduroam federation [RFC7593] provides a Configuration Assistant Tool (CAT) to automate the configuration process. In the absence of a trusted root CA certificate (user configured or system-wide), EAP peers MAY implement a trust on first use (TOFU) mechanism where the peer trusts and stores the server certificate during the first connection attempt. The EAP peer ensures that the server presents the same stored certificate on subsequent interactions. Use of a TOFU mechanism does not allow for the server certificate to change without out-of-band validation of the certificate and is therefore not suitable for many deployments including ones where multiple EAP servers are deployed for high availability. TOFU mechanisms increase the susceptibility to traffic interception attacks and should only be used if there are adequate controls in place to mitigate this risk.

### 2.3. Key Hierarchy

This section updates Section 2.3 of [RFC5216] by replacing it in accordance with the following discussion.

TLS 1.3 replaces the TLS pseudorandom function (PRF) used in earlier versions of TLS with HKDF and completely changes the Key Schedule. The key hierarchies shown in Section 2.3 of [RFC5216] are therefore not correct when EAP-TLS is used with TLS version 1.3. For TLS 1.3 the key schedule is described in Section 7.1 of [RFC8446].

When EAP-TLS is used with TLS version 1.3 the Key\_Material and Method-Id SHALL be derived from the exporter\_secret using the TLS exporter interface [RFC5705] (for TLS 1.3 this is defined in Section 7.5 of [RFC8446]). Type is the value of the EAP Type field defined in Section 2 of [RFC3748]. For EAP-TLS the Type field has value 0x0D.

```
Type = 0x0D
Key_Material = TLS-Exporter("EXPORTER_EAP_TLS_Key_Material",
                             Type, 128)
Method-Id    = TLS-Exporter("EXPORTER_EAP_TLS_Method-Id",
                             Type, 64)
Session-Id   = Type || Method-Id
```

The MSK and EMSK are derived from the Key\_Material in the same manner as with EAP-TLS [RFC5216], Section 2.3. The definitions are repeated below for simplicity:

```
MSK          = Key_Material(0, 63)
EMSK         = Key_Material(64, 127)
```

Other TLS based EAP methods can use the TLS exporter in a similar fashion, see [I-D.ietf-emu-tls-eap-types].

[RFC5247] deprecates the use of IV. Thus, RECV-IV and SEND-IV are not exported in EAP-TLS with TLS 1.3. As noted in [RFC5247], lower layers use the MSK in a lower-layer-dependent manner. EAP-TLS with TLS 1.3 exports the MSK and does not specify how it is used by lower layers.

Note that the key derivation MUST use the length values given above. While in TLS 1.2 and earlier it was possible to truncate the output by requesting less data from the TLS-Exporter function, this practice is not possible with TLS 1.3. If an implementation intends to use only a part of the output of the TLS-Exporter function, then it MUST ask for the full output and then only use the desired part. Failure to do so will result in incorrect values being calculated for the above keying material.

By using the TLS exporter, EAP-TLS can use any TLS 1.3 implementation which provides a public API for the exporter. Note that when TLS 1.2 is used with the EAP-TLS exporter [RFC5705] it generates the same key material as in EAP-TLS [RFC5216].

## 2.4. Parameter Negotiation and Compliance Requirements

This section updates Section 2.4 of [RFC5216] by amending it in accordance with the following discussion.

TLS 1.3 cipher suites are defined differently than in earlier versions of TLS (see Section B.4 of [RFC8446]), and the cipher suites discussed in Section 2.4 of [RFC5216] can therefore not be used when EAP-TLS is used with TLS version 1.3.

When EAP-TLS is used with TLS version 1.3, the EAP-TLS peers and EAP-TLS servers MUST comply with the compliance requirements (mandatory-to-implement cipher suites, signature algorithms, key exchange algorithms, extensions, etc.) defined in Section 9 of [RFC8446]. In EAP-TLS with TLS 1.3, only cipher suites with confidentiality SHALL be supported.

While EAP-TLS does not protect any application data except for the 0x00 byte that serves as protected success indication, the negotiated cipher suites and algorithms MAY be used to secure data as done in other TLS-based EAP methods.

## 2.5. EAP State Machines

This is a new section when compared to [RFC5216] and only applies to TLS 1.3. [RFC4137] offers a proposed state machine for EAP.

TLS 1.3 [RFC8446] introduces post-handshake messages. These post-handshake messages use the handshake content type and can be sent after the main handshake. Examples of post-handshake messages are NewSessionTicket, which is used for resumption and KeyUpdate, which is not useful and not expected in EAP-TLS. After sending TLS Finished, the EAP-TLS server may send any number of post-handshake messages in one or more EAP-Requests.

To provide a protected success result indication and to decrease the uncertainty for the EAP-TLS peer, the following procedure MUST be followed:

When an EAP-TLS server has successfully processed the TLS client Finished and sent its last handshake message (Finished or a post-handshake message), it sends an encrypted TLS record with application data 0x00. The encrypted TLS record with application data 0x00 is a protected success result indication, as defined in [RFC3748]. After sending an EAP-Request that contains the protected success result indication, the EAP-TLS server must not send any more EAP-Request and may only send an EAP-Success. The EAP-TLS server MUST NOT send an encrypted TLS record with application data 0x00 alert before it has successfully processed the client finished and sent its last handshake message.

TLS Error alerts SHOULD be considered a failure result indication, as defined in [RFC3748]. Implementations following [RFC4137] set the alternate indication of failure variable altReject after sending or receiving an error alert. After sending or receiving a TLS Error alert, the EAP-TLS server may only send an EAP-Failure. Protected TLS Error alerts are protected failure result indications, unprotected TLS Error alerts are not.

The keying material can be derived after the TLS server Finished has been sent or received. Implementations following [RFC4137] can then set the eapKeyData and aaaEapKeyData variables.

The keying material can be made available to lower layers and the authenticator after the authenticated success result indication has been sent or received. Implementations following [RFC4137] can set the eapKeyAvailable and aaaEapKeyAvailable variables.

### 3. Detailed Description of the EAP-TLS Protocol

No updates to Section 3 of [RFC5216].

### 4. IANA considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP-TLS 1.3 protocol in accordance with [RFC8126].

This document requires IANA to add the following labels to the TLS Exporter Label Registry defined by [RFC5705]. These labels are used in derivation of Key\_Material and Method-Id as defined in Section 2.3:

Value	DTLS-OK	Recommended	Note
EXPORTER_EAP_TLS_Key_Material	N	Y	
EXPORTER_EAP_TLS_Method-Id	N	Y	

Table 1: TLS Exporter Label Registry

### 5. Security Considerations

The security considerations of TLS 1.3 [RFC8446] apply to EAP-TLS 1.3

#### 5.1. Security Claims

Using EAP-TLS with TLS 1.3 does not change the security claims for EAP-TLS as given in Section 5.1 of [RFC5216]. However, it strengthens several of the claims as described in the following updates to the notes given in Section 5.1 of [RFC5216].

