

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2020

T. Aura
Aalto University
M. Sethi
Ericsson
October 29, 2019

Nimble out-of-band authentication for EAP (EAP-NOOB)
draft-aura-eap-noob-07

Abstract

Extensible Authentication Protocol (EAP) provides support for multiple authentication methods. This document defines the EAP-NOOB authentication method for nimble out-of-band (OOB) authentication and key derivation. This EAP method is intended for bootstrapping all kinds of Internet-of-Things (IoT) devices that have a minimal user interface and no pre-configured authentication credentials. The method makes use of a user-assisted one-directional OOB channel between the peer device and authentication server.

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 May 1, 2020.

Copyright Notice

Copyright (c) 2019 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
2. Terminology	4
3. EAP-NOOB protocol	5
3.1. Protocol overview	5
3.2. Protocol messages and sequences	8
3.2.1. Common handshake in all EAP exchanges	8
3.2.2. Initial Exchange	10
3.2.3. OOB Step	11
3.2.4. Completion Exchange	13
3.2.5. Waiting Exchange	15
3.3. Protocol data fields	16
3.3.1. Peer identifier, realm and NAI	16
3.3.2. Message data fields	18
3.4. Fast reconnect and rekeying	23
3.4.1. Persistent EAP-NOOB association	23
3.4.2. Reconnect Exchange	24
3.4.3. User reset	27
3.5. Key derivation	28
3.6. Error handling	31
3.6.1. Invalid messages	33
3.6.2. Unwanted peer	33
3.6.3. State mismatch	33
3.6.4. Negotiation failure	33
3.6.5. Cryptographic verification failure	34
3.6.6. Application-specific failure	34
4. IANA Considerations	35
4.1. Cryptosuites	35
4.2. Message Types	35
4.3. Error codes	36
4.4. Domain name reservation considerations	37
5. Implementation Status	38
5.1. Implementation with wpa_supplicant and hostapd	38
5.2. Implementation on Contiki	39
5.3. Protocol modeling	39
6. Security considerations	39
6.1. Authentication principle	39
6.2. Identifying correct endpoints	41
6.3. Trusted path issues and misbinding attacks	42
6.4. Peer identifiers and attributes	43
6.5. Identity protection	43
6.6. Downgrading threats	44

6.7. Recovery from loss of last message	45
6.8. EAP security claims	46
7. References	48
7.1. Normative references	48
7.2. Informative references	49
Appendix A. Exchanges and events per state	51
Appendix B. Application-specific parameters	52
Appendix C. ServerInfo and PeerInfo contents	53
Appendix D. EAP-NOOB roaming	55
Appendix E. OOB message as URL	56
Appendix F. Example messages	57
Appendix G. TODO list	59
Appendix H. Version history	59
Appendix I. Acknowledgments	62
Authors' Addresses	62

1. Introduction

This document describes a method for registration, authentication and key derivation for network-connected ubiquitous computing devices, such as consumer and enterprise appliances that are part of the Internet of Things (IoT). These devices may be off-the-shelf hardware that is sold and distributed without any prior registration or credential-provisioning process. Thus, the device registration in a server database, ownership of the device, and the authentication credentials for both network access and application-level security must all be established at the time of the device deployment. Furthermore, many such devices have only limited user interfaces that could be used for their configuration. Often, the interfaces are limited to either only input (e.g. camera) or output (e.g. display screen). The device configuration is made more challenging by the fact that the devices may exist in large numbers and may have to be deployed or re-configured nimbly based on user needs.

More specifically, the devices may have the following characteristics:

- o no pre-established relation with a specific server or user,
- o no pre-provisioned device identifier or authentication credentials,
- o limited user interface and configuration capabilities.

Many proprietary OOB configuration methods exists for specific IoT devices. The goal of this specification is to provide an open standard and a generic protocol for bootstrapping the security of network-connected appliances, such as displays, printers, speakers,

and cameras. The security bootstrapping in this specification makes use of a user-assisted out-of-band (OOB) channel. The device authentication relies on user having physical access to the device, and the of the key exchange security is based on the assumption that attackers are not able to observe or modify the messages conveyed through the OOB channel. We follow the common approach taken in pairing protocols: performing a Diffie-Hellman key exchange over the insecure network and authenticating the established key with the help of the OOB channel in order to prevent impersonation and man-in-the-middle (MitM) attacks.

The solution presented here is intended for devices that have either an input or output interface, such as a camera, microphone, display screen, speakers or blinking LED light, which is able to send or receive dynamically generated messages of tens of bytes in length. Naturally, this solution may not be appropriate for very small sensors or actuators that have no user interface at all or for devices that are inaccessible to the user. We also assume that the OOB channel is at least partly automated (e.g. camera scanning a bar code) and, thus, there is no need to absolutely minimize the length of the data transferred through the OOB channel. This differs, for example, from Bluetooth simple pairing [BluetoothPairing], where it is critical to minimize the length of the manually transferred or compared codes. Since the OOB messages are dynamically generated, we do not support static printed registration codes. This also prevents attacks where a static secret code would be leaked.

2. Terminology

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

In addition, this document frequently uses the following terms as they have been defined in [RFC5216]:

authenticator The entity initiating EAP authentication.

peer The entity that responds to the authenticator. In [IEEE-802.1X], this entity is known as the supplicant.

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.

3. EAP-NOOB protocol

This section defines the EAP-NOOB protocol. The protocol is a generalized version of the original idea presented by Sethi et al. [Sethi14].

3.1. Protocol overview

One EAP-NOOB protocol execution spans multiple EAP conversations, called Exchanges. This is necessary to leave time for the OOB message to be delivered, as will be explained below.

The overall protocol starts with the Initial Exchange, in which the server allocates an identifier to the peer, and the server and peer negotiate the protocol version and cryptosuite (i.e. cryptographic algorithm suite), exchange nonces and perform an Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange. The user-assisted OOB Step then takes place. This step requires only one out-of-band message either from the peer to the server or from the server to the peer. While waiting for the OOB Step action, the peer MAY probe the server by reconnecting to it with EAP-NOOB. If the OOB Step has already taken place, the probe leads to the Completion Exchange, which completes the mutual authentication and key confirmation. On the other hand, if the OOB Step has not yet taken place, the probe leads to the Waiting Exchange, and the peer will perform another probe after a server-defined minimum waiting time. The Initial Exchange and Waiting Exchange always end in EAP-Failure, while the Completion Exchange may result in EAP-Success. Once the peer and server have performed a successful Completion Exchange, both endpoints store the created association in persistent storage, and the OOB Step is not repeated. Thereafter, creation of new temporal keys, ECDHE rekeying, and updates of cryptographic algorithms can be achieved with the Reconnect Exchange.

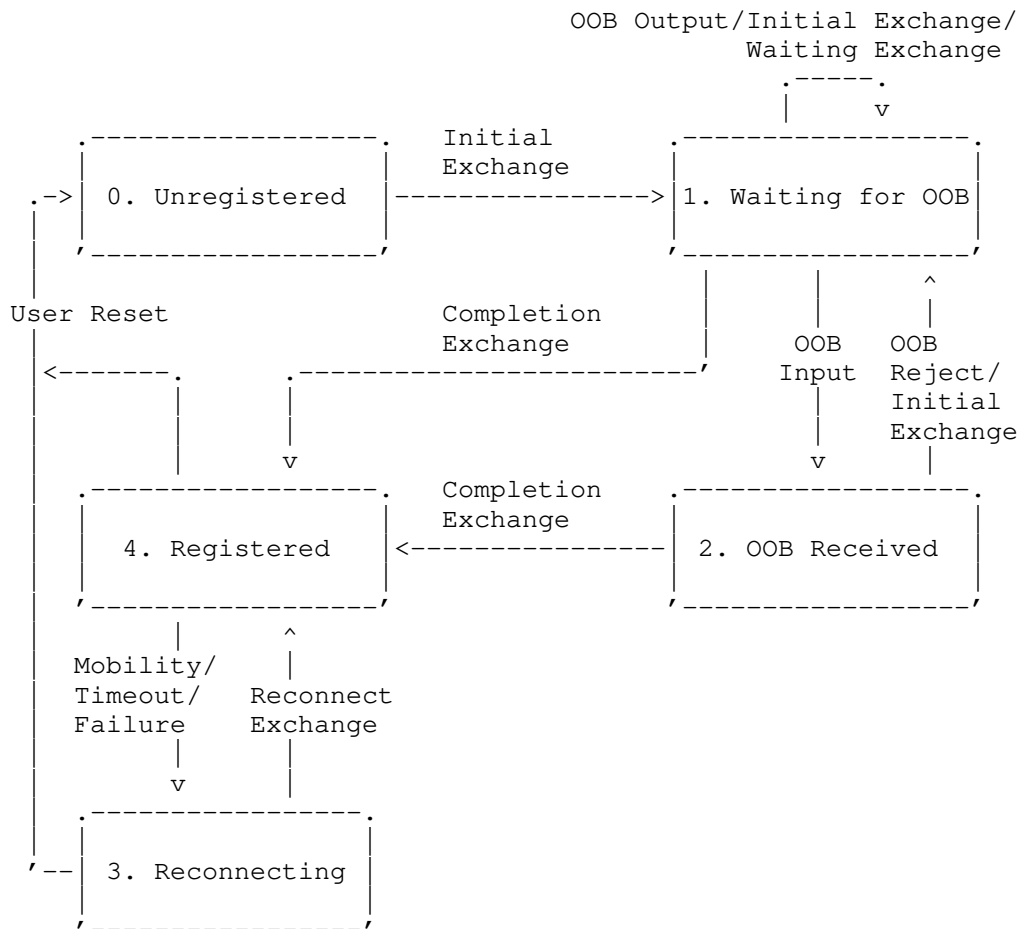


Figure 1: EAP-NOOB server-peer association state machine

Figure 1 shows the association state machine, which is the same for the server and for the peer. (For readability, only the main state transitions are shown. The complete table of transitions can be found in Appendix A.) When the peer initiates the EAP-NOOB method, the server chooses the ensuing message exchange based on the combination of the server and peer states. The EAP server and peer are initially in the Unregistered state, in which no state information needs to be stored. Before a successful Completion Exchange, the server-peer association state is ephemeral in both the server and peer (ephemeral states 0..2), and either endpoint may cause the protocol to fall back to the Initial Exchange. After the Completion Exchange has resulted in EAP-Success, the association

state becomes persistent (persistent states 3..4). Only user reset or memory failure can cause the return of the server or the peer from the persistent states to the ephemeral states and to the Initial Exchange.

The server MUST NOT repeat a successful OOB Step with the same peer except if the association with the peer is explicitly reset by the user or lost due to failure of the persistent storage in the server. More specifically, once the association has entered the Registered state, the server MUST NOT delete the association or go back to states 0..2 without explicit user approval. Similarly, the peer MUST NOT repeat the OOB Step unless the user explicitly deletes from the peer the association with the server or resets the peer to the Unregistered state. The server and peer MAY implement user reset of the association by deleting the state data from that endpoint. If an endpoint continues to store data about the association after the user reset, its behavior SHOULD be equivalent to having deleted the association data.

It can happen that the peer accidentally or through user reset loses its persistent state and reconnects to the server without a previously allocated peer identifier. In that case, the server MUST treat the peer as a new peer. The server MAY use auxiliary information, such as the PeerInfo field received in the Initial Exchange, to detect multiple associations with the same peer. However, it MUST NOT delete or merge redundant associations without user or application approval because EAP-NOOB internally has no secure way of verifying that the two peers are the same physical device. Similarly, the server might lose the association state because of a memory failure or user reset. In that case, the only way to recover is that the user resets also the peer.

A special feature of the EAP-NOOB method is that the server is not assumed to have any a-priori knowledge of the peer. Therefore, the peer initially uses the generic identity string "noob@eap-noob.net" as its network access identifier (NAI). The server then allocates a server-specific identifier to the peer. The generic NAI serves two purposes: firstly, it tells the server that the peer supports and expects the EAP-NOOB method and, secondly, it allows routing of the EAP-NOOB sessions to a specific authentication server in the AAA architecture.

EAP-NOOB is an unusual EAP method in that the peer has to have multiple EAP conversations with the server before it can receive EAP-Success. The reason is that, while EAP allows delays between the request-response pairs, e.g. for repeated password entry, the user delays in OOB authentication can be much longer than in password trials. In particular, EAP-NOOB supports also peers with no input

capability in the user interface. Since user cannot initiate the protocol in these devices, they have to perform the Initial Exchange opportunistically and hope for the OOB Step to take place within a timeout period (NoobTimeout), which is why the timeout needs to be several minutes rather than seconds. For example, consider a printer (peer) that outputs the OOB message on paper, which is then scanned for the server. To support such high-latency OOB channels, the peer and server perform the Initial Exchange in one EAP conversation, then allow time for the OOB message to be delivered, and later perform the Waiting and Completion Exchanges in different EAP conversations.

3.2. Protocol messages and sequences

This section defines the EAP-NOOB exchanges, which correspond to EAP conversations. The exchanges start with a common handshake, which determines the type of the following exchange. The common handshake messages and the subsequent messages for each exchange type are listed in the diagrams below. The diagrams also specify the data members present in each message. Each exchange comprises multiple EAP requests-response pairs and ends in either EAP-Failure, indicating that authentication is not (yet) successful, or in EAP-Success.

3.2.1. Common handshake in all EAP exchanges

All EAP-NOOB exchanges start with common handshake messages. The handshake starts with the identity request and response that are common to all EAP methods. Their purpose is to enable the AAA architecture to route the EAP conversation to the EAP server and to enable the EAP server to select the EAP method. The handshake then continues with one EAP-NOOB request-response pair in which the server discovers the peer identifier used in EAP-NOOB and the peer state.