[1] Mutual authentication: By mandating revocation checking of certificates, the authentication in EAP-TLS with TLS 1.3 is stronger as authentication with revoked certificates will always fail.

[2] Confidentiality: The TLS 1.3 handshake offers much better confidentiality than earlier versions of TLS. EAP-TLS with TLS 1.3 mandates use of cipher suites that ensure confidentiality. TLS 1.3 also encrypts certificates and some of the extensions. When using EAP-TLS with TLS 1.3, the use of privacy is mandatory and does not cause any additional round-trips.

[3] Cryptographic strength: TLS 1.3 only defines strong algorithms without major weaknesses and EAP-TLS with TLS 1.3 always provides forward secrecy, see [RFC8446]. Weak algorithms such as 3DES, CBC mode, RC4, SHA-1, MD5, P-192, and RSA-1024 have not been registered for use in TLS 1.3.

[4] Cryptographic Negotiation: The TLS layer handles the negotiation of cryptographic parameters. When EAP-TLS is used with TLS 1.3, EAP-TLS inherits the cryptographic negotiation of AEAD algorithm, HKDF hash algorithm, key exchange groups, and signature algorithm, see Section 4.1.1 of [RFC8446].

## 5.2. Peer and Server Identities

No updates to section 5.2 of [RFC5216]. Note that Section 2.2 has additional discussion on identities.

## 5.3. Certificate Validation

No updates to section 5.3 of [RFC5216]. In addition to section 5.3 of [RFC5216], guidance on server certificate validation can be found in [RFC6125].

## 5.4. Certificate Revocation

This section updates Section 5.4 of [RFC5216] by amending it in accordance with the following discussion.

There are a number of reasons (e.g., key compromise, CA compromise, privilege withdrawn, etc.) why EAP-TLS peer, EAP-TLS server, or sub-CA certificates have to be revoked before their expiry date. Revocation of the EAP-TLS server's certificate is complicated by the fact that the EAP-TLS peer may not have Internet connectivity until authentication completes.

When EAP-TLS is used with TLS 1.3, the revocation status of all the certificates in the certificate chains MUST be checked (except the trust anchor). An implementation may use Certificate Revocation List (CRL), Online Certificate Status Protocol (OCSP), or other standardized/proprietary methods for revocation checking. Examples of proprietary methods are non-standard formats for distribution of revocation lists as well as certificates with very short lifetime.

EAP-TLS servers supporting TLS 1.3 MUST implement Certificate Status Requests (OCSP stapling) as specified in [RFC6066] and Section 4.4.2.1 of [RFC8446]. It is RECOMMENDED that EAP-TLS peers and EAP-TLS servers use OCSP stapling for verifying the status of the EAP-TLS server's certificate chain. When an EAP-TLS peer uses Certificate Status Requests to check the revocation status of the EAP-TLS server's certificate chain it MUST treat a CertificateEntry (except the trust anchor) without a valid CertificateStatus extension as invalid and abort the handshake with an appropriate alert. The OCSP status handling in TLS 1.3 is different from earlier versions of TLS, see Section 4.4.2.1 of [RFC8446]. In TLS 1.3 the OCSP information is carried in the CertificateEntry containing the associated certificate instead of a separate CertificateStatus message as in [RFC6066]. This enables sending OCSP information for all certificates in the certificate chain (except the trust anchor).

To enable revocation checking in situations where EAP-TLS peers do not implement or use OCSP stapling, and where network connectivity is not available prior to authentication completion, EAP-TLS peer implementations MUST also support checking for certificate revocation after authentication completes and network connectivity is available. An EAP peer implementation SHOULD NOT trust the network (and any services) until it has verified the revocation status of the server certificate after receiving network connectivity. An EAP peer MUST use a secure transport to verify the revocation status of the server certificate. An EAP peer SHOULD NOT send any other traffic before revocation checking for the server certificate is complete.

## 5.5. Packet Modification Attacks

This section updates Section 5.5 of [RFC5216] by amending it in accordance with the following discussion.

As described in [RFC3748] and Section 5.5 of [RFC5216], the only information that is integrity and replay protected in EAP-TLS are the parts of the TLS Data that TLS protects. All other information in the EAP-TLS message exchange including EAP-Request and EAP-Response headers, the identity in the identity response, EAP-TLS packet header fields, Type, and Flags, EAP-Success, and EAP-Failure can be modified, spoofed, or replayed.



Protected TLS Error alerts are protected failure result indications and enables the EAP-TLS peer and EAP-TLS server to determine that the failure result was not spoofed by an attacker. Protected failure result indications provide integrity and replay protection but MAY be unauthenticated. Protected failure results do not significantly improve availability as TLS 1.3 treats most malformed data as a fatal error.

## 5.6. Authorization

This is a new section when compared to [RFC5216]. The guidance in this section is relevant for EAP-TLS in general (regardless of the underlying TLS version used).

EAP servers will usually require the EAP peer to provide a valid certificate and will fail the connection if one is not provided. Some deployments may permit no peer authentication for some or all connections. When peer authentication is not used, EAP-TLS server implementations MUST take care to limit network access appropriately for unauthenticated peers and implementations MUST use resumption with caution to ensure that a resumed session is not granted more privilege than was intended for the original session. An example of limiting network access would be to invoke a vendor's walled garden or quarantine network functionality.

EAP-TLS is typically encapsulated in other protocols, such as PPP [RFC1661], RADIUS [RFC2865], Diameter [RFC6733], or PANA [RFC5191]. The encapsulating protocols can also provide additional, non-EAP information to an EAP-TLS server. This information can include, but is not limited to, information about the authenticator, information about the EAP-TLS peer, or information about the protocol layers above or below EAP (MAC addresses, IP addresses, port numbers, Wi-Fi SSID, etc.). EAP-TLS servers implementing EAP-TLS inside those protocols can make policy decisions and enforce authorization based on a combination of information from the EAP-TLS exchange and non-EAP information.

As noted in Section 2.2, the identity presented in EAP-Response/Identity is not authenticated by EAP-TLS and is therefore trivial for an attacker to forge, modify, or replay. Authorization and accounting MUST be based on authenticated information such as information in the certificate or the PSK identity and cached data provisioned for resumption as described in Section 5.7. Note that the requirements for Network Access Identifiers (NAIs) specified in Section 4 of [RFC7542] still apply and MUST be followed.

EAP-TLS servers MAY reject conversations based on non-EAP information provided by the encapsulating protocol, for example, if the MAC address of the authenticator does not match the expected policy.

In addition to allowing configuration of one or more trusted root certificates (CA certificate) to authenticate the server certificate and one or more server names to match against the SubjectAltName (SAN) extension, EAP peer implementations MAY allow binding the configured acceptable SAN to a specific CA (or CAs) that should have issued the server certificate to prevent attacks from rogue or compromised CAs.

### 5.7. Resumption

This is a new section when compared to [RFC5216]. The guidance in this section is relevant for EAP-TLS in general (regardless of the underlying TLS version used).

There are a number of security issues related to resumption that are not described in [RFC5216]. The problems, guidelines, and requirements in this section therefore applies to EAP-TLS when it is used with any version of TLS.

When resumption occurs, it is based on cached information at the TLS layer. To perform resumption securely, the EAP-TLS peer and EAP-TLS server need to be able to securely retrieve authorization information such as certificate chains from the initial full handshake. This document uses the term "cached data" to describe such information. Authorization during resumption MUST be based on such cached data. The EAP-TLS peer and EAP-TLS server MAY perform fresh revocation checks on the cached certificate data. Any security policies for authorization MUST be followed also for resumption. The certificates may have been revoked since the initial full handshake and the authorizations of the other party may have been reduced. If the cached revocation data is not sufficiently current, the EAP-TLS peer or EAP-TLS server MAY force a full TLS handshake.

There are two ways to retrieve the cached data from the original full handshake. The first method is that the EAP-TLS server and client cache the information locally. The cached information is identified by an identifier. For TLS versions before 1.3, the identifier can be the session ID, for TLS 1.3, the identifier is the PSK identity. The second method for retrieving cached information is via [RFC5077] or [RFC8446], where the EAP-TLS server avoids storing information locally and instead encapsulates the information into a ticket which is sent to the client for storage. This ticket is encrypted using a key that only the EAP-TLS server knows. Note that the client still needs to cache the original handshake information locally and will

obtain it while determining the session ID or PSK identity to use for resumption. However, the EAP-TLS server is able to decrypt the ticket or PSK to obtain the original handshake information.

The EAP-TLS server or EAP client MUST cache data during the initial full handshake sufficient to allow authorization decisions to be made during resumption. If cached data cannot be retrieved securely, resumption MUST NOT be done.

The above requirements also apply if the EAP-TLS server expects some system to perform accounting for the session. Since accounting must be tied to an authenticated identity, and resumption does not supply such an identity, accounting is impossible without access to cached data. Therefore, systems which expect to perform accounting for the session SHOULD cache an identifier which can be used in subsequent accounting.

As suggested in [RFC8446], EAP-TLS peers MUST NOT store resumption PSKs or tickets (and associated cached data) for longer than 604800 seconds (7 days), regardless of the PSK or ticket lifetime. The EAP-TLS peer MAY delete them earlier based on local policy. The cached data MAY also be removed on the EAP-TLS server or EAP-TLS peer if any certificate in the certificate chain has been revoked or has expired. In all such cases, an attempt at resumption results in a full TLS handshake instead.

Information from the EAP-TLS exchange (e.g., the identity provided in EAP-Response/Identity) as well as non-EAP information (e.g., IP addresses) may change between the initial full handshake and resumption. This change creates a "time-of-check time-of-use" (TOCTOU) security vulnerability. A malicious or compromised user could supply one set of data during the initial authentication, and a different set of data during resumption, potentially allowing them to obtain access that they should not have.

If any authorization, accounting, or policy decisions were made with information that has changed between the initial full handshake and resumption, and if change may lead to a different decision, such decisions MUST be reevaluated. It is RECOMMENDED that authorization, accounting, and policy decisions are reevaluated based on the information given in the resumption. EAP-TLS servers MAY reject resumption where the information supplied during resumption does not match the information supplied during the original authentication. If a safe decision is not possible, EAP-TLS servers SHOULD reject the resumption and continue with a full handshake.

Section 2.2 and 4.2.11 of [RFC8446] provides security considerations for TLS 1.3 resumption.

## 5.8. Privacy Considerations

This is a new section when compared to [RFC5216].

TLS 1.3 offers much better privacy than earlier versions of TLS as discussed in Section 2.1.8. In this section, we only discuss the privacy properties of EAP-TLS with TLS 1.3. For privacy properties of TLS 1.3 itself, see [RFC8446].

EAP-TLS sends the standard TLS 1.3 handshake messages encapsulated in EAP packets. Additionally, the EAP-TLS peer sends an identity in the first EAP-Response. The other fields in the EAP-TLS Request and the EAP-TLS Response packets do not contain any cleartext privacy-sensitive information.

Tracking of users by eavesdropping on identity responses or certificates is a well-known problem in many EAP methods. When EAP-TLS is used with TLS 1.3, all certificates are encrypted, and the username part of the identity response is not revealed (e.g., using anonymous NAIs). Note that even though all certificates are encrypted, the server's identity is only protected against passive attackers while the client's identity is protected against both passive and active attackers. As with other EAP methods, even when privacy-friendly identifiers or EAP tunneling is used, the domain name (i.e., the realm) in the NAI is still typically visible. How much privacy-sensitive information the domain name leaks is highly dependent on how many other users are using the same domain name in the particular access network. If all EAP-TLS peers have the same domain, no additional information is leaked. If a domain name is used by a small subset of the EAP-TLS peers, it may aid an attacker in tracking or identifying the user.

Without padding, information about the size of the client certificate is leaked from the size of the EAP-TLS packets. The EAP-TLS packets sizes may therefore leak information that can be used to track or identify the user. If all client certificates have the same length, no information is leaked. EAP-TLS peers SHOULD use record padding, see Section 5.4 of [RFC8446] to reduce information leakage of certificate sizes.

If anonymous NAIs are not used, the privacy-friendly identifiers need to be generated with care. The identities MUST be generated in a cryptographically secure way so that it is computationally infeasible for an attacker to differentiate two identities belonging to the same user from two identities belonging to different users in the same realm. This can be achieved, for instance, by using random or pseudo-random usernames such as random byte strings or ciphertexts and only using the pseudo-random usernames a single time. Note that

the privacy-friendly usernames also MUST NOT include substrings that can be used to relate the identity to a specific user. Similarly, privacy-friendly username MUST NOT be formed by a fixed mapping that stays the same across multiple different authentications.

An EAP-TLS peer with a policy allowing communication with EAP-TLS servers supporting only TLS 1.2 without privacy and with a static RSA key exchange is vulnerable to disclosure of the EAP-TLS peer username. An active attacker can in this case make the EAP-TLS peer believe that an EAP-TLS server supporting TLS 1.3 only supports TLS 1.2 without privacy. The attacker can simply impersonate the EAP-TLS server and negotiate TLS 1.2 with static RSA key exchange and send an TLS alert message when the EAP-TLS peer tries to use privacy by sending an empty certificate message. Since the attacker (impersonating the EAP-TLS server) does not provide a proof-of-possession of the private key until the Finished message when a static RSA key exchange is used, an EAP-TLS peer may inadvertently disclose its identity (username) to an attacker. Therefore, it is RECOMMENDED for EAP-TLS peers to not use EAP-TLS with TLS 1.2 and static RSA based cipher suites without privacy. This implies that an EAP-TLS peer SHOULD NOT continue the EAP authentication attempt if a TLS 1.2 EAP-TLS server sends an EAP-TLS/Request with a TLS alert message in response to an empty certificate message from the peer.

#### 5.9. Pervasive Monitoring

This is a new section when compared to [RFC5216].