In more detail, each EAP-NOOB exchanges begin with the authenticator sending an EAP-Request/Identity packet to the peer. From this point on, the EAP conversation occurs between the server and the peer, and the authenticator acts as a pass-through device. The peer responds to the authenticator with an EAP-Response/Identity packet, which contains the network access identifier (NAI). The authenticator, acting as a pass-through device, forwards this response and the following EAP conversation between the peer and the AAA architecture. The AAA architecture routes the conversation to a specific AAA server (called "EAP server" or simply "server" in this specification) based on the realm part of the NAI. The server selects the EAP-NOOB method based on the user part of the NAI, as defined in Section 3.3.1.

After receiving the EAP-Response/Identity message, the server sends the first EAP-NOOB request (Type=9) to the peer, which responds with

the peer identifier (PeerId) and state (PeerState) in the range 0..3. However, the peer SHOULD omit the PeerId from the response (Type=9) when PeerState=0. The server then chooses the EAP-NOOB exchange, i.e. the ensuing message sequence, as explained below. The peer recognizes the exchange based on the message type field (Type) of the next EAP-NOOB request received from the server.

The server determines the exchange type based on the combination of the peer and server states as follows (also summarized in Figure 11). If one of the peer and server is in the Unregistered (0) state and the other is in one of the ephemeral states (0..2), the server chooses the Initial Exchange. If one of the peer or server is in the OOB Received (2) state and the other is either in the Waiting for OOB (1) or OOB Received (2) state, the OOB Step has taken place and the server chooses the Completion Exchange. If both the server and peer are in the Waiting for OOB (1) state, the server chooses the Waiting Exchange. If the peer is in the Reconnecting (3) state and the server is in the Registered (4) or Reconnecting (3) state, the server chooses the Reconnect Exchange. All other state combinations are error situations where user action is required, and the server indicates such errors to the peer with the error code 2002 (see Section 3.6.3). Note also that the peer MUST NOT initiate EAP-NOOB when the peer is in Registered (4) state.

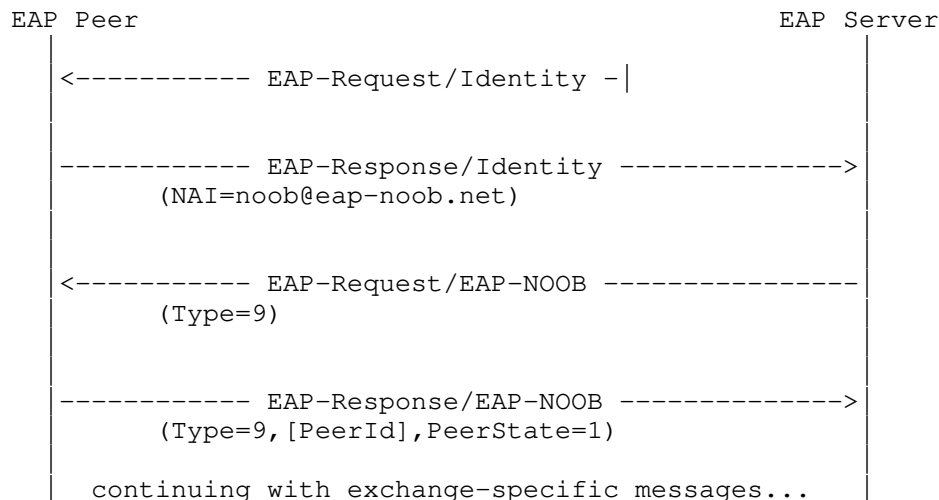


Figure 2: Common handshake in all EAP-NOOB exchanges

3.2.2. Initial Exchange

The Initial Exchange comprises the common handshake and two further EAP-NOOB request-response pairs, one for version, cryptosuite and parameter negotiation and the other for the ECDHE key exchange. The first EAP-NOOB request (Type=1) from the server contains a newly allocated PeerId for the peer and an optional Realm. The server allocates a new PeerId in the Initial Exchange regardless of any old PeerId in the username part of the received NAI. The server also sends in the request a list of the protocol versions (Vers) and cryptosuites (Cryptosuites) it supports, an indicator of the OOB channel directions it supports (Dirs), and a ServerInfo object. The peer chooses one of the versions and cryptosuites. The peer sends a response (Type=1) with the selected protocol version (Verp), the received PeerId, the selected cryptosuite (Cryptosuitep), an indicator of the OOB channel directions selected by the peer (Dirp), and a PeerInfo object. In the second EAP-NOOB request and response (Type=2), the server and peer exchange the public components of their ECDHE keys and nonces (PKs,Ns,PKp,Np). The ECDHE keys MUST be based on the negotiated cryptosuite i.e. Cryptosuitep. The Initial Exchange always ends with EAP-Failure from the server because the authentication cannot yet be completed.

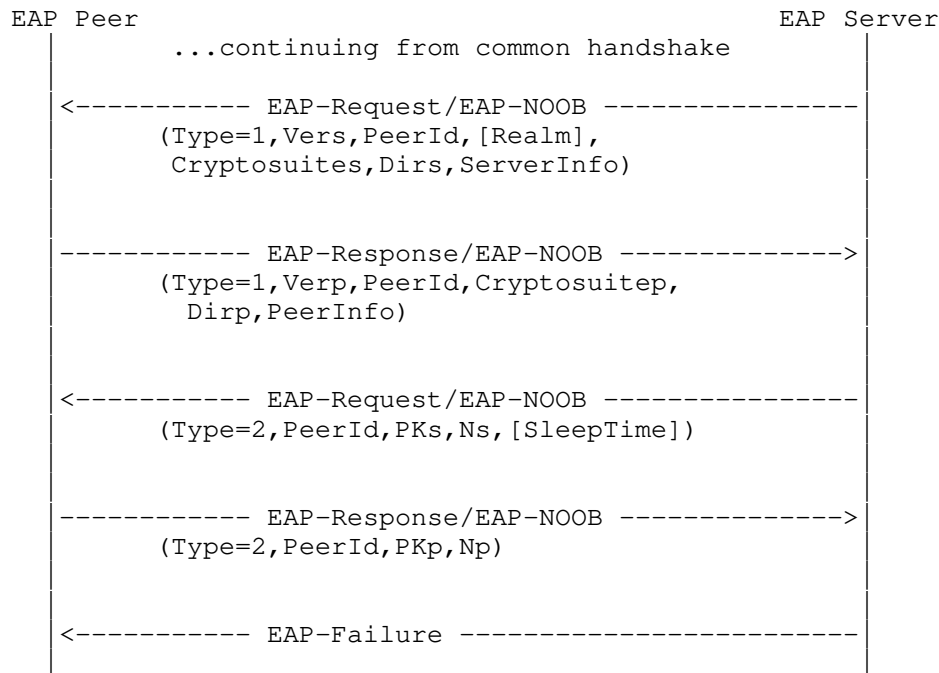


Figure 3: Initial Exchange

At the conclusion of the Initial Exchange, both the server and the peer move to the Waiting for OOB (1) state.

3.2.3. OOB Step

The OOB Step, labeled as OOB Output and OOB Input in Figure 1, takes place after the Initial Exchange. Depending on the negotiated OOB channel direction, the peer or the server outputs the OOB message shown in Figure 4 or Figure 5, respectively. The data fields are the PeerId, the secret nonce Noob, and the cryptographic fingerprint Hoob. The contents of the data fields are defined in Section 3.3.2. The OOB message is delivered to the other endpoint via a user-assisted OOB channel.

For brevity, we will use the terms OOB sender and OOB receiver in addition to the already familiar EAP server and EAP peer. If the OOB message is sent in the server-to-peer direction, the OOB sender is the server and the OOB receiver is the peer. On the other hand, if the OOB message is sent in the peer-to-server direction, the OOB sender is the peer and the OOB receiver is the server.

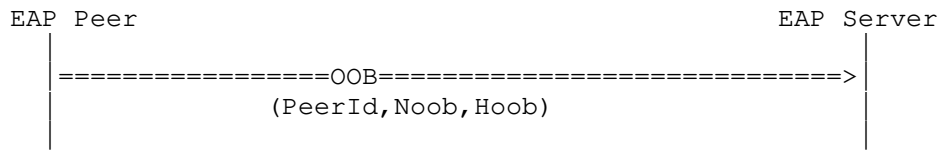


Figure 4: OOB Step, from peer to EAP server

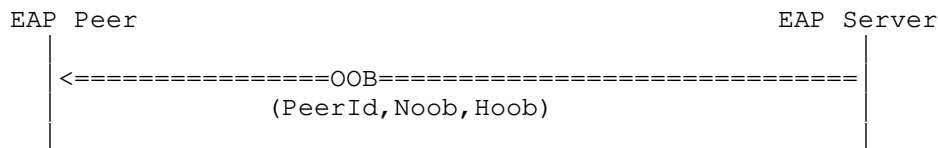


Figure 5: OOB Step, from EAP server to peer

The OOB receiver **MUST** compare the received value of the fingerprint Hoob with a value that it computes locally. If the values are equal, the receiver moves to the OOB Received (2) state. Otherwise, the receiver **MUST** reject the OOB message. For usability reasons, the OOB receiver **SHOULD** indicate the acceptance or rejection of the OOB message to the user. The receiver **SHOULD** reject invalid OOB messages without changing its state, until an application-specific number of invalid messages (OobRetries) has been reached, after which the receiver **SHOULD** consider it an error and go back to the Unregistered (0) state.

The server or peer **MAY** send multiple OOB messages with different Noob values while in the Waiting for OOB (1) state. The OOB sender **SHOULD** remember the Noob values until they expire and accept any one of them in the following Completion Exchange. The Noob values sent by the server expire after an application-dependent timeout (NoobTimeout), and the server **MUST NOT** accept Noob values older than that in the Completion Exchange. The **RECOMMENDED** value for NoobTimeout is 3600 seconds if there are no application-specific reasons for making it shorter or longer. The Noob values sent by the peer expire as defined in Section 3.2.5.

The OOB receiver does not accept further OOB messages after it has accepted one and moved to the OOB Received (2) state. However, the receiver **MAY** buffer redundant OOB messages in case OOB message expiry or similar error detected in the Completion Exchange causes it to return to the Waiting for OOB (1) state. It is **RECOMMENDED** that the OOB receiver notifies the user about redundant OOB messages, but it **MAY** also discard them silently.

The sender will typically generate a new Noob, and therefore a new OOB message, at constant time intervals (NoobInterval). The RECOMMENDED interval is $\text{NoobInterval} = \text{NoobTimeout} / 2$, so that the two latest values are always accepted. However, the timing of the Noob generation may also be based on user interaction or on implementation considerations.

Even though not recommended (see Section 3.3), this specification allows both directions to be negotiated (Dirp=3) for the OOB channel. In that case, both sides SHOULD output the OOB message, and it is up to the user to deliver one of them.

The details of the OOB channel implementation including the message encoding are defined by the application. Appendix E gives an example of how the OOB message can be encoded as a URL that may be embedded in a QR code and NFC tag.

3.2.4. Completion Exchange

After the Initial Exchange, if both the server and the peer support the peer-to-server direction for the OOB channel, the peer SHOULD initiate the EAP-NOOB method again after an applications-specific waiting time in order to probe for completion of the OOB Step. Also, if both sides support the server-to-peer direction of the OOB exchange and the peer receives the OOB message, it SHOULD initiate the EAP-NOOB method immediately. Depending on the combination of the peer and server states, the server continues with with the Completion Exchange or Waiting Exchange (see Section 3.2.1 on how the server makes this decision).

The Completion Exchange comprises the common handshake and one or two further EAP-NOOB request-response pairs. If the peer is in the Waiting for OOB (1) state, the OOB message has been sent in the peer-to-server direction. In that case, only one request-response pair (Type=4) takes place. In the request, the server sends the NoobId value, which the peer uses to identify the exact OOB message received by the server. On the other hand, if the peer is in the OOB Received (2) state, the direction of the OOB message is from server to peer. In that case, two request-response pairs (Type=8 and Type=4) are needed. The purpose of the first request-response pair (Type=8) is that it enables the server to discover NoobId, which identifies the exact OOB message received by the peer. The server returns the same NoobId to the peer in the latter request.

In the last and sometimes only request-response pair (Type=4) of the Completion Exchange, the server and peer exchange message authentication codes. Both sides MUST compute the keys Kms and Kmp as defined in Section 3.5 and the message authentication codes MACs

and MACp as defined in Section 3.3.2. Both sides MUST compare the received message authentication code with a locally computed value. If the peer finds that it has received the correct value of MACs and the server finds that it has received the correct value of MACp, the Completion Exchange ends in EAP-Success. Otherwise, the endpoint where the comparison fails indicates this with an error message (error code 4001, see Section 3.6.1) and the Completion Exchange ends in EAP-Failure.

After successful Completion Exchange, both the server and the peer move to the Registered (4) state. They also derive the output keying material and store the persistent EAP-NOOB association state as defined in Section 3.4 and Section 3.5.

It is possible that the OOB message expires before it is received. In that case, the sender of the OOB message no longer recognizes the NoobId that it receives in the Completion Exchange. Another reason why the OOB sender might not recognize the NoobId is if the received OOB message was spoofed and contained an attacker-generated Noob value. The recipient of an unrecognized NoobId indicates this with an error message (error code 2003, see Section 3.6.1) and the Completion Exchange ends in EAP-Failure. The recipient of the error message 2003 moves back to the Waiting for OOB (1) state. This state transition is shown as OOB Reject in Figure 1 (even though it really is a specific type of failed Completion Exchange). The sender of the error message, on the other hand, stays in its previous state.

Although it is not expected to occur in practice, poor user interface design could lead to two OOB messages delivered simultaneously, one from the peer to the server and the other from the server to the peer. The server detects this event in the beginning of the Completion Exchange by observing that both the server and peer are in the OOB Received state (2). In that case, as a tiebreaker, the server MUST behave as if only the server-to-peer message had been delivered.

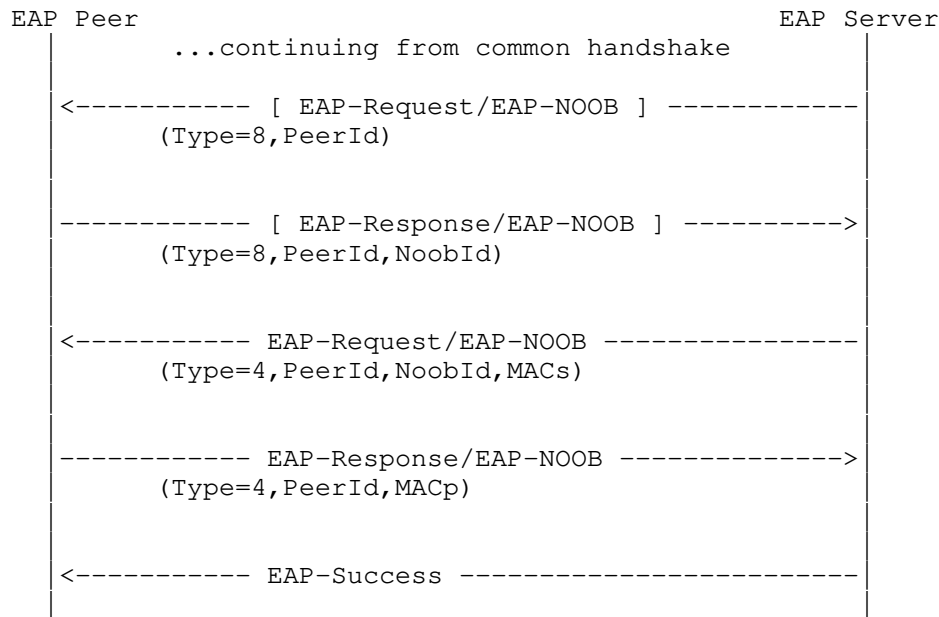


Figure 6: Completion Exchange

3.2.5. Waiting Exchange

As explained in Section 3.2.4, the peer SHOULD probe the server for completion of the OOB Step. When the combination of the peer and server states indicates that the OOB message has not yet been delivered, the server chooses the Waiting Exchange (see Section 3.2.1 on how the server makes this decision). The Waiting Exchange comprises the common handshake and one further request-response pair, and it ends always in EAP-Failure.

In order to limit the rate at which peers probe the server, the server MAY send to the peer either in the Initial Exchange or in the Waiting Exchange a minimum time to wait before probing the server again. A peer that has not received an OOB message MUST wait at least the server-specified minimum waiting time in seconds (SleepTime) before initiating EAP again with the same server. The peer uses the latest SleepTime value that it has received in or after the Initial Exchange. If the server has not sent any SleepTime value, the peer SHOULD wait for an application-specified minimum time (SleepTimeDefault).

After the Waiting Exchange, the peer MUST discard (from its local ephemeral storage) Noob values that it has sent to the server in OOB

messages that are older than the application-defined timeout NoobTimeout (see Section 3.2.3). The peer SHOULD discard such expired Noob values even if the probing failed, e.g. because of failure to connect to the EAP server or incorrect HMAC. The timeout of peer-generated Noob values is defined like this in order to allow the peer to probe the server once after it has waited for the server-specified SleepTime.

If the server and peer have negotiated to use only the server-to-peer direction for the OOB channel (Dirp=2), the peer SHOULD nevertheless probe the server. The purpose of this is to keep the server informed about the peers that are still waiting for OOB messages. The server MAY set SleepTime to a high number (3600) to prevent the peer from probing the server frequently.

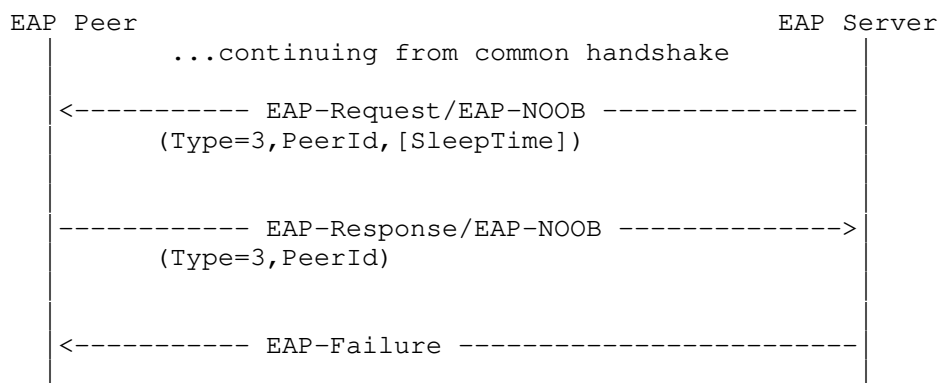


Figure 7: Waiting Exchange

3.3. Protocol data fields

This section defines the various identifiers and data fields used in the EAP-NOOB protocol.

3.3.1. Peer identifier, realm and NAI

The server allocates a new peer identifier (PeerId) for the peer in the Initial Exchange. The peer identifier MUST follow the syntax of the utf8-username specified in [RFC7542]. The server MUST generate the identifiers in such a way that they do not repeat and cannot be guessed by the peer or third parties before the server sends them to the peer in the Initial Exchange. One way to generate the identifiers is to choose a random 16-byte identifier and to base64url encode it without padding [RFC4648] into a 22-character string.

Another way to generate the identifiers is to choose a random 22-character alphanumeric string. It is RECOMMENDED to not use identifiers longer than this because they result in longer OOB messages.

The peer uses the allocated PeerId to identify itself to the server in the subsequent exchanges. It sets the PeerId value in response type 9 as follows. When the peer is in the Unregistered (0) state, it SHOULD omit the PeerId from response type 9. When the peer is in one of the states 1..2, it MUST use the PeerId that the server assigned to it in the latest Initial Exchange. When the peer is in one of the persistent states 3..4, it MUST use the PeerId from its persistent EAP-NOOB association. (The PeerId is written to the association when the peer moves to the Registered (4) state after a Completion Exchange.)

The default realm for the peer is "eap-noob.net". However, the user or application MAY provide a different default realm to the peer. Furthermore, the server MAY assign a new realm to the peer in the Initial Exchange or Reconnect Exchange, in the Realm field of response types 1 and 5. The Realm value MUST follow the syntax of the utf8-realm specified in [RFC7542]. When the peer is in the Unregistered (0) state, or when the peer is in one of the states 1..2 and the server did not send a Realm in the latest Initial Exchange, the peer MUST use the default realm. When the peer is in one of the states 1..2 and the server sent a Realm in the latest Initial Exchange, the peer MUST use that realm. Finally, when the peer is in one of the persistent states 3..4, it MUST use the Realm from its persistent EAP-NOOB association. (The Realm is written to the association when the peer moves to the Registered (4) state after a Completion Exchange or Reconnect Exchange.)

To compose its NAI [RFC7542], the peer concatenates the string "noob@" and the server-assigned realm. When no server-assigned realm is available, the default value is used instead.

The purpose of the server-assigned realm is to enable more flexible routing of the EAP sessions over the AAA infrastructure, including roaming scenarios (see Appendix D). Moreover, some Authenticators or AAA servers use the assigned Realm to determine peer-specific connection parameters, such as isolating the peer to a specific VLAN. The possibility to configure a different default realm enables registration of new devices while roaming. It also enables manufacturers to set up their own AAA servers for bootstrapping of new peer devices.

The peer's PeerId and Realm are ephemeral until a successful Completion Exchange takes place. Thereafter, the values become parts

of the persistent EAP-NOOB association, until the user resets the peer and the server or until a new Realm is assigned in the Reconnect Exchange.

3.3.2. Message data fields

Table 1 defines the data fields in the protocol messages. The in-band messages are formatted as JSON objects [RFC8259] in UTF-8 encoding. The JSON member names are in the left-hand column of the table.

Data field	Description
Vers, Verp	EAP-NOOB protocol versions supported by the EAP server, and the protocol version chosen by the peer. Vers is a JSON array of unsigned integers, and Verp is an unsigned integer. Example values are "[1]" and "1", respectively.
PeerId	Peer identifier as defined in Section 3.3.1.
Realm	Peer realm as defined in Section 3.3.1.
PeerState	Peer state is an integer in the range 0..4 (see Figure 1). However, only values 0..3 are ever sent in the protocol messages.
Type	EAP-NOOB message type. The type is an integer in the range 0..9. EAP-NOOB requests and the corresponding responses share the same type value.
PKs, PKp	The public components of the ECDHE keys of the server and peer. PKs and PKp are sent in the JSON Web Key (JWK) format [RFC7517]. Detailed format of the JWK object is defined by the cryptosuite.
Cryptosuites, Cryptosuitep	The identifiers of cryptosuites supported by the server and of the cryptosuite selected by the peer. The server-supported cryptosuites in Cryptosuites are formatted as a JSON array of the identifier integers. The server MUST send a nonempty array with no repeating elements, ordered by decreasing priority. The peer MUST respond with exactly one suite in the

	Cryptosuitep value, formatted as an identifier integer. The registration of cryptosuites is specified in Section 4.1. Example values are "[1]" and "1", respectively.
Dirs, Dirp	The OOB channel directions supported by the server and the directions selected by the peer. The possible values are 1=peer-to-server, 2=server-to-peer, 3=both directions.
Dir	The actual direction of the OOB message (1=peer-to-server, 2=server-to-peer). This value is not sent over any communication channel but it is included in the computation of the cryptographic fingerprint Hoob.
Ns, Np	32-byte nonces for the Initial Exchange.
ServerInfo	This field contains information about the server to be passed from the EAP method to the application layer in the peer. The information is specific to the application or to the OOB channel and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the access-network name and server name or a Uniform Resource Locator (URL) [RFC4266] or some other information that helps the user to deliver the OOB message to the server through the out-of-band channel.
PeerInfo	This field contains information about the peer to be passed from the EAP method to the application layer in the server. The information is specific to the application or to the OOB channel and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the peer brand, model and serial number, which help the user to distinguish between devices and to deliver the OOB message to the correct peer through the out-of-band channel.
SleepTime	The number of seconds for which peer MUST NOT start a new execution of the EAP-NOOB method with the authenticator, unless the peer receives the OOB message or the peer is reset by the user. The server can use this field to limit the rate at which peers probe it.

	SleepTime is an unsigned integer in the range 0..3600.
Noob	16-byte secret nonce sent through the OOB channel and used for the session key derivation. The endpoint that received the OOB message uses this secret in the Completion Exchange to authenticate the exchanged key to the endpoint that sent the OOB message.
Hoob	16-byte cryptographic fingerprint (i.e. hash value) computed from all the parameters exchanged in the Initial Exchange and in the OOB message. Receiving this fingerprint over the OOB channel guarantees the integrity of the key exchange and parameter negotiation. Hence, it authenticates the exchanged key to the endpoint that receives the OOB message.
NoobId	16-byte identifier for the OOB message, computed with a one-way function from the nonce Noob in the message.
MACs, MACp	Message authentication codes (HMAC) for mutual authentication, key confirmation, and integrity check on the exchanged information. The input to the HMAC is defined below, and the key for the HMAC is defined in Section 3.5.
Ns2, Np2	32-byte Nonces for the Reconnect Exchange.
KeyingMode	Integer indicating the key derivation method. 0 in the Completion Exchange, and 1..3 in the Reconnect Exchange.
PKs2, PKp2	The public components of the ECDHE keys of the server and peer for the Reconnect Exchange. PKp2 and PKs2 are sent in the JSON Web Key (JWK) format [RFC7517]. Detailed format of the JWK object is defined by the cryptosuite.
MACs2, MACp2	Message authentication codes (HMAC) for mutual authentication, key confirmation, and integrity check on the Reconnect Exchange. The input to the HMAC is defined below, and the key for the HMAC is defined in Section 3.5.

ErrorCode	Integer indicating an error condition. Defined in Section 4.3.
ErrorInfo	Textual error message for logging and debugging purposes. UTF-8 string of at most 500 bytes.

Table 1: Message data fields

It is RECOMMENDED for servers to support both OOB channel directions (Dirs=3), unless the type of the OOB channel limits them to one direction (Dirs=1 or Dirs=2). On the other hand, it is RECOMMENDED that the peer selects only one direction (Dirp=1 or Dirp=2) even when both directions (Dirp=3) would be technically possible. The reason is that, if value 3 is negotiated, the user may be presented with two OOB messages, one for each direction, even though only one of them needs to be delivered. This can be confusing to the user. Nevertheless, the EAP-NOOB protocol is designed to cope also with selected value 3, in which case it uses the first delivered OOB message. In the unlikely case of simultaneously delivered OOB messages, the protocol prioritizes the server-to-peer direction.

The nonces in the in-band messages (Ns, Np, Ns2, Np2) are 32-byte fresh random byte strings, and the secret nonce Noob is a 16-byte fresh random byte string. All the nonces are generated by the endpoint that sends the message.