Pervasive monitoring refers to widespread surveillance of users. In the context of EAP-TLS, pervasive monitoring attacks can target EAP-TLS peer devices for tracking them (and their users) as and when they join a network. By encrypting more information, mandating the use of privacy, and always providing forward secrecy, EAP-TLS with TLS 1.3 offers much better protection against pervasive monitoring. In addition to the privacy attacks discussed above, surveillance on a large scale may enable tracking of a user over a wide geographical area and across different access networks. Using information from EAP-TLS together with information gathered from other protocols increases the risk of identifying individual users.

#### 5.10. Discovered Vulnerabilities

This is a new section when compared to [RFC5216].

Over the years, there have been several serious attacks on earlier versions of Transport Layer Security (TLS), including attacks on its most commonly used ciphers and modes of operation. [RFC7457] summarizes the attacks that were known at the time of publishing and

BCP 195 [RFC7525] [RFC8996] provides recommendations and requirements for improving the security of deployed services that use TLS. However, many of the attacks are less serious for EAP-TLS as EAP-TLS only uses the TLS handshake and does not protect any application data. EAP-TLS implementations MUST mitigate known attacks. EAP-TLS implementations need to monitor and follow new EAP and TLS related security guidance and requirements such as [RFC8447] and [I-D.ietf-tls-md5-sha1-deprecate].

#### 5.11. Cross-Protocol Attacks

This is a new section when compared to [RFC5216].

Allowing the same certificate to be used in multiple protocols can potentially allow an attacker to authenticate via one protocol, and then "resume" that session in another protocol. Section 2.2 above suggests that certificates typically have one or more FQDNs in the SAN extension. However, those fields are for EAP validation only, and do not indicate that the certificates are suitable for use on WWW (or other) protocol server on the named host.

Section 2.1.3 above suggests that authorization rules should be re-applied on resumption, but does not mandate this behavior. As a result, this cross-protocol resumption could allow the attacker to bypass authorization policies, and to obtain undesired access to secured systems. Along with making sure that appropriate authorization information is available and used during resumption, using different certificates and resumption caches for different protocols is RECOMMENDED to help keep different protocol usages separate.

### 6. References

#### 6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <<https://www.rfc-editor.org/info/rfc5216>>.

- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <<https://www.rfc-editor.org/info/rfc6960>>.
- [RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <<https://www.rfc-editor.org/info/rfc7542>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## 6.2. Informative references

- [IEEE-802.1X] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and metropolitan area networks -- Port-Based Network Access Control", IEEE Standard 802.1X-2020 , February 2020.

- [IEEE-802.11] Institute of Electrical and Electronics Engineers, "IEEE Standard for Information technologyTelecommunications and information exchange between systems Local and metropolitan area networksSpecific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11-2020 , February 2021.
- [IEEE-802.1AE] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and metropolitan area networks -- Media Access Control (MAC) Security", IEEE Standard 802.1AE-2018 , December 2018.
- [TS.33.501] 3GPP, "Security architecture and procedures for 5G System", 3GPP TS 33.501 17.3.0, September 2021.
- [MultaFire] MultaFire, "MultaFire Release 1.1 specification", 2019.
- [PEAP] Microsoft Corporation, "[MS-PEAP]: Protected Extensible Authentication Protocol (PEAP)", June 2021.
- [RFC1661] Simpson, W., Ed., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, DOI 10.17487/RFC1661, July 1994, <<https://www.rfc-editor.org/info/rfc1661>>.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <<https://www.rfc-editor.org/info/rfc2246>>.
- [RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, DOI 10.17487/RFC2560, June 1999, <<https://www.rfc-editor.org/info/rfc2560>>.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <<https://www.rfc-editor.org/info/rfc2865>>.



- [RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, DOI 10.17487/RFC3280, April 2002, <<https://www.rfc-editor.org/info/rfc3280>>.
- [RFC4137] Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", RFC 4137, DOI 10.17487/RFC4137, August 2005, <<https://www.rfc-editor.org/info/rfc4137>>.
- [RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, DOI 10.17487/RFC4282, December 2005, <<https://www.rfc-editor.org/info/rfc4282>>.
- [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006, <<https://www.rfc-editor.org/info/rfc4346>>.
- [RFC4851] Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, DOI 10.17487/RFC4851, May 2007, <<https://www.rfc-editor.org/info/rfc4851>>.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <<https://www.rfc-editor.org/info/rfc5077>>.
- [RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <<https://www.rfc-editor.org/info/rfc5191>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <<https://www.rfc-editor.org/info/rfc5247>>.

- [RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, DOI 10.17487/RFC5281, August 2008, <<https://www.rfc-editor.org/info/rfc5281>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <<https://www.rfc-editor.org/info/rfc6733>>.
- [RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <<https://www.rfc-editor.org/info/rfc7170>>.
- [RFC7406] Schulzrinne, H., McCann, S., Bajko, G., Tschofenig, H., and D. Kroesenberg, "Extensions to the Emergency Services Architecture for Dealing With Unauthenticated and Unauthorized Devices", RFC 7406, DOI 10.17487/RFC7406, December 2014, <<https://www.rfc-editor.org/info/rfc7406>>.
- [RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <<https://www.rfc-editor.org/info/rfc7457>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC7593] Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam Architecture for Network Roaming", RFC 7593, DOI 10.17487/RFC7593, September 2015, <<https://www.rfc-editor.org/info/rfc7593>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <<https://www.rfc-editor.org/info/rfc8447>>.
- [RFC8996] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <<https://www.rfc-editor.org/info/rfc8996>>.
- [I-D.ietf-tls-md5-sha1-deprecate]  
Velvindron, L., Moriarty, K., and A. Ghedini, "Deprecating MD5 and SHA-1 signature hashes in (D)TLS 1.2", Work in Progress, Internet-Draft, draft-ietf-tls-md5-sha1-deprecate-09, 20 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-tls-md5-sha1-deprecate-09.txt>>.
- [I-D.ietf-emu-eaptls-cert]  
Sethi, M., Mattsson, J., and S. Turner, "Handling Large Certificates and Long Certificate Chains in TLS-based EAP Methods", Work in Progress, Internet-Draft, draft-ietf-emu-eaptls-cert-08, 20 November 2020, <<https://www.ietf.org/archive/id/draft-ietf-emu-eaptls-cert-08.txt>>.
- [I-D.ietf-tls-ticket-requests]  
Pauly, T., Schinazi, D., and C. A. Wood, "TLS Ticket Requests", Work in Progress, Internet-Draft, draft-ietf-tls-ticket-requests-07, 3 December 2020, <<https://www.ietf.org/archive/id/draft-ietf-tls-ticket-requests-07.txt>>.
- [I-D.ietf-emu-tls-eap-types]  
DeKok, A., "TLS-based EAP types and TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-emu-tls-eap-types-03, 22 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-emu-tls-eap-types-03.txt>>.
- [I-D.ietf-tls-rfc8446bis]  
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-02, 23 August 2021, <<https://www.ietf.org/archive/id/draft-ietf-tls-rfc8446bis-02.txt>>.

## Appendix A. Updated References

All the following references in [RFC5216] are updated as specified below when EAP-TLS is used with TLS 1.3.