The fingerprint Hoob and the identifier NoobId are computed with the cryptographic hash function specified in the negotiated cryptosuite and truncated to the 16 leftmost bytes of the output. The message authentication codes (MACs, MACp, MACs2, MACp2) are computed with the HMAC function [RFC2104] based on the same cryptographic hash function and truncated to the 32 leftmost bytes of the output.

The inputs to the hash function for computing the fingerprint Hoob and to the HMAC for computing MACs, MACp, MACs2 and MACp2 are JSON arrays containing a fixed number (17) of elements. The array elements MUST be copied to the array verbatim from the sent and received in-band messages. When the element is a JSON object, its members MUST NOT be reordered or re-encoded. Whitespace MUST NOT be added anywhere in the JSON structure. Implementers should check that their JSON library copies the elements as UTF-8 strings and does not modify them in any way, and that it does not add whitespace to the HMAC input.

The inputs for computing the fingerprint and message authentication codes are the following:

```
Hoob = H(Dir, Vers, Verp, PeerId, Cryptosuites, Dirs, ServerInfo, Cryptosuitep, Dirp, [Realm], PeerInfo, 0, PKs, Ns, PKp, Np, Noob) .
```

```
NoobId = H("NoobId", Noob) .
```

```
MACs = HMAC(Kms; 2, Vers, Verp, PeerId, Cryptosuites, Dirs, ServerInfo, Cryptosuitep, Dirp, [Realm], PeerInfo, 0, PKs, Ns, PKp, Np, Noob) .
```

```
MACp = HMAC(Kmp; 1, Vers, Verp, PeerId, Cryptosuites, Dirs, ServerInfo, Cryptosuitep, Dirp, [Realm], PeerInfo, 0, PKs, Ns, PKp, Np, Noob) .
```

```
MACs2 = HMAC(Kms2; 2, Vers, Verp, PeerId, Cryptosuites, "", [ServerInfo], Cryptosuitep, "", [Realm], [PeerInfo], KeyingMode, [PKs2], Ns2, [PKp2], Np2, "")
```

```
MACp2 = HMAC(Kmp2; 1, Vers, Verp, PeerId, Cryptosuites, "", [ServerInfo], Cryptosuitep, "", [Realm], [PeerInfo], KeyingMode, [PKs2], Ns2, [PKp2], Np2, "")
```

Missing input values are represented by empty strings "" in the array. The values indicated with "" above are always empty strings. Realm is included in the computation of MACs and MACp if it was sent or received in the preceding Initial Exchange. Each of the values in brackets for the computation of Macs2 and Macp2 MUST be included if it was sent or received in the same Reconnect Exchange; otherwise the value is replaced by an empty string "".

The parameter Dir indicates the direction in which the OOB message containing the Noob value is being sent (1=peer-to-server, 2=server-to-peer). This field is included in the Hoob input to prevent the user from accidentally delivering the OOB message back to its originator in the rare cases where both OOB directions have been negotiated. The keys (Kms, Kmp, Kms2, Kmp2) for the HMACs are defined in Section 3.5.

The nonces (Ns, Np, Ns2, Np2, Noob) and the hash value (NoobId) MUST be base64url encoded [RFC4648] when they are used as input to the cryptographic functions H or HMAC. These values and the message authentication codes (MACs, MACp, MACs2, MACp2) MUST also be base64url encoded when they are sent in the in-band messages. The values Noob and Hoob in the OOB channel MAY be base64url encoded if that is appropriate for the application and the OOB channel. All base64url encoding is done without padding. The base64url encoded values will naturally consume more space than the number of bytes specified above (22-character string for a 16-byte nonce and

43-character string for a 32-byte nonce or message authentication code). In the key derivation in Section 3.5, on the other hand, the unencoded nonces (raw bytes) are used as input to the key derivation function.

The ServerInfo and PeerInfo are JSON objects with UTF-8 encoding. The length of either encoded object as a byte array MUST NOT exceed 500 bytes. The format and semantics of these objects MUST be defined by the application that uses the EAP-NOOB method.

3.4. Fast reconnect and rekeying

EAP-NOOB implements Fast Reconnect ([RFC3748], section 7.2.1) that avoids repeated use of the user-assisted OOB channel.

The rekeying and the Reconnect Exchange may be needed for several reasons. New EAP output values MSK and EMSK may be needed because of mobility or timeout of session keys. Software or hardware failure or user action may also cause the authenticator, EAP server or peer to lose its non-persistent state data. The failure would typically be detected by the peer or authenticator when session keys no longer are accepted by the other endpoint. Change in the supported cryptosuites in the EAP server or peer may also cause the need for a new key exchange. When the EAP server or peer detects any one of these events, it MUST change from the Registered to Reconnecting state. These state transitions are labeled Mobility/Timeout/Failure in Figure 1. The EAP-NOOB method will then perform the Reconnect Exchange next time when EAP is triggered.

3.4.1. Persistent EAP-NOOB association

To enable rekeying, the EAP server and peer store the session state in persistent memory after a successful Completion Exchange. This state data, called "persistent EAP-NOOB association", MUST include at least the data fields shown in Table 2. They are used for identifying and authenticating the peer in the Reconnect Exchange. When a persistent EAP-NOOB association exists, the EAP server and peer are in the Registered state (4) or Reconnecting state (3), as shown in Figure 1.

Data field	Value	Type
PeerId	Peer identifier allocated by server	UTF-8 string (typically 22 bytes)
Verp	Negotiated protocol version	integer
Cryptosuitep	Negotiated cryptosuite	integer
CryptosuitepPrev (at peer only)	Previous cryptosuite	integer
Realm	Optional realm assigned by server (default value is "eap-noob.net")	UTF-8 string
Kz	Persistent key material	32 bytes
KzPrev (at peer only)	Previous Kz value	32 bytes

Table 2: Persistent EAP-NOOB association

3.4.2. Reconnect Exchange

The server chooses the Reconnect Exchange when both the peer and the server are in a persistent state and fast reconnection is needed (see Section 3.2.1 for details).

The Reconnect Exchange comprises the common handshake and three further EAP-NOOB request-response pairs, one for cryptosuite and parameter negotiation, another for the nonce and ECDHE key exchange, and the last one for exchanging message authentication codes. In the first request and response (Type=5) the server and peer negotiate a protocol version and cryptosuite in the same way as in the Initial Exchange. The server SHOULD NOT offer and the peer MUST NOT accept protocol versions or cryptosuites that it knows to be weaker than the one currently in the Cryptosuitep field of the persistent EAP-NOOB association. The server SHOULD NOT needlessly change the cryptosuites it offers to the same peer because peer devices may have limited ability to update their persistent storage. However, if the peer has different values in the Cryptosuitep and CryptosuitepPrev fields, it SHOULD also accept offers that are not weaker than CryptosuitepPrev. Note that Cryptosuitep and CryptosuitepPrev from the persistent EAP-NOOB association are only used to support the

negotiation as described above; all actual cryptographic operations use the negotiated cryptosuite. The request and response (Type=5) MAY additionally contain PeerInfo and ServerInfo objects.

The server then determines the KeyingMode (defined in Section 3.5) based on changes in the negotiated cryptosuite and whether it desires to achieve forward secrecy or not. The server SHOULD only select KeyingMode 3 when the negotiated cryptosuite differs from the Cryptosuitep in the server's persistent EAP-NOOB association, although it is technically possible to select this values without changing the cryptosuite. In the second request and response (Type=6), the server informs the peer about the KeyingMode, and the server and peer exchange nonces (Ns2, Np2). When KeyingMode is 2 or 3 (rekeying with ECDHE), they also exchange public components of ECDHE keys (PKs2, PKp2). The server ECDHE key MUST be fresh, i.e. not previously used with the same peer, and the peer ECDHE key SHOULD be fresh, i.e. not previously used.

In the third and final request and response (Type=7), the server and peer exchange message authentication codes. Both sides MUST compute the keys Kms2 and Kmp2 as defined in Section 3.5 and the message authentication codes MACs2 and MACp2 as defined in Section 3.3.2. Both sides MUST compare the received message authentication code with a locally computed value.

The rules by which the peer compares the received MACs2 are non-trivial because, in addition to authenticating the current exchange, MACs2 may confirm the success or failure of a recent cryptosuite upgrade. The peer processes the final request (Type=7) as follows:

1. The peer first compares the received MACs2 value with one it computed using the Kz stored in the persistent EAP-NOOB association. If the received and computed values match, the peer deletes any data stored in the CryptosuitepPrev and KzPrev fields of the persistent EAP-NOOB association. It does this because the received MACs2 confirms that the peer and server share the same Cryptosuitep and Kz, and any previous values must no longer be accepted.
2. If, on the other hand, the peer finds that the received MACs2 value does not match the one it computed locally with Kz, the peer checks whether the KzPrev field in the persistent EAP-NOOB association stores a key. If it does, the peer repeats the key derivation (Section 3.5) and local MACs2 computation (Section 3.3.2) using KzPrev in place of Kz. If this second computed MACs2 matches the received value, the match indicates synchronization failure caused by the loss of the last response (Type=7) in a previously attempted cryptosuite upgrade. In this

case, the peer rolls back that upgrade by overwriting Cryptosuitep with CryptosuitepPrev and Kz with KzPrev in the persistent EAP-NOOB association. It also clears the CryptosuitepPrev and KzPrev fields.

3. If the received MACs2 matched one of the locally computed values, the peer proceeds to send the final response (Type=7). The peer also moves to the Registered (4) state. When KeyingMode is 1 or 2, the peer stops here. When KeyingMode is 3, the peer also updates the persistent EAP-NOOB association with the negotiated Cryptosuitep and the newly-derived Kz value. To prepare for possible synchronization failure caused by the loss of the final response (Type=7) during cryptosuite upgrade, the peer copies the old Cryptosuitep and Kz values in the persistent EAP-NOOB association to the CryptosuitepPrev and KzPrev fields.
4. Finally, if the peer finds that the received MACs2 does not match either of the two values that it computed locally (or one value if no KzPrev was stored), the peer sends an error message (error code 4001, see Section 3.6.1), which causes the the Reconnect Exchange to end in EAP-Failure.

The server rules for processing the final message are simpler than the peer rules because the server does not store previous keys and it never rolls back a cryptosuite upgrade. Upon receiving the final response (Type=7), the server compares the received value of MACp2 with one it computes locally. If the values match, the Reconnect Exchange ends in EAP-Success. When KeyingMode is 3, the server also updates Cryptosuitep and Kz in the persistent EAP-NOOB association. On the other hand, if the server finds that the values do not match, it sends an error message (error code 4001), and the Reconnect Exchange ends in EAP-Failure.

The endpoints MAY send updated Realm, ServerInfo and PeerInfo objects in the Reconnect Exchange. When there is no update to the values, they SHOULD omit this information from the messages. If the Realm was sent, each side updates Realm in the persistent EAP-NOOB association when moving to the Registered (4) state.

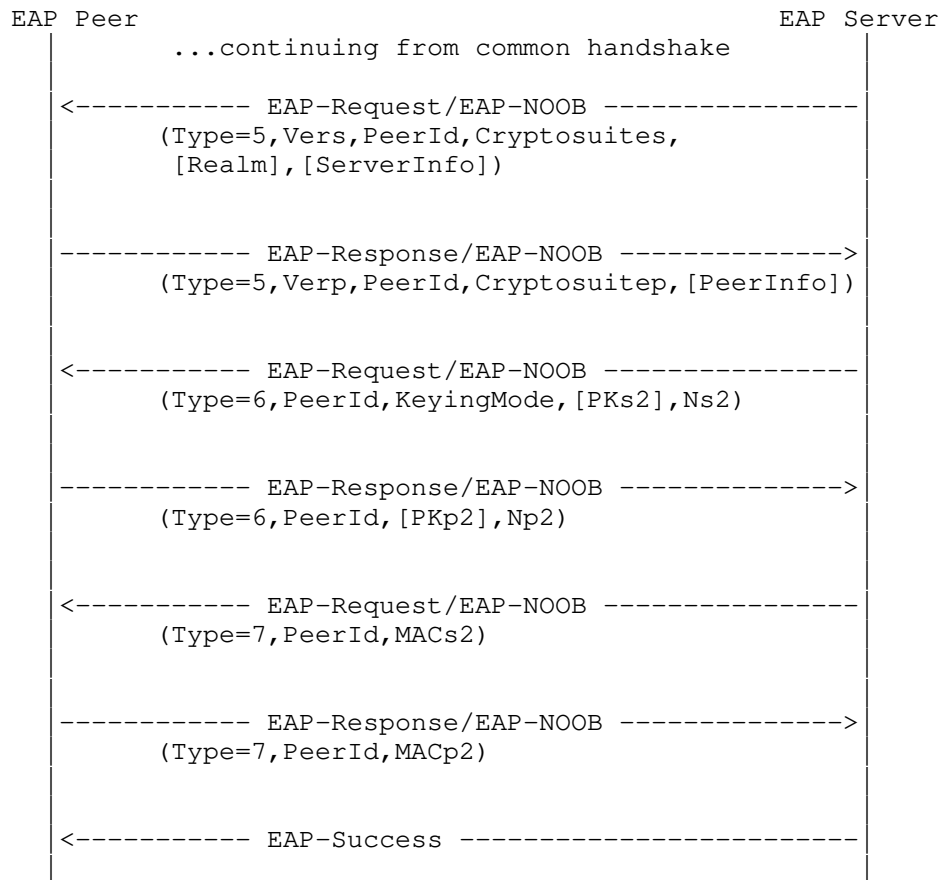


Figure 8: Reconnect Exchange

3.4.3. User reset

As shown in the association state machine in Figure 1, the only specified way for the association to return from the Registered state (4) to the Unregistered state (0) is through user-initiated reset. After the reset, a new OOB message will be needed to establish a new association between the EAP server and peer. Typical situations in which the user reset is required are when the other side has accidentally lost the persistent EAP-NOOB association data, or when the peer device is decommissioned.