All references to [RFC2560] are updated to refer to [RFC6960].

All references to [RFC3280] are updated to refer to [RFC5280].  
References to Section 4.2.1.13 of [RFC3280] are updated to refer to  
Section 4.2.1.12 of [RFC5280].

All references to [RFC4282] are updated to refer to [RFC7542].  
References to Section 2.1 of [RFC4282] are updated to refer to  
Section 2.2 of [RFC7542].

## Acknowledgments

The authors want to thank Bernard Aboba, Jari Arkko, Terry Burton, Alan DeKok, Ari Keraenen, Benjamin Kaduk, Jouni Malinen, Oleg Pekar, Eric Rescorla, Jim Schaad, Joseph Salowey, Martin Thomson, Vesa Torvinen, Hannes Tschofenig, and Heikki Vatiainen for comments and suggestions on the draft. Special thanks to the document shepherd Joseph Salowey.

## Contributors

Alan DeKok, FreeRADIUS

## Authors' Addresses

John Preuß Mattsson  
Ericsson  
SE-164 40 Stockholm  
Sweden

Email: john.mattsson@ericsson.com

Mohit Sethi  
Ericsson  
FI-02420 Jorvas  
Finland

Email: mohit@piuha.net

Network Working Group  
INTERNET-DRAFT  
Updates: 5247, 5281, 7170  
Category: Standards Track  
Expires: September 05, 2022

DeKok, Alan  
FreeRADIUS  
5 March 2022

TLS-based EAP types and TLS 1.3  
draft-ietf-emu-tls-eap-types-05.txt

Abstract

EAP-TLS (RFC 5216) has been updated for TLS 1.3 in RFC 9190. Many other EAP types also depend on TLS, such as FAST (RFC 4851), TTLS (RFC 5281), TEAP (RFC 7170), and possibly many vendor specific EAP methods. This document updates those methods in order to use the new key derivation methods available in TLS 1.3. Additional changes necessitated by TLS 1.3 are also discussed.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on January 29, 2021.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents  
(<http://trustee.ietf.org/license-info/>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction .....	4
1.1. Requirements Language .....	4
2. Using TLS-based EAP methods with TLS 1.3 .....	5
2.1. Key Derivation .....	5
2.2. TEAP .....	6
2.3. FAST .....	7
2.4. TTLS .....	8
2.4.1. Client Certificates .....	8
2.5. PEAP .....	9
2.5.1. Client Certificates .....	9
3. Application Data .....	9
3.1. Identities .....	11
4. Resumption .....	13
5. Implementation Status .....	14
6. Security Considerations .....	14
6.1. Protected Success and Failure indicators .....	15
7. IANA Considerations .....	16
8. References .....	17
8.1. Normative References .....	17
8.2. Informative References .....	18

## 1. Introduction

EAP-TLS has been updated for TLS 1.3 in [RFC9190]. Many other EAP types also depend on TLS, such as FAST [RFC4851], TTLS [RFC5281], TEAP [RFC7170], and possibly many vendor specific EAP methods such as PEAP [PEAP]. All of these methods use key derivation functions which are no longer applicable to TLS 1.3. As such, all of those methods are incompatible with TLS 1.3.

This document updates those methods in order to be used with TLS 1.3. These changes involve defining new key derivation functions. We also discuss implementation issues in order to highlight differences between TLS 1.3 and earlier versions of TLS.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.



## 2. Using TLS-based EAP methods with TLS 1.3

In general, all of the requirements of [RFC9190] apply to other EAP methods that wish to use TLS 1.3. Unless otherwise required herein, implementations of EAP methods that wish to use TLS 1.3 MUST follow the guidelines in [RFC9190].

There remain some differences between EAP-TLS and other TLS-based EAP methods which necessitates this document. The main difference is that [RFC9190] uses the EAP-TLS Type (value 0x0D) in a number of calculations, whereas other method types will use their own Type value instead of the EAP-TLS Type value. This topic is discussed further below in Section 2.

An additional difference is that [RFC9190] Section 2.5 requires that once the EAP-TLS handshake has completed, the EAP server sends a protected success result indication. This indication is composed of one octet (0x00) of application data. Other TLS-based EAP methods also use this indicator, but only during resumption. When other TLS-based EAP methods use full authentication, the indicator is not needed, and is not used. This topic is explained in more detail below, in Section 3 and Section 4.

Finally, the document includes clarifications on how various TLS-based parameters are calculated when using TLS 1.3. These parameters are different for each EAP method, so they are discussed separately.

### 2.1. Key Derivation

The key derivation for TLS-based EAP methods depends on the value of the EAP Type as defined by [IANA] in the Extensible Authentication Protocol (EAP) Registry. The most important definition is of the Type field, as first defined in [RFC3748] Section 2:

Type = value of the EAP Method type

For the purposes of this specification, when we refer to logical Type, we mean that the logical Type is defined to be 1 octet for values smaller than 254 (the value for the Expanded Type), and when Expanded EAP Types are used, the logical Type is defined to be the concatenation of the fields required to define the Expanded Type, including the Type with value 0xfe, Vendor-Id (in network byte order) and Vendor-Type fields (in network byte order) defined in [RFC3748] Section 5.7, as given below:

Type = 0xFE || Vendor-Id || Vendor-Type

This definition does not alter the meaning of Type in [RFC3748], or

change the structure of EAP packets. Instead, this definition allows us to simplify references to EAP Types, by just using a logical "Type" instead of referring to "the Type field or the Type field with value 0xfe, plus the Vendor-ID and Vendor-Type". For example, the value of Type for PEAP is simply 0x19.

Unless otherwise discussed below, the key derivation functions for all TLS-based EAP Types are defined in [RFC9190] Section 2.3, and reproduced here for clarity:

```
Key_Material = TLS-Exporter("EXPORTER_EAP_TLS_Key_Material",
                             Type, 128)
Method-Id    = TLS-Exporter("EXPORTER_EAP_TLS_Method-Id",
                             Type, 64)
Session-Id   = Type || Method-Id
MSK          = Key_Material(0, 63)
EMSK         = Key_Material(64, 127)
```

We note that these definitions re-use the EAP-TLS exporter labels, and change the derivation only by adding a dependency on the logical Type. The reason for this change is simplicity. There does not appear to be compelling reasons to make the labels method-specific, when they can just include the logical Type in the key derivation.

These definitions apply in their entirety to TTLS [RFC5281] and PEAP as defined in [PEAP] and [MSPEAP]. Some definitions apply to FAST and TEAP, with exceptions as noted below.

It is RECOMMENDED that vendor-defined TLS-based EAP methods use the above definitions for TLS 1.3. There is no compelling reason to use different definitions.

## 2.2. TEAP

[RFC7170] Section 5.2 gives a definition for the Inner Method Session Key (IMSK), which depends on the TLS-PRF. When the inner methods generates an EMSK, we update that definition for TLS 1.3 as:

```
IMSK = TLS-Exporter("TEAPbindkey@ietf.org", EMSK, 32)
```

If an inner method does not support export of an Extended Master Session Key (EMSK), then IMSK is the MSK of the inner method as per [RFC7170] Section 5.2.