The server could detect that the peer is in the Registered or Reconnecting state but the server itself is in one of the ephemeral states 0..2 (including situations where the server does not recognize

the PeerId). In this case, effort should be made to recover the persistent server state, for example, from a backup storage - especially if many peer devices are similarly affected. If that is not possible, the EAP server SHOULD log the error or notify an administrator. The only way to continue from such a situation is by having the user reset the peer device.

On the other hand, if the peer is in any of the ephemeral states 0..2, including the Unregistered state, the server will treat the peer as a new peer device and allocate a new PeerId to it. The PeerInfo can be used by the user as a clue to which physical device has lost its state. However, there is no secure way of matching the "new" peer with the old PeerId without repeating the OOB Step. This situation will be resolved when the user performs the OOB Step and, thus, identifies the physical peer device. The server user interface MAY support situations where the "new" peer is actually a previously registered peer that has been reset by a user or otherwise lost its persistent data. In those cases, the user could choose to merge new peer identity with the old one in the server. The alternative is to treat the device just like a new peer.

3.5. Key derivation

EAP-NOOB derives the EAP output values MSK and EMSK and other secret keying material from the output of an Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) algorithm following the NIST specification [NIST-DH]. In NIST terminology, we use a $C(2, 0, \text{ECC CDH})$ scheme, i.e. two ephemeral keys and no static keys. In the Initial and Reconnect Exchanges, the server and peer compute the ECDHE shared secret Z as defined in section 6.1.2.2 of the NIST specification [NIST-DH]. In the Completion and Reconnect Exchanges, the server and peer compute the secret keying material from Z with the single-step key derivation function (KDF) defined in section 5.8.1 of the NIST specification. The hash function H for KDF is taken from the negotiated cryptosuite.

KeyingMode	Description
0	Completion Exchange (always with ECDHE)
1	Reconnect Exchange, rekeying without ECDHE
2	Reconnect Exchange, rekeying with ECHDE, no change in cryptosuite
3	Reconnect Exchange, rekeying with ECDHE, new cryptosuite negotiated

Table 3: Keying modes

The key derivation has three different modes (KeyingMode), which are specified in Table 3. Table 4 defines the inputs to KDF in each KeyingMode.

In the Completion Exchange (KeyingMode=0), the input Z comes from the preceding Initial exchange. KDF takes some additional inputs (OtherInfo), for which we use the concatenation format defined in section 5.8.1.2.1 of the NIST specification [NIST-DH]. OtherInfo consists of the AlgorithmId, PartyUInfo, PartyVInfo, and SuppPrivInfo fields. The first three fields are fixed-length bit strings, and SuppPrivInfo is a variable-length string with a one-byte Datalength counter. AlgorithmId is the fixed-length 8-byte ASCII string "EAP-NOOB". The other input values are the server and peer nonces. In the Completion Exchange, the inputs also include the secret nonce Noob from the OOB message.

In the simplest form of the Reconnect Exchange (KeyingMode=1), fresh nonces are exchanged but no ECDHE keys are sent. In this case, input Z to the KDF is replaced with the shared key Kz from the persistent EAP-NOOB association. The result is rekeying without the computational cost of the ECDHE exchange, but also without forward secrecy.

When forward secrecy is desired in the Reconnect Exchange (KeyingMode=2 or KeyingMode=3), both nonces and ECDHE keys are exchanged. Input Z is the fresh shared secret from the ECDHE exchange with PKs2 and PKp2. The inputs also include the shared secret Kz from the persistent EAP-NOOB association. This binds the rekeying output to the previously authenticated keys.

KeyingMode	KDF input field	Value	Length (bytes)
0 Completion	Z	ECDHE shared secret from PKs and PKp	variable
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np	32
	PartyVInfo	Ns	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	Noob	16
1 Reconnect, rekeying without ECDHE	Z	Kz	32
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np2	32
	PartyVInfo	Ns2	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	(null)	0
2 or 3 Reconnect, rekeying, with ECDHE, same or new cryptosuite	Z	ECDHE shared secret from PKs2 and PKp2	variable
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np2	32
	PartyVInfo	Ns2	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	Kz	32

Table 4: Key derivation input

Table 5 defines how the output bytes of KDF are used. In addition to the EAP output values MSK and EMSK, the server and peer derive another shared secret key AMSK, which MAY be used for application-layer security. Further output bytes are used internally by EAP-NOOB for the message authentication keys (Kms, Kmp, Kms2, Kmp2).

The Completion Exchange (KeyingMode=0) produces the shared secret Kz, which the server and peer store in the persistent EAP-NOOB association. When a new cryptosuite is negotiated in the Reconnect Exchange (KeyingMode=3), it similarly produces a new Kz. In that case, the server and peer update both the cryptosuite and Kz in the persistent EAP-NOOB association. Additionally, the peer stores the previous Cryptosuitep and Kz values in the CryptosuitepPrev and KzPrev fields of the persistent EAP-NOOB association.

KeyingMode	KDF output bytes	Used as	Length (bytes)
0 Completion	0..63	MSK	64
	64..127	EMSK	64
	128..191	AMSK	64
	192..223	MethodId	32
	224..255	Kms	32
	256..287	Kmp	32
	288..319	Kz	32
1 or 2 Reconnect, rekeying without ECDHE, or with ECDHE and unchanged cryptosuite	0..63	MSK	64
	64..127	EMSK	64
	128..191	AMSK	64
	192..223	MethodId	32
	224..255	Kms2	32
	256..287	Kmp2	32
3 Reconnect, rekeying with ECDHE, new cryptosuite	0..63	MSK	64
	64..127	EMSK	64
	128..191	AMSK	64
	192..223	MethodId	32
	224..255	Kms2	32
	256..287	Kmp2	32
	288..319	Kz	32

Table 5: Key derivation output

Finally, every EAP method must export a Server-Id, Peer-Id and Session-Id [RFC5247]. In EAP-NOOB, the exported Peer-Id is the PeerId which the server has assigned to the peer. The exported Server-Id is a zero-length string (i.e. null string) because EAP-NOOB neither knows nor assigns any server identifier. The exported Session-Id is created by concatenating the Type-Code xxx (TBA) with the MethodId, which is obtained from the KDF output as shown in Table 5.

3.6. Error handling

Various error conditions in EAP-NOOB are handled by sending an error notification message (Type=0) instead of the expected next EAP request or response message. Both the EAP server and the peer may send the error notification, as shown in Figure 9 and Figure 10. After sending or receiving an error notification, the server MUST send an EAP-Failure (as required by [RFC3748] section 4.2). The

notification MAY contain an `ErrorInfo` field, which is a UTF-8 encoded text string with a maximum length of 500 bytes. It is used for sending descriptive information about the error for logging and debugging purposes.

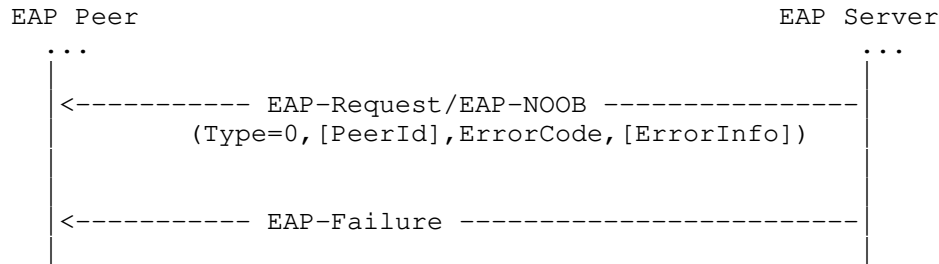


Figure 9: Error notification from server to peer

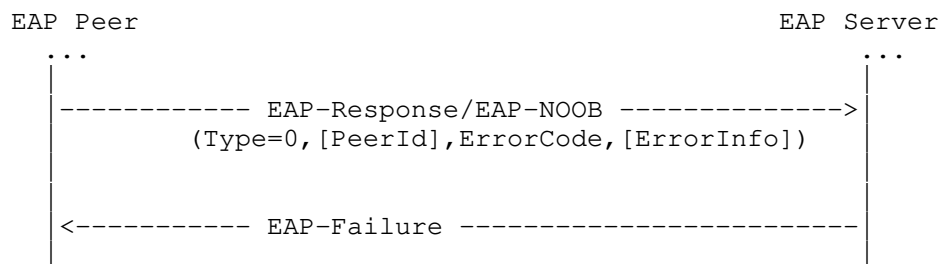


Figure 10: Error notification from peer to server

After the exchange fails due to an error notification, the server and peer set the association state as follows. In the Initial Exchange, both the sender and recipient of the error notification MUST set the association state to the Unregistered (0) state. In the Waiting and Completion Exchanges, each side MUST remain in its old state as if the failed exchange had not taken place, with the exception that the recipient of error code 2003 processes it as specified in Section 3.2.4. In the Reconnect Exchange, both sides MUST set the association state to the Reconnecting (3) state.

Errors that occur in the OOB channel are not explicitly notified in-band.

3.6.1. Invalid messages

If the NAI structure is invalid, the server SHOULD send the error code 1001 to the peer. The recipient of an EAP-NOOB request or response SHOULD send the following error codes back to the sender: 1002 if it cannot parse the message as a JSON object or the top-level JSON object has missing or unrecognized members; 1003 if a data field has an invalid value, such as an integer out of range, and there is no more specific error code available; 1004 if the received message type was unexpected in the current state; 2004 if the PeerId has an unexpected value; 2003 if the NoobId is not recognized; and 1007 if the ECDHE key is invalid.

3.6.2. Unwanted peer

The preferred way for the EAP server to rate limit EAP-NOOB connections from a peer is to use the SleepTime parameter in the Waiting Exchange. However, if the EAP server receives repeated EAP-NOOB connections from a peer which apparently should not connect to this server, the server MAY indicate that the connections are unwanted by sending the error code 2001. After receiving this error message, the peer MAY refrain from reconnecting to the same EAP server and, if possible, both the EAP server and peer SHOULD indicate this error condition to the user or server administrator. However, in order to avoid persistent denial of service, the peer is not required to stop entirely from reconnecting to the server.

3.6.3. State mismatch

In the states indicated by "-" in Figure 11 in Appendix A, user action is required to reset the association state or to recover it, for example, from backup storage. In those cases, the server sends the error code 2002 to the peer. If possible, both the EAP server and peer SHOULD indicate this error condition to the user or server administrator.

3.6.4. Negotiation failure

If there is no matching protocol version, the peer sends the error code 3001 to the server. If there is no matching cryptosuite, the peer sends the error code 3002 to the server. If there is no matching OOB direction, the peer sends the error code 3003 to the server.

In practice, there is no way of recovering from these errors without software or hardware changes. If possible, both the EAP server and peer SHOULD indicate these error conditions to the user.

3.6.5. Cryptographic verification failure

If the receiver of the OOB message detects an unrecognized PeerId or incorrect fingerprint (Hoob) in the OOB message, the receiver **MUST** remain in the Waiting for OOB state (1) as if no OOB message was received. The receiver **SHOULD** indicate the failure to accept the OOB message to the user. No in-band error message is sent.

Note that if the OOB message was delivered from the server to the peer and the peer does not recognize the PeerId, the likely cause is that the user has unintentionally delivered the OOB message to the wrong peer device. If possible, the peer **SHOULD** indicate this to the user; however, the peer device may not have the capability for many different error indications to the user and it **MAY** use the same indication as in the case of an incorrect fingerprint.

The rationale for the above is that the invalid OOB message could have been presented to the receiver by mistake or intentionally by a malicious party and, thus, it should be ignored in the hope that the honest user will soon deliver a correct OOB message.

If the EAP server or peer detects an incorrect message authentication code (MACs, MACp, MACs2, MACp2), it sends the error code 4001 to the other side. As specified in the beginning of Section 3.6, the failed Completion Exchange will not result in server or peer state changes while error in the Reconnect Exchange will put both sides to the Reconnecting (3) state and thus lead to another reconnect attempt.

The rationale for this is that the invalid cryptographic message may have been spoofed by a malicious party and, thus, it should be ignored. In particular, a spoofed message on the in-band channel should not force the honest user to perform the OOB Step again. In practice, however, the error may be caused by other failures, such as a software bug. For this reason, the EAP server **MAY** limit the rate of peer connections with SleepTime after the above error. Also, there **SHOULD** be a way for the user to reset the peer to the Unregistered state (0), so that the OOB Step can be repeated at the last resort.

3.6.6. Application-specific failure

Applications **MAY** define new error messages for failures that are specific to the application or to one type of OOB channel. They **MAY** also use the generic application-specific error code 5001, or the error codes 5002 and 5004, which have been reserved for indicating invalid data in the ServerInfo and PeerInfo fields, respectively. Additionally, anticipating OOB channels that make use of a URL, the error code 5003 has been reserved for indicating invalid server URL.

4. IANA Considerations

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

The EAP Method Type number for EAP-NOOB needs to be assigned.

This memo also requires IANA to create new registries as defined in the following subsections.

4.1. Cryptosuites

Cryptosuites are identified by an integer. Each cryptosuite MUST specify an ECDHE curve for the key exchange, encoding of the ECDHE public key as a JWK object, and a cryptographic hash function for the fingerprint and HMAC computation and key derivation. The hash value output by the cryptographic hash function MUST be at least 32 bytes in length. The following suites are defined by EAP-NOOB:

Cryptosuite	Algorithms
1	ECDHE curve Curve25519 [RFC7748], public-key format [RFC7518] Section 6.2.1, hash function SHA-256 [RFC6234]

Table 6: EAP-NOOB cryptosuites

An example of Cryptosuite 1 public-key encoded as a JWK object is given below (line breaks are for readability only).

```
"jwk":{"kty":"EC","crv":"Curve25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-
DQ8hbeGdNrFx-FG-IK08"}
```

Assignment of new values for new cryptosuites MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126].

4.2. Message Types

EAP-NOOB request and response pairs are identified by an integer Message Type. The following Message Types are defined by EAP-NOOB:

Message Type	Used in Exchange	Purpose
1	Initial	Version, cryptosuite and parameter negotiation
2	Initial	Exchange of ECDHE keys and nonces
3	Waiting	Indication to peer that the server has not yet received an OOB message
4	Completion	Authentication and key confirmation with HMAC
5	Reconnect	Version, cryptosuite, and parameter negotiation
6	Reconnect	Exchange of ECDHE keys and nonces
7	Reconnect	Authentication and key confirmation with HMAC
8	Completion	NoobId discovery
9	All exchanges	PeerId and PeerState discovery
0	Error	Error notification

Table 7: EAP-NOOB

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

4.3. Error codes

The error codes defined by EAP-NOOB are listed in Table 8.

Error code	Purpose
1001	Invalid NAI
1002	Invalid message structure
1003	Invalid data
1004	Unexpected message type
1007	Invalid ECDHE key
2001	Unwanted peer
2002	State mismatch, user action required
2003	Unrecognized OOB message identifier
2004	Unexpected peer identifier
3001	No mutually supported protocol version
3002	No mutually supported cryptosuite
3003	No mutually supported OOB direction
4001	HMAC verification failure
5001	Application-specific error
5002	Invalid server info
5003	Invalid server URL
5004	Invalid peer info
6001-6999	Private and experimental use

Table 8: EAP-NOOB error codes

Assignment of new error codes MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126], with the exception of the range 6001-6999, which is reserved for "Private Use" and "Experimental Use".

4.4. Domain name reservation considerations

"eap-noob.net" should be registered as a special-use domain. The considerations required by [RFC6761] for registering this special-use domain name are the following:

- o **Users:** Non-admin users are not expected to encounter this name or recognize it as special. AAA administrators may need to recognize the name.
- o **Application Software:** Application software is not expected to recognize this domain name as special.
- o **Name Resolution APIs and Libraries:** Name resolution APIs and libraries are not expected to recognize this domain name as special.

- o Caching DNS Servers: Caching servers are not expected to recognize this domain name as special.
- o Authoritative DNS Servers: Authoritative DNS servers MUST respond to queries for eap-noob.net with NXDOMAIN.
- o DNS Server Operators: Except for the authoritative DNS server, there are no special requirements for the operators.
- o DNS Registries/Registrars: There are no special requirements for DNS registrars.

5. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

5.1. Implementation with wpa_supplicant and hostapd

- o Responsible Organization: Aalto University
- o Location: <<https://github.com/tuomaura/eap-noob>>
- o Coverage: This implementation includes all of the features described in the current specification. The implementation supports two dimensional QR codes and NFC as example out-of-band (OOB) channels
- o Level of Maturity: Alpha
- o Version compatibility: Version 06 of the draft implemented
- o Licensing: BSD
- o Contact Information: Tuomas Aura, tuomas.aura@aalto.fi

5.2. Implementation on Contiki

- o Responsible Organization: University of Murcia and Aalto University
- o Location: <<https://github.com/eduingles/coap-eap-noob>>
- o Coverage: This implementation includes all of the features described in the current specification. The implementation uses a blinking LED light as the out-of-band (OOB) channel
- o Level of Maturity: Alpha
- o Version compatibility: Version 05 of the draft implemented
- o Licensing: BSD
- o Contact Information: Eduardo Ingles, eduardo.ingles@um.es

5.3. Protocol modeling

The current EAP-NOOB specification has been modeled with the mCRL2 formal specification language [mcrl2]. The model <<https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/mcrl2>> was used mainly for simulating the protocol behavior and for verifying basic safety and liveness properties as part of the specification process. For example, we verified the correctness of the tiebreaking mechanism when two OOB messages are received simultaneously, one in each direction. We also verified that a man-in-the-middle attacker cannot cause persistent failure by spoofing a finite number of messages in the Reconnect Exchange. Additionally, the protocol has been modeled with the ProVerif [proverif] tool. This model <<https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/proverif>> was used to verify security properties such as mutual authentication.

6. Security considerations

EAP-NOOB is an authentication and key derivation protocol and, thus, security considerations can be found in most sections of this specification. In the following, we explain the protocol design and highlight some other special considerations.

6.1. Authentication principle

EAP-NOOB establishes a shared secret with an authenticated ECDHE key exchange. The mutual authentication in EAP-NOOB is based on two separate features, both conveyed in the OOB message. The first

authentication feature is the secret nonce Noob. The peer and server use this secret in the Completion Exchange to mutually authenticate the session key previously created with ECDHE. The message authentication codes computed with the secret nonce Noob are alone sufficient for authenticating the key exchange. The second authentication feature is the integrity-protecting fingerprint Hoob. Its purpose is to prevent impersonation and man-in-the-middle attacks even in situations where the attacker is able to eavesdrop the OOB channel and the nonce Noob is compromised. In some human-assisted OOB channels, such as sound burst or user-transferred URL, it may be easier to detect tampering than spying of the OOB message, and such applications benefit from the second authentication feature.

The additional security provided by the cryptographic fingerprint Hoob is somewhat intricate to understand. The endpoint that receives the OOB message uses Hoob to verify the integrity of the ECDHE exchange. Thus, the OOB receiver can detect impersonation and man-in-the-middle attacks on the in-band channel. The other endpoint, however, is not equally protected because the OOB message and fingerprint are sent only in one direction. Some protection to the OOB sender is afforded by the fact that the user may notice the failure of the association at the OOB receiver and therefore reset the OOB sender. Other device-pairing protocols have solved similar situations by requiring the user to confirm to the OOB sender that the association was accepted by the OOB receiver, e.g. by pressing an "confirm" button on the sender side. Applications MAY implement EAP-NOOB in this way. Nevertheless, since EAP-NOOB was designed to work with strictly one-directional OOB communication and the fingerprint is only the second authentication feature, the EAP-NOOB specification does not mandate such explicit confirmation to the OOB sender.

To summarize, EAP-NOOB uses the combined protection of the secret nonce Noob and the cryptographic fingerprint Hoob, both conveyed in the OOB message. The secret nonce Noob alone is sufficient for mutual authentication, unless the attacker can eavesdrop it from the OOB channel. Even if an attacker is able to eavesdrop the secret nonce Noob, it nevertheless cannot perform a full man-in-the-middle attack on the in-band channel because the mismatching fingerprint would alert the OOB receiver, which would reject the OOB message. The attacker that eavesdropped the secret nonce can impersonate the OOB receiver to the OOB sender. In this case, the association will appear to be complete only on the OOB sender side, and such situations have to be resolved by the user by resetting the OOB sender to the initial state.

The expected use cases for EAP-NOOB are ones where it replaces a user-entered access credentials in IoT appliances. In wireless network access without EAP, the user-entered credential is often a

passphrase that is shared by all the network stations. The advantage of an EAP-based solution, including EAP-NOOB, is that it establishes a different master secret for each peer device, which makes the system more resilient against device compromise than if there were a common master secret. Additionally, it is possible to revoke the security association for an individual device on the server side.

Forward secrecy in EAP-NOOB is optional. The Reconnect Exchange in EAP-NOOB provides forward secrecy only if both the server and peer send their fresh ECDHE keys. This allows both the server and the peer to limit the frequency of the costly computation that is required for forward secrecy. The server MAY adjust the frequency of its attempts at ECDHE rekeying based on what it knows about the peer's computational capabilities.

The users delivering the OOB messages will often authenticate themselves to the EAP server, e.g. by logging into a secure web page. In this case, the server can reliably associate the peer device with the user account. Applications that make use of EAP-NOOB can use this information for configuring the initial owner of the freshly-registered device.

6.2. Identifying correct endpoints

Potential weaknesses in EAP-NOOB arise from the fact that the user must identify physically the correct peer device. If the attacker is able to trick the user into delivering the OOB message to or from the wrong peer device, the server may create an association with the wrong peer. This reliance on user in identifying the correct endpoints is an inherent property of user-assisted out-of-band authentication.

It is, however, not possible to exploit accidental delivery of the OOB message to the wrong device when the user makes a mistake. This is because the wrong peer device would not have prepared for the attack by performing the Initial Exchange with the server. In comparison, simpler solutions where the master key is transferred to the device via the OOB channel are vulnerable to opportunistic attacks if the user mistakenly delivers the master key to more than one device.

One mechanism that can mitigate user mistakes is certification of peer devices. The certificate can convey to the server authentic identifiers and attributes of the peer device. Compared to a fully certificate-based authentication, however, EAP-NOOB can be used without trusted third parties and does not require the user to know any identifier of the peer device; physical access to the device is sufficient.

Similarly, the attacker can try to trick the user to deliver the OOB message to the wrong server, so that the peer device becomes associated with the wrong server. Since the EAP server is typically online and accessed through a web user interface, the attack would be akin to phishing attacks where the user is tricked to accessing the wrong URL and wrong web page.

6.3. Trusted path issues and misbinding attacks

Another potential threat is spoofed user input or output on the peer device. When the user is delivering the OOB message to or from the correct peer device, a trusted path between the user and the peer device is needed. That is, the user must communicate directly with an authentic operating system and EAP-NOOB implementation in the peer device and not with a spoofed user interface. Otherwise, a Registered device that is under the control of the attacker could emulate the behavior of an unregistered device. The secure path can be implemented, for example, by having the user pressing a reset button to return the device to the Unregistered state and a trusted UI. The problem with such trusted paths is that they are not standardized across devices.

Another potential consequence of spoofed UI is the misbinding attack where the user tries to register the correct but compromised device, and that device tricks the user into registering another device instead. For example, a compromised device might have a malicious full-screen app running, which presents to the user QR codes copied, in real time, from another device's screen. If the unwitting user scans the QR code and delivers the OOB message in it to the server, the wrong device may become registered in the server. Such misbinding vulnerabilities arise because the user does not have any secure way of verifying that the in-band cryptographic handshake and the out-of-band physical access are terminated at the same physical device. Sethi et al. [Sethi19] analyze the binding threat against device-pairing protocols and also EAP-NOOB. Essentially, all protocols where the authentication relies on the user's physical access to the device are vulnerable to misbinding, including EAP-NOOB.

A standardized trusted path for communicating directly with the trusted computing base in a physical device would mitigate the misbinding threat, but such paths rarely exist in practice. Careful asset tracking can also prevent most misbinding attacks because the PeerInfo sent in-band by the wrong device will not match expected values. Device certification by the manufacturer can further strengthen the asset tracking.

6.4. Peer identifiers and attributes

The PeerId value in the protocol is a server-allocated identifier for its association with the peer and SHOULD NOT be shown to the user because its value is initially ephemeral. Since the PeerId is allocated by the server and the scope of the identifier is the single server, the so-called identifier squatting attacks, where a malicious peer could reserve another peer's identifier, are not possible in EAP-NOOB. The server SHOULD assign a random or pseudo-random PeerId to each new peer. It SHOULD NOT select the PeerId based on any peer characteristics that it may know, such as the peer's link-layer network address.

User reset or failure in the OOB Step can cause the peer to perform many Initial Exchanges with the server and to allocate many PeerIds and to store the ephemeral protocol state for them. The peer will typically only remember the latest one. EAP-NOOB leaves it to the implementation to decide when to delete these ephemeral associations. There is no security reason to delete them early, and the server does not have any way to verify that the peers are actually the same one. Thus, it is safest to store the ephemeral states for at least one day. If the OOB messages are sent only in the server-to-peer direction, the server SHOULD NOT delete the ephemeral state before all the related Noob values have expired.

After completion of EAP-NOOB, the server may store the PeerInfo data, and the user may use it to identify the peer and its properties, such as the make and model or serial number. A compromised peer could lie in the PeerInfo that it sends to the server. If the server stores any information about the peer, it is important that this information is approved by the user during or after the OOB Step. Without verification by the user or authentication with vendor certificates on the application level, the PeerInfo is not authenticated information and should not be relied on.

One possible use for the PeerInfo field is EAP channel binding ([RFC3748] Section 7.15). That is, the PeerInfo may include data items that bind the EAP-NOOB association and exported keys to properties of the authenticator or the access link, such as the SSID and BSSID of the wireless network (see Appendix C).

6.5. Identity protection

The PeerInfo field contains identifiers and other information about the peer device (see Appendix C), and the peer sends this information in plaintext to the EAP server before the server authentication in EAP-NOOB has been completed. While the information refers to the peer device and not directly to the user, it may be better for user

privacy to avoid sending unnecessary information. In the Reconnect Exchange, the optional PeerInfo SHOULD be omitted unless some critical data has changed.

Peer devices that randomize their layer-2 address to prevent tracking can do this whenever the user resets the EAP-NOOB association. During the lifetime of the association, the PeerId is a unique identifier that can be used to track the peer in the access network. Later versions of this specification may consider updating the PeerId at each Reconnect Exchange. In that case, it is necessary to consider how the authenticator and access-network administrators can recognize and blacklist misbehaving peer devices and how to avoid loss of synchronization between the server and the peer if messages are lost during the identifier update.