For MSK and EMSK, TEAP [RFC7170] uses an inner tunnel EMSK to calculate the outer EMSK. As such, those key derivations cannot use the above derivation.

The other key derivations for TEAP are given here. All derivations not given here are the same as given above in the previous section. These derivations are also used for FAST, but using the FAST Type.

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

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

Where | denotes concatenation. The outer MSK and EMSK are then derived from the above definitions, as:

```
MSK = TLS-Exporter("EXPORTER: Session Key Generating Function",
                   S-IMCK[j], 64)

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

The TEAP Compound MAC defined in [RFC7170] Section 5.3 is updated to use the definition of CMK[j] given above, which then leads to the following definition

```
CMK = CMK[j]

Compound-MAC = MAC( CMK, BUFFER )
```

where j is the number of the last successfully executed inner EAP method. For TLS 1.3, the message authentication code (MAC) is computed with the HMAC algorithm negotiated for HKDF in the key schedule, as per section 7.1 of RFC 8446. The definition of BUFFER is unchanged from [RFC7170] Section 5.3

### 2.3. FAST

For FAST, the session\_key\_seed is also part of the key\_block, as defined in [RFC4851] Section 5.1.

The definition of S-IMCK[n], MSK, and EMSK are the same as given above for TEAP. We reiterate that the EAP-FAST Type must be used when deriving the session\_key\_seed, and not the TEAP Type.

Unlike [RFC4851] Section 5.2, the definition of IMCK[j] places the reference to S-IMCK after the textual label, and the concatenates the

IMSK instead of MSK.

EAP-FAST previously used a PAC, which is a session ticket that contains a pre-shared key (PSK) along with other data. As TLS 1.3 allows session resumption using a PSK, the use of a PAC is deprecated for EAP-FAST in TLS 1.3. PAC provisioning [RFC5422] is also no longer part of EAP-FAST when TLS 1.3 is used.

The T-PRF given in [RFC4851] Section 5.5 is not used for TLS 1.3. Instead, it is replaced with the TLS 1.3 TLS-Exporter function.

#### 2.4. TTLS

[RFC5281] Section 11.1 defines an implicit challenge when the inner methods of CHAP [RFC1994], MS-CHAP [RFC2433], or MS-CHAPv2 [RFC2759] are used. The derivation for TLS 1.3 is instead given as

```
EAP-TTLS_challenge = TLS-Exporter("ttls challenge",, n)
```

There is no "context\_value" ([RFC8446] Section 7.5) passed to the TLS-Exporter function. The value "n" given here is the length of the data required, which [RFC5281] requires it to be 17 octets for CHAP (Section 11.2.2) and MS-CHAP-V2 (Section 11.2.4), and to be 9 octets for MS-CHAP (Section 11.2.3).

Note that unlike TLS 1.2 and earlier, the calculation of TLS-Exporter depends on the length passed to it. Implementations therefore MUST pass the correct length instead of passing a large length and truncating the output. Any output calculated using a larger length value, and which is then truncated, will be different from the output which was calculated using the correct length.

##### 2.4.1. Client Certificates

[RFC5281] Section 7.6 permits "Authentication of the client via client certificate during phase 1, with no additional authentication or information exchange required.". This practice is forbidden when TTLS is used with TLS 1.3. If there is a requirement to use client certificates with no inner tunnel methods, then EAP-TLS should be used instead of TTLS.

The use of client certificates is still permitted when using TTLS with TLS 1.3. However, if the client certificate is accepted, then the EAP peer MUST proceed with additional authentication of Phase 2, as per [RFC5281] Section 7.2 and following. If there is no Phase 2 data, then the EAP server MUST reject the session.

## 2.5. PEAP

When PEAP uses crypto binding, it uses a different key calculation defined in [PEAP-MPPE] which consumes inner method keying material. The pseudo-random function (PRF+) used here is not taken from the TLS exporter, but is instead calculated via a different method which is given in [PEAP-PRF]. That derivation remains unchanged in this specification.

However, the key calculation uses a PEAP Tunnel Key [PEAP-TK] which is defined as:

```
... 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).
```

We note that this text does not define Key\_Material. Instead, it defines TK as the first octets of Key\_Material, and gives a definition of Key\_Material which is appropriate for TLS versions before TLS 1.3.

For TLS 1.3, the TK should be derived from the Key\_Material defined above in Section 2.1, instead of using the TLS-PRF-128 derivation given above.

### 2.5.1. Client Certificates

As with TTLS, [PEAP] permits the use of client certificates in addition to inner tunnel methods.

The use of client certificates is still permitted when using PEAP with TLS 1.3. However, if the client certificate is accepted, then the EAP peer MUST proceed with additional authentication of the inner tunnel. If there is no inner tunnel authentication data, then the EAP server MUST reject the session.

## 3. Application Data

Unlike previous TLS versions, TLS 1.3 can continue negotiation after the initial TLS handshake has been completed, which TLS 1.3 calls the "CONNECTED" state. Some implementations use a "TLS finished" determination as an indication that TLS negotiation has completed, and that an "inner tunnel" session can now be negotiated. This assumption is not always correct with TLS 1.3.

Earlier TLS versions did not always send application data along with the "TLS finished" method. It was then possible for implementations to assume that a transition to "TLS finished" also meant that there

was no application data available, and that another round trip was required. This assumption is not true with TLS 1.3, and applications relying on that behavior will not operate correctly with TLS 1.3.

As a result, implementations MUST check for application data once the TLS session has been established. This check MUST be performed before proceeding with another round trip of TLS negotiation. If application data is available, it MUST be processed according to the relevant resumption and/or EAP type.

TLS 1.3 also permits NewSessionTicket messages to be sent before the TLS "Finished", and after application data is sent. This change can cause many implementations to fail in a number of different ways, due to a reliance on implicit behavior seen in earlier TLS versions.

In order to correct this failure, we require that if the underlying TLS connection is still performing negotiation, then implementations MUST NOT send, or expect to receive application data in the TLS session. Implementations MUST delay processing of application data until such time as the TLS negotiation has finished. If the TLS negotiation is successful, then the application data can be examined. If the TLS negotiation is unsuccessful, then the application data is untrusted, and therefore MUST be discarded without being examined.

The default for many TLS library implementations is to send a NewSessionTicket message immediately after, or along with, the TLS Finished message. This ticket could be used for resumption, even if the "inner tunnel" authentication has not been completed. If the ticket could be used, then it could allow a malicious EAP peer to completely bypass the "inner tunnel" authentication.

Therefore, the EAP server MUST NOT permit any session ticket to successfully resume authentication, unless the inner tunnel authentication has completed successfully. The alternative would allow an attacker to bypass authentication by obtaining a session ticket, and then immediately closing the current session, and "resuming" using the session ticket.

To protect against that attack, implementations SHOULD NOT send NewSessionTicket messages until the "inner tunnel" authentication has completed. There is no reason to send session tickets which will later be invalidated or ignored. However, we recognize that this suggestion may not always be possible to implement with some available TLS libraries. As such, EAP servers MUST take care to either invalidate or discard session tickets which are associated with sessions that terminate in EAP Failure.

The NewSessionTicket message SHOULD also be sent along with other

application data, if possible. Sending that message alone prolongs the packet exchange to no benefit.

[RFC9190] Section 2.5 requires a protected result indicator which indicates that TLS negotiation has finished. Methods which use "inner tunnel" methods MUST instead begin their "inner tunnel" negotiation by sending Type-specific application data.

### 3.1. Identities

[RFC9190] Sections 2.1.3 and 2.1.7 recommend the use of anonymous Network Access Identifiers (NAIs) [RFC7542] in the EAP Identity Response packet. However, as EAP-TLS does not send application data inside of the TLS tunnel, that specification does not address the subject of "inner" identities in tunneled EAP methods. This subject must, however, be addressed for the tunneled methods.

Using an anonymous NAI as per [RFC7542] Section 2.4 has two benefits. First, an anonymous identity makes it more difficult to track users. Second, an NAI allows the EAP session to be routed in an AAA framework as described in [RFC7542] Section 3.

For the purposes of tunneled EAP methods, we can therefore view the outer TLS layer as being mainly a secure transport layer. That transport layer is responsible for getting the actual (inner) authentication credentials securely from the EAP peer to the EAP server. As the outer identity is often used as an anonymous routing identifier for AAA ([RFC7542] Section 3), there is little reason for it to be the same as the inner identity. We therefore have a few recommendations on the inner identity, and its relationship to the outer identity.

For the purpose of this section, we define the inner identity as the identification information carried inside of the TLS tunnel. For PEAP, that identity may be an EAP Response Identity. For TTLS, it may be the User-Name attribute. Vendor-specific EAP methods which use TLS will generally also have an inner identity.

Implementations MUST NOT use anonymous identities for the inner identity. If anonymous network access is desired, eap peers MUST use EAP-TLS without peer authentication, as per [RFC9190] section 2.1.5. EAP servers MUST cause authentication to fail if an EAP peer uses an anonymous "inner" identity for any TLS-based EAP method.

Implementations SHOULD NOT use inner identities which contain an NAI realm. The outer identity contains an NAI realm, which ensures that the inner authentication method is routed to the correct destination. As such, any NAI realm in the inner identity is almost always

redundant.

However, if the inner identity does contain an NAI realm, the inner realm SHOULD be either an exact copy of the outer realm, or be a subdomain of the outer realm. The inner realm SHOULD NOT be from a different realm than the outer realm. There are very few reasons for those realms to be different.

In general, routing identifiers should be associated with the authentication data that they are routing. For example, if a user has an inner identity of "user@example.com", then it generally makes no sense to have an outer identity of "@example.org". The authentication request would then be routed to the "example.org" domain, which may have no idea what to do with the credentials for "user@example.com". At best, the authentication request would be discarded. At worst, the "example.org" domain could harvest user credentials for later use in attacks on "example.com".

In addition, associating disparate inner/outer identities in the same EAP authentication session means that otherwise unrelated realms are tied together, which can make networks more fragile.

For example, an organization which uses a "hosted" AAA provider may choose to use the realm of the AAA provider as the outer identity. The inner identity can then be fully qualified: user name plus realm of the organization. This practice can result in successful authentications, but it has difficulties.

Other organizations may host their own AAA servers, but use a "cloud" identity provider to hold user accounts. In that situation, the organizations may use their own realm as the outer (routing) identity, then use an identity from the "cloud" provider as the inner identity. This practice is NOT RECOMMENDED. User accounts for an organization should be qualified as belonging to that organization, and not to an unrelated third party.

Both of these practices mean that changing "cloud" providers is difficult. When such a change happens, each individual supplicant must be updated with a different outer identity which points to the new "cloud" provider. This process can be expensive, and some supplicants may not be online when this changeover happens. The result could be devices or users who are unable to obtain network access, even if all relevant network systems are online and functional.

Further, standards such as [RFC7585] allow for dynamic discovery of home servers for authentication. That specification has been widely deployed, and means that there is minimal cost to routing



authentication to a particular domain. The authentication can also be routed to a particular identity provider, and changed at will, with no loss of functionality. That specification is also scalable, in that it does not require changes to many systems when a domain updates its configuration. Instead, only one thing has to change: the configuration of that domain. Everything else is discovered dynamically.

That is, changing the configuration for one domain is significantly simpler and more scalable than changing the configuration for potentially millions of end-user devices.

We recognize that there may be existing use-cases where the inner and outer identities use different realms. As such, we cannot forbid that practice. We hope that the discussion above shows not only why such practices are problematic, but also that it shows how alternative methods are more flexible, more scalable, and are easier to manage.

#### 4. Resumption

[RFC9190] Section 2.1.3 defines the process for resumption. This process is the same for all TLS-based EAP types. The only practical difference is that the value of the Type field is different.

Note that if resumption is performed, then the EAP server MUST send the protected success result indicator (one octet of 0x00) inside the TLS tunnel as per [RFC9190]. If either peer or server instead initiates an inner tunnel method, then that method MUST be followed, and resumption MUST NOT be used. The EAP peer MUST in turn check for the existence the protected success result indicator (one octet of 0x00), and cause authentication to fail if that octet is not received.