6.6. Downgrading threats

The fingerprint Hoob protects all the information exchanged in the Initial Exchange, including the cryptosuite negotiation. The message authentication codes MACs and MACp also protect the same information. The message authentication codes MACs2 and MACp2 protect information exchanged during key renegotiation in the Reconnect Exchange. This prevents downgrading attacks to weaker cryptosuites as long as the possible attacks take more time than the maximum time allowed for the EAP-NOOB completion. This is typically the case for recently discovered cryptanalytic attacks.

As an additional precaution, the EAP server and peer SHOULD check for downgrading attacks in the Reconnect Exchange. As long as the server or peer saves any information about the other endpoint, it MUST also remember the previously negotiated cryptosuite and MUST NOT accept renegotiation of any cryptosuite that is known to be weaker than the previous one, such as a deprecated cryptosuite.

Integrity of the direction negotiation cannot be verified in the same way as the integrity of the cryptosuite negotiation. That is, if the OOB channel used in an application is critically insecure in one direction, a man-in-the-middle attacker could modify the negotiation messages and thereby cause that direction to be used. Applications that support OOB messages in both directions SHOULD therefore ensure that the OOB channel has sufficiently strong security in both directions. While this is a theoretical vulnerability, it could arise in practice if EAP-NOOB is deployed in unexpected applications. However, most devices acting as the peer are likely to support only one direction of exchange, in which case interfering with the direction negotiation can only prevent the completion of the protocol.

The long-term shared key material Kz in the persistent EAP-NOOB association is established with an ECDHE key exchange when the peer and server are first associated. It is a weaker secret than a manually configured random shared key because advances in cryptanalysis against the used ECDHE curve could eventually enable the attacker to recover Kz. EAP-NOOB protects against such attacks by allowing cryptosuite upgrades in the Reconnect Exchange and by updating shared key material Kz whenever the cryptosuite is upgraded. We do not expect the cryptosuite upgrades to be frequent, but if one becomes necessary, the upgrade can be made without manual resetting and reassociation of the peer devices.

6.7. Recovery from loss of last message

The EAP-NOOB Completion Exchange, as well as the Reconnect Exchange with cryptosuite update, result in a persistent state change that should take place either on both endpoints or on neither; otherwise, the result is a state mismatch that requires user action to resolve. The state mismatch can occur if the final EAP response of the exchanges is lost. In the Completion Exchange, the loss of the final response (Type=4) results in the peer moving to Registered (4) state and creating a persistent EAP-NOOB association while the server stays in an ephemeral state (1 or 2). In the Reconnect Exchange, the loss of the final response (Type=7) results in the peer moving to the Registered (4) state and updating its persistent key material Kz while the server stays in the Reconnecting (3) state and keeps the old key material.

The state mismatch is an example of an unavoidable problem in distributed systems: it is theoretically impossible to guarantee synchronous state changes in endpoints that communicate asynchronously. The protocol will always have one critical message that may get lost, so that one side commits to the state change and the other side does not. In EAP, the critical message is the final response from the peer to the server. While the final response is normally followed by EAP-Success, [RFC3748] section 4.2 states that the peer MAY assume that the EAP-Success was lost and the authentication was successful. Furthermore, EAP methods in the peer do not receive notification of the EAP-Success message from the parent EAP state machine [RFC4137]. For these reasons, EAP-NOOB on the peer side commits to a state change already when it sends the final response.

The best available solution to the loss of the critical message is to keep trying. EAP retransmission behavior defined in Section 4.3 of [RFC3748] suggests 3-5 retransmissions. In the absence of an attacker, this would be sufficient to reduce the probability of failure to an acceptable level. However, a determined attacker on

the in-band channel can drop the final EAP-Response message and all subsequent retransmissions. In the Completion Exchange (KeyingMode=0) and in the Reconnect Exchange with cryptosuite upgrade (KeyingMode=3), this could result in state mismatch and persistent denial of service until user resets the peer state.

EAP-NOOB implements its own recovery mechanism that allows unlimited retries of the Reconnect Exchange. When the DoS attacker eventually stops dropping packets on the in-band channel, the protocol will recover. The logic for this recovery mechanism is specified in Section 3.4.2.

EAP-NOOB does not implement the same kind of retry mechanism in the Completion Exchange. The reason is that there is always a user involved in the initial association process, and the user can repeat the OOB Step to complete the association after the DoS attacker has left. On the other hand, Reconnect Exchange needs to work without user involvement.

6.8. EAP security claims

EAP security claims are defined in section 7.2.1 of [RFC3748]. The security claims for EAP-NOOB are listed in Table 9.

Security property	EAP-NOOB claim
Authentication mechanism	ECDHE key exchange with out-of-band authentication
Protected cryptosuite negotiation	yes
Mutual authentication	yes
Integrity protection	yes
Replay protection	yes
Key derivation	yes
Key strength	The specified cryptosuites provide key strength of at least 128 bits.
Dictionary attack protection	yes
Fast reconnect	yes
Cryptographic binding	not applicable
Session independence	yes
Fragmentation	no
Channel binding	yes (The ServerInfo and PeerInfo can be used to convey integrity-protected channel properties such as network SSID or peer MAC address.)

Table 9: EAP security claims

7. References

7.1. Normative references

- [NIST-DH] Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2, May 2013, <<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf>>.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<https://www.rfc-editor.org/info/rfc2104>>.
- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [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>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <<https://www.rfc-editor.org/info/rfc6761>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.

- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/info/rfc7518>>.
- [RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <<https://www.rfc-editor.org/info/rfc7542>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [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>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

7.2. Informative references

- [BluetoothPairing] Bluetooth, SIG, "Simple pairing whitepaper", Technical report , 2007.
- [EUI-48] Institute of Electrical and Electronics Engineers, "802-2014 IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.", IEEE Standard 802-2014. , June 2014.
- [IEEE-802.1X] Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X-2004. , December 2004.
- [mcrl2] Groote, J. and M. Mousavi, "Modeling and analysis of communicating systems", The MIT press , 2014, <<https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems>>.

- [proverif] Blanchet, B., Smyth, B., Cheval, V., and M. Sylvestre, "ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial", The MIT press, 2018, <<http://prosecco.gforge.inria.fr/personal/bblanche/proverif/>>.
- [RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <<https://www.rfc-editor.org/info/rfc2904>>.
- [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>>.
- [RFC4266] Hoffman, P., "The gopher URI Scheme", RFC 4266, DOI 10.17487/RFC4266, November 2005, <<https://www.rfc-editor.org/info/rfc4266>>.
- [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>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [Sethi14] Sethi, M., Oat, E., Di Francesco, M., and T. Aura, "Secure Bootstrapping of Cloud-Managed Ubiquitous Displays", Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014), pp. 739-750, Seattle, USA, September 2014, <<http://dx.doi.org/10.1145/2632048.2632049>>.
- [Sethi19] Sethi, M., Peltonen, A., and T. Aura, "Misbinding Attacks on Secure Device Pairing", 2019, <<https://arxiv.org/abs/1902.07550>>.

Appendix A. Exchanges and events per state

Figure 11 shows how the EAP server chooses the exchange type depending on the server and peer states. In the state combinations marked with hyphen "-", there is no possible exchange and user action is required to make progress. Note that peer state 4 is omitted from the table because the peer never connects to the server when the peer is in that state. The table also shows the handling of errors in each exchange. A notable detail is that the recipient of error code 2003 moves to state 1.

peer states	exchange chosen by server	next peer and server states
server state: Unregistered (0)		
0..2 3	Initial Exchange -	both 1 (0 on error) no change, notify user
server state: Waiting for OOB (1)		
0 1 2 3	Initial Exchange Waiting Exchange Completion Exchange -	both 1 (0 on error) both 1 (no change on error) both 4 (A) no change, notify user
server state: OOB Received (2)		
0 1 2 3	Initial Exchange Completion Exchange Completion Exchange -	both 1 (0 on error) both 4 (B) both 4 (A) no change, notify user
server state: Reconnecting (3) or Registered (4)		
0..2 3	- Reconnect Exchange	no change, notify user both 4 (3 on error)

(A) peer to 1 on error 2003, no other changes on error

(B) server to 1 on error 2003, no other changes on error

Figure 11: How server chooses the exchange type

Figure 12 lists the local events that can take place in the server or peer. Both the server and peer output and accept OOB messages in

association state 1, leading the receiver to state 2. Communication errors and timeouts in states 0..2 lead back to state 0, while similar errors in states 3..4 lead to state 3. Application request for rekeying (e.g. to refresh session keys or to upgrade cryptosuite) also takes the association from state 3..4 to state 3. User can always reset the association state to 0. Recovering association data, e.g. from a backup, leads to state 3.

server/ peer state	possible local events on server and peer	next state
1	OoB Output*	1
1	OoB Input*	2 (1 on error)
0..2	Timeout/network failure	0
3..4	Timeout/network failure	3
3..4	Rekeying request	3
0..4	User resets peer state	0
0..4	Association state recovery	3

Figure 12: Local events on server and peer

Appendix B. Application-specific parameters

Table 10 lists OoB channel parameters that need to be specified in each application that makes use of EAP-NOOB. The list is not exhaustive and is included for the convenience of implementors only.

Parameter	Description
OobDirs	Allowed directions of the OOB channel
OobMessageEncoding	How the OOB message data fields are encoded for the OOB channel
SleepTimeDefault	Default minimum time in seconds that the peer should sleep before the next Waiting Exchange
OobRetries	Number of received OOB messages with invalid Hoob after which the receiver moves to Unregistered (0) state
NoobTimeout	How many seconds the sender of the OOB message remembers the sent Noob value. The RECOMMENDED value is 3600 seconds.
ServerInfoMembers	Required members in ServerInfo
PeerInfoMembers	Required members in PeerInfo

Table 10: OOB channel characteristics

Appendix C. ServerInfo and PeerInfo contents

The ServerInfo and PeerInfo fields in the Initial Exchange and Reconnect Exchange enable the server and peer, respectively, send information about themselves to the other endpoint. They contain JSON objects whose structure may be specified separately for each application and each type of OOB channel. ServerInfo and PeerInfo MAY contain auxiliary data needed for the OOB channel messaging and for EAP channel binding. Table 11 lists some suggested data fields for ServerInfo.

Data field	Description
ServerName	String that may be used to aid human identification of the server.
ServerURL	Prefix string when the OOB message is formatted as URL, as suggested in Appendix E.
SSIDList	List of wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix D. JSON array of UTF-8 encoded SSID strings.
Base64SSIDList	List of wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix D. JSON array of SSIDs, each of which is base64url encoded without padding. Peer SHOULD send at most one of the fields SSIDList and Base64SSIDList in PeerInfo, and the server SHOULD ignore SSIDList if Base64SSIDList is included.

Table 11: Suggested ServerInfo data fields

PeerInfo typically contains auxiliary information for identifying and managing peers on the application level at the server end. Table 12 lists some suggested data fields for PeerInfo.

Data field	Description
PeerName	String that may be used to aid human identification of the peer.
Manufacturer	Manufacturer or brand string.
Model	Manufacturer-specified model string.
SerialNumber	Manufacturer-assigned serial number.
MACAddress	Peer link-layer identifier (EUI-48) in the 12-digit base-16 form [EUI-48]. The string MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.
SSID	Wireless network SSID for channel binding. The SSID is a UTF-8 string.
Base64SSID	Wireless network SSID for channel binding. The SSID is base64url encoded. Peer SHOULD send at most one of the fields SSID and Base64SSID in PeerInfo, and the server SHOULD ignore SSID if Base64SSID is included.
BSSID	Wireless network BSSID (EUI-48) in the 12-digit base-16 form [EUI-48]. The string MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.

Table 12: Suggested PeerInfo data fields

Appendix D. EAP-NOOB roaming

AAA architectures [RFC2904] allow for roaming of network-connected appliances that are authenticated over EAP. While the peer is roaming in a visited network, authentication still takes place between the peer and an authentication server at its home network. EAP-NOOB supports such roaming by assigning a Realm to the peer. After the Realm has been assigned, the peer's NAI enables the visited network to route the EAP session to the peer's home AAA server.

A peer device that is new or has gone through a hard reset should be connected first to the home network and establish an EAP-NOOB association with its home AAA server before it is able to roam.

After that, it can perform the Reconnect Exchange from the visited network.

Alternatively, the device may provide some method for the user to configure the Realm of the home network. In that case, the EAP-NOOB association can be created while roaming. The device will use the user-assigned Realm in the Initial Exchange, which enables the EAP messages to be routed correctly to the home AAA server.

While roaming, the device needs to identify the networks where the EAP-NOOB association can be used to gain network access. For 802.11 access networks, the server MAY send a list of SSID strings in the ServerInfo JSON object in a member called either SSIDList or Base64SSIDList. The list is formatted as explained in Table 11. If present, the peer MAY use this list as a hint to determine the networks where the EAP-NOOB association can be used for access authorization, in addition to the access network where the Initial Exchange took place.

Appendix E. OOB message as URL

While EAP-NOOB does not mandate any particular OOB communication channel, typical OOB channels include graphical displays and emulated NFC tags. In the peer-to-server direction, it may be convenient to encode the OOB message as a URL, which is then encoded as a QR code for displays and printers or as an NDEF record for NFC tags. A user can then simply scan the QR code or NFC tag and open the URL, which causes the OOB message to be delivered to the authentication server. The URL MUST specify the https protocol i.e. secure connection to the server, so that the man-in-the-middle attacker cannot read or modify the OOB message.