All TLS-based EAP methods support resumption, as it is a property of the underlying TLS protocol. All EAP servers and peers MUST support resumption for all TLS-based EAP methods. We note that EAP servers and peers can still choose to not resume any particular session. For example, EAP servers may forbid resumption for administrative, or other policy reasons.

It is RECOMMENDED that EAP servers and peers enable resumption, and use it where possible. The use of resumption decreases the number of round trips used for authentication. This decrease leads to lower latency for authentications, and less load on the EAP server. Resumption can also lower load on external systems, such as databases which contain user credentials.

As the packet flows for resumption are essentially identical across all TLS-based EAP types, it is technically possible to authenticate using EAP-TLS (Type 13), and then perform resumption using another EAP type, just as EAP-TTLS (Type 21). However, there is no practical benefit to doing so. It is also not clear what this behavior would mean, or what (if any) security issues there may be with it. As a result, this behavior is forbidden.

EAP servers therefore MUST NOT resume sessions across different EAP Types, and EAP servers MUST reject resumptions in which the EAP Type value is different from the original authentication.

## 5. Implementation Status

TTLS and PEAP are implemented and tested to be inter-operable with wpa\_supplicant 2.10 and Windows 11 as clients, and FreeRADIUS 3.0.26 and Radiator as RADIUS servers.

The wpa\_supplicant implementation requires that a configuration flag be set "tls\_disable\_tlsv1\_3=0", and describes the flag as "enable TLSv1.3 (experimental - disabled by default)". However, interoperability testing shows that PEAP and TTLS both work with Radiator and FreeRADIUS.

Implementors have demonstrated significant interest in getting PEAP and TTLS working for TLS 1.3, but less interest in EAP-FAST and TEAP. As such, there is no implementation experience with EAP-FAST or TEAP. However, we believe that the definitions described above are correct, and are workable.

## 6. Security Considerations

[RFC9190] Section 5 is included here by reference.

Updating the above EAP methods to use TLS 1.3 is of high importance for the Internet Community. Using the most recent security protocols can significantly improve security and privacy of a network.

In some cases, client certificates are not used for TLS-based EAP methods. In those cases, the user is authenticated only after successful completion of the inner tunnel authentication. However, the TLS protocol may send one or more NewSessionTicket after receiving the TLS Finished message from the client, and therefore before the user is authenticated.

This separation of data allows for a "time of use, time of check" security issue. Malicious clients can begin a session and receive a NewSessionTicket. The malicious client can then abort the

authentication session, and the obtained NewSessionTicket to "resume" the previous session.

As a result, EAP servers MUST NOT permit sessions to be resumed until after authentication has successfully completed. This requirement may be met in a number of ways. For example, by not caching the session ticket until after authentication has completed, or by marking up the cached session ticket with a flag stating whether or not authentication has completed.

For PEAP, some derivations use HMAC-SHA1 [PEAP-MPPE]. In the interests of interoperability and minimal changes, we do not change that derivation, as there are no known security issues with HMAC-SHA1. Further, the data derived from the HMAC-SHA1 calculations is exchanged inside of the TLS tunnel, and is visible only to users who have already successfully authenticated. As such, the security risks are minimal.

#### 6.1. Protected Success and Failure indicators

[RFC9190] provides for protected success and failure indicators as discussed in Section 4.1.1 of [RFC4137]. These indicators are provided for both full authentication, and for resumption.

Other TLS-based EAP methods provide these indicators only for resumption.

For full authentication, the other TLS-based EAP methods do not provide for protected success and failure indicators as part of the outer TLS exchange. That is, the protected result indicator is not used, and there is no TLS-layer alert sent when the inner authentication fails. Instead, there is simply either an EAP-Success or EAP-Failure sent. This behavior is the same as for previous TLS versions, and therefore introduces no new security issues.

We note that most TLS-based EAP methods provide for success and failure indicators as part of the authentication exchange performed inside of the TLS tunnel. These indicators are therefore protected, as they cannot be modified or forged.

However, some inner methods do not provide for success or failure indicators. For example, the use of TTLS with inner PAP or CHAP. Those methods send authentication credentials to the server via the inner tunnel, with no method to signal success or failure inside of the tunnel.

There are functionally equivalent authentication methods which can be used to provide protected indicators. PAP can often be replaced with

EAP-GTC, and CHAP with EAP-MD5. Both replacement methods provide for similar functionality, and have protected success and failure indicator. The main cost to this change is additional round trips.

It is RECOMMENDED that implementations deprecate inner tunnel methods which do not provided protected success and failure indicators. Implementations SHOULD use EAP-GTC instead of PAP, and EAP-MD5 instead of CHAP. New TLS-based EAP methods MUST provide protected success and failure indicators inside of the TLS tunnel.

When the inner authentication protocol indicates that authentication has failed, then implementations MUST fail authentication for the entire session. There MAY be additional protocol exchanges in order to exchange more detailed failure indicators, but the final result MUST be a failed authentication. As noted earlier, any session tickets for this failed authentication MUST be either invalidated or discarded.

Similarly, when the inner authentication protocol indicates that authentication has succeed, then implementations SHOULD cause authentication to succeed for the entire session. There MAY be additional protocol exchanges in order which could cause other failures, so success is not required here.

In both of these cases, the EAP server MUST send an EAP-Failure or EAP-Success message, as indicated by Section 2, item 4 of [RFC3748]. Even though both parties have already determined the final authentication status, the full EAP state machine must still be followed.

## 7. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the TLS-based EAP methods for TLS 1.3 protocol in accordance with [RFC8126].

This memo requires IANA to add the following labels to the TLS Exporter Label Registry defined by [RFC5705]. These labels are used in the derivation of Key\_Material and Method-Id as defined above in Section 2.

The labels below need to be added to the "TLS Exporter Labels" registry. These labels are used only for TEAP.

- \* EXPORTER: session key seed
- \* EXPORTER: Inner Methods Compound Keys
- \* EXPORTER: Session Key Generating Function
- \* EXPORTER: Extended Session Key Generating Function

\* TEAPbindkey@ietf.org

## 8. References

### 8.1. Normative References

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[RFC3748]

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

[RFC5216]

Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008

[RFC5247]

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

[RFC5705]

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

[RFC7170]

Zhou, H., et al., "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, May 2014.

[RFC8126]

Cotton, M., et al., "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 8126, June 2017.

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

[RFC8446]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

[RFC9190]

Mattsson, J., and Sethi, M., "Using EAP-TLS with TLS 1.3", draft-

ietf-emu-eap-tls13-18, July 2021.

[IANA]

<https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml#eap-numbers-4>

## 8.2. Informative References

[MSPEAP]

<https://msdn.microsoft.com/en-us/library/cc238354.aspx>

[PEAP]

Palekar, A. et al, "Protected EAP Protocol (PEAP)", draft-josefsson-pppext-eap-tls-eap-06.txt, May 2003.

[PEAP-MPPE]

[https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/MS-PEAP/e75b0385-915a-4fc3-a549-fd3d06b995b0](https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/e75b0385-915a-4fc3-a549-fd3d06b995b0)

[PEAP-PRF]

[https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/MS-PEAP/0de54161-0bd3-424a-9b1a-854b4040a6df](https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/0de54161-0bd3-424a-9b1a-854b4040a6df)

[PEAP-TK]

[https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/MS-PEAP/41288c09-3d7d-482f-a57f-e83691d4d246](https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/41288c09-3d7d-482f-a57f-e83691d4d246)

[RFC1994]

Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[RFC2433]

Zorn, G. and Cobb, S., "Microsoft PPP CHAP Extensions", RFC 2433, October 1998.

[RFC2759]

Zorn, G., "Microsoft PPP CHAP Extensions, Version 2", RFC 2759, January 2000.

[RFC4137]

Vollbrecht, J., et al, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator ", RFC 4137, August 2005.

[RFC4851]

Cam-Winget, N., et al, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

[RFC5281]

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

[RFC5422]

Cam-Winget, N., et al, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, March 2009.

[RFC7542]

DeKoK, A, "The Network Access Identifier", RFC 7542, May 2015.

[RFC7585]

Winter, S, and McCauley, M., "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, October 2015.

#### Acknowledgments

Thanks to Jorge Vergara for a detailed review of the requirements for various EAP types.

Thanks to Jorge Vergara, Bruno Periera Vidal, Alexander Clouter, Karri Huhtanen, and Heikki Vatiainen for reviews of this document, and for assistance with interoperability testing.

#### Authors' Addresses

Alan DeKok  
The FreeRADIUS Server Project  
  
Email: [aland@freeradius.org](mailto:aland@freeradius.org)