The ServerInfo in this case includes a JSON member called ServerUrl of the following format with maximum length of 60 characters:

```
https://<host>[:<port>]/[<path>]
```

To this, the peer appends the OOB message fields (PeerId, Noob, Hoob) as a query string. PeerId is provided to the peer by the server and might be a 22-character string. The peer base64url encodes, without padding, the 16-byte values Noob and Hoob into 22-character strings. The query parameters MAY be in any order. The resulting URL is of the following format:

```
https://<host>[:<port>]/[<path>]?P=<PeerId>&N=<Noob>&H=<Hoob>
```

The following is an example of a well-formed URL encoding the OOB message (without line breaks):

https://example.com/Noob?P=ZrD7qkcZNoHGbGcN2bN0&N=rMinS0-F4EfCU8D9lJxX_A&H=QvnMp4UGxuQVFaxPW_14UW

Appendix F. Example messages

The message examples in this section are generated with Curve25519 ECDHE test vectors specified in section 6.1 of [RFC7748] (server=Alice, peer=Bob). The direction of the OOB channel negotiated is 2 (server-to-peer). The JSON messages are as follows (line breaks are for readability only).

===== Initial Exchange =====

Identity response:
noob@eap-noob.net

EAP request (type 9):
{"Type":9}

EAP response (type 9):
{"Type":9,"PeerId":"07KRU6OgqX0HIeRfLdnbSW","PeerState":0}

EAP request (type 1):
{"Type":1,"Vers":[1],"PeerId":"07KRU6OgqX0HIeRfLdnbSW","Realm":"noob.example.com","Cryptosuites":[1],"Dirs":3,"ServerInfo":{"Name":"Example","Url":"https://noob.example.com/sendOOB"}}

EAP response (type 1):
{"Type":1,"Verp":1,"PeerId":"07KRU6OgqX0HIeRfLdnbSW","Cryptosuitep":1,"Dirp":2,"PeerInfo":{"Make":"Acme","Type":"None","Serial":"DU-9999","SSID":"Noob1","BSSID":"6c:19:8f:83:c2:80"}}

EAP request (type 2):
{"Type":2,"PeerId":"07KRU6OgqX0HIeRfLdnbSW","PKs":{"kty":"EC","crv":"Curve25519","x":"hSDwCYkwp1R0i33ctD73Wg2_Og0mOBr066SpjqqbTmo"},"Ns":"PYO7NVd9Af3BxEr1lMI6hL8Ck49YxwCjSRPqlC1SPbw","SleepTime":60}

EAP response (type 2):
{"Type":2,"PeerId":"07KRU6OgqX0HIeRfLdnbSW","PKp":{"kty":"EC","crv":"Curve25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrfx-FG-IK08"},"Np":"HIvB6g0n2btpxEcU7YXnWB-451ED6L6veQQd6ugiPFU"}

===== Waiting Exchange =====

Identity response:
noob@eap-noob.net

EAP request (type 9):

```
    {"Type":9}

EAP response (type 9):
    {"Type":9,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PeerState":1}

EAP request (type 3):
    {"Type":3,"PeerId":"07KRU6OgqX0HIeRFldnbSW","SleepTime":60}

EAP response (type 3):
    {"Type":3,"PeerId":"07KRU6OgqX0HIeRFldnbSW"}

===== OOB Step =====

OOB message:
    P=07KRU6OgqX0HIeRFldnbSW&N=x3JlolaPciK4Wa6XlMJxtQ&H=WJ6Covspd50NT2
    RxkLHSeA

===== Completion Exchange =====

Identity response:
    noob@eap-noob.net

EAP request (type 9):
    {"Type":9}

EAP response (type 9):
    {"Type":9,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PeerState":2}

EAP request (type 8):
    {"Type":8,"PeerId":"07KRU6OgqX0HIeRFldnbSW"}

EAP response (type 8):
    {"Type":8,"PeerId":"07KRU6OgqX0HIeRFldnbSW","NoobId":"U0OHwYGCS4nE
    kzk2TPIE6g"}

EAP request (type 4):
    {"Type":4,"PeerId":"07KRU6OgqX0HIeRFldnbSW","NoobId":"U0OHwYGCS4nE
    kzk2TPIE6g","MACs":"APpnhlFLWS2pfJPH5S7N3yr6FJWocuaAiuVrhgh8Xko"}

EAP response (type 4):
    {"Type":4,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACp":"hihGS4v8w4cDy_
    yokNlOyQa87GRBLvMfmF9JFwJ6RrQ"}

===== Reconnect Exchange =====

Identity response:
    noob@eap-noob.net
```

EAP request (type 9):
{"Type":9}

EAP response (type 9):
{"Type":9,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PeerState":3}

EAP request (type 5):
{"Type":5,"Vers":[1],"PeerId":"07KRU6OgqX0HIeRFldnbSW","Cryptosuites":[1],"Realm":"noob.example.com","ServerInfo":{"Name":"Example","Url":"https://noob.example.com/sendOOB"}}

EAP response (type 5):
{"Type":5,"Verp":1,"PeerId":"07KRU6OgqX0HIeRFldnbSW","Cryptosuitep":1,"PeerInfo":{"Make":"Acme","Type":"None","Serial":"DU-9999","SSID":"Noob1","BSSID":"6c:19:8f:83:c2:80"}}

EAP request (type 6):
{"Type":6,"PeerId":"07KRU6OgqX0HIeRFldnbSW","KeyingMode":2,"Ns2":"RDLahHB1IgnmL_FxcynrHurLPkCsrp3G3B_S82WUF4"}

EAP response (type 6):
{"Type":6,"PeerId":"07KRU6OgqX0HIeRFldnbSW","Np2":"jN0_V4P0JoTqwI9VHHQKd9ozUh7tQdc9ABd-j6oTy_4"}

EAP request (type 7):
{"Type":7,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACs2":"TT_B9w-o86C1c1O_rhNxzcf9gJa0_8SiIhyxQecdM70"}

EAP response (type 7):
{"Type":7,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACp2":"GS9f8Mw3mUFvjIDKS54U27xPt6umIrnVXOGLl-iFRKk"}

Appendix G. TODO list

- o

Appendix H. Version history

- o Version 01:
 - * Fixed Reconnection Exchange.
 - * URL examples.
 - * Message examples.
 - * Improved state transition (event) tables.

o Version 02:

- * Reworked the rekeying and key derivation.
- * Increased internal key lengths and in-band nonce and HMAC lengths to 32 bytes.
- * Less data in the persistent EAP-NOOB association.
- * Updated reference [NIST-DH] to Revision 2 (2013).
- * Shorter suggested PeerId format.
- * Optimized the example of encoding OOB message as URL.
- * NoobId in Completion Exchange to differentiate between multiple valid Noob values.
- * List of application-specific parameters in appendix.
- * Clarified the equivalence of Unregistered state and no state.
- * Peer SHOULD probe the server regardless of the OOB channel direction.
- * Added new error messages.
- * Realm is part of the persistent association and can be updated.
- * Clarified error handling.
- * Updated message examples.
- * Explained roaming in appendix.
- * More accurate definition of timeout for the Noob nonce.
- * Additions to security considerations.

o Version 03:

- * Clarified reasons for going to Reconnecting state.
- * Included Verp in persistent state.
- * Added appendix on suggested ServerInfo and PeerInfo fields.
- * Exporting PeerId and SessionId.

- * Explicitly specified next state after OOB Step.
- * Clarified the processing of an expired OOB message and unrecognized NoobId.
- * Enabled protocol version upgrade in Reconnect Exchange.
- * Explained handling of redundant received OOB messages.
- * Clarified where raw and base64url encoded values are used.
- * Cryptosuite must specify the detailed format of the JWK object.
- * Base64url encoding in JSON strings is done without padding.
- * Simplified explanation of PeerId, Realm and NAI.
- * Added error codes for private and experimental use.
- * Updated the security considerations.
- o Version 04:
 - * Recovery from synchronization failure due to lost last response.
- o Version 05:
 - * Kz identifier added to help recovery from lost last messages.
 - * Error message codes changed for better structure.
 - * Improved security considerations section.
- o Version 06:
 - * Kz identifier removed to enable PeerId anonymization in the future.
 - * Clarified text on when to use server-assigned realm.
 - * Send PeerId and PeerState in a separate request-reponse pair, not in NAI.
 - * New subsection for the common handshake in all exchanges to avoid repetition.
- o Version 07:

- * Updated example messages.
- * Added pointers to new implementation in Contiki.

Appendix I. Acknowledgments

Aleksi Peltonen modeled the protocol specification with the mCRL2 formal specification language. Shiva Prasad TP and Raghavendra MS implemented parts of the protocol with wpa_supplicant and hostapd. Their inputs helped us in improving the specification.

The authors would also like to thank Rhys Smith and Josh Howlett for providing valuable feedback as well as new use cases and requirements for the protocol. Thanks to Eric Rescorla, Darshak Thakore, Stefan Winter, and Hannes Tschofenig for interesting discussions in this problem space.

Authors' Addresses

Tuomas Aura
Aalto University
Aalto 00076
Finland

EMail: tuomas.aura@aalto.fi

Mohit Sethi
Ericsson
Jorvas 02420
Finland

EMail: mohit@piuha.net

Network Working Group
INTERNET-DRAFT
Updates: 5247, 5281, 7170
Category: Standards Track
<draft-dekok-emu-tls-eap-types-00.txt>
11 February 2019

DeKok, Alan
FreeRADIUS

TLS-based EAP types and TLS 1.3
draft-dekok-emu-tls-eap-types-00.txt

Abstract

EAP-TLS [RFC5216] is being updated for TLS 1.3 in [EAPTLS]. Many other EAP [RFC3748] and [RFC5247] types also depend on TLS, such as FAST [RFC4851], TTLS [RFC5281], TEAP [RFC7170], 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 August 11, 2019.

Copyright Notice

Copyright (c) 2019 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. FAST and TEAP	6
3. Application Data	6
4. Security Considerations	7
5. IANA Considerations	8
6. References	8
6.1. Normative References	8
6.2. Informative References	9

1. Introduction

EAP-TLS is being updated for TLS 1.3 in [EAPTLS]. Many other EAP types also depend on TLS, such as FAST [RFC4851], TTLS [RFC5281], TEAP [RFC7170], and possibly many vendor specific EAP methods. All of these methods use key derivation functions that rely on the information which is no longer available in TLS 1.3. As such, all of those methods are incompatible with TLS 1.3.

We wish to enable the use of TLS 1.3 in the wider Internet community. As such, it is necessary to update the above EAP types. 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 [EAPTLS] apply to other EAP methods that wish to use TLS 1.3. Implementations of other methods that wish to use TLS 1.3 MUST follow the guidelines in [EAPTLS].

There are, however a few key differences between EAP-TLS and other TLS-based EAP methods that necessitate this document. The simplest difference is that [EAPTLS] uses the EAP-TLS type ID (0x0D) in a number of calculations. That value should change for other method types.

More complex differences include derivation of additional keying material, as in FAST [RFC4851].

2.1. Key Derivation

The key derivation for TLS-based EAP methods depends on the value of the Type-Code as defined by [IANA]. The most important definition is of the Type-Code:

Type-Code = EAP Method type

The Type-Code is defined to be 1 octet for values smaller than 256, otherwise it is a 32-bit number (four octets), in network byte order.

Unless otherwise discussed below, the key derivation functions for all TLS-based EAP types are defined as follows:

```
Key_Material = TLS-Exporter("EXPORTER_EAP_TLS_Key_Material", Type-
Code, 128) IV = TLS-Exporter("EXPORTER_EAP_TLS_IV",
Type-Code, 64) Method-Id = TLS-
Exporter("EXPORTER_EAP_TLS_Method-Id", Type-Code, 64) Session-Id
= Type-Code || Method-Id MSK = Key_Material(0, 63) EMSK
= Key_Material(64, 127) Enc-RECV-Key = MSK(0, 31) Enc-SEND-Key =
MSK(32, 63) RECV-IV = IV(0, 31) SEND-IV = IV(32, 63)
```

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

2.2. FAST and TEAP

EAP-FAST [RFC4851] and TEAP [RFC7170] cannot use the above derivation. Those methods use an inner tunnel EMSK to calculate the outer EMSK. As such, those key derivations cannot use the above derivation.

EAP-FAST previously used a PAC, which is a type of pre-shared key (PSK). Such uses are deprecated in TLS 1.3. As such, PAC provisioning is no longer part of EAP-FAST when TLS 1.3 is used.

TBD: Is this true? Comments from EAP-FAST people are useful here.

The key derivation for FAST and TEAP are similar enough that they gave be given together here. The only difference is the Type-Code. All derivations not given here are the same as given above in the previous section.

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

For FAST, the session_key_seed is also used as the key_block, as defined in [RFC4851] Section 5.1.

```
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] | MSK[j], 60)
    S-IMCK[j] = first 40 octets of IMCK[j]
    CMK[j] = last 20 octets of IMCK[j]
```

Where | denotes concatenation.

```
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)
```

3. Application Data

Unlike previous TLS version, TLS 1.3 continues negotiation after the TLS session has been initialized. Some implementations use the TLS "Finished" state as a signal that application data is now available, and an "inner tunnel" session can now be negotiated. As noted in [RFC8446], TLS 1.3 may include a "NewSessionTicket" after the "Finished" state. This change can cause many implementations to fail.

In order to correct this failure, implementations MUST also check if

"Application Data" is available for a TLS connection. If the underlying TLS connection is still performing negotiations, then implementations MUST NOT send, or expect to receive application data in the TLS session.

We note that some TLS Application Programming Interfaces (APIs) signal the availability of application data by returning zero octets of application data, where they previously had returned an error which signalled that negotiation should continue. For those APIs, implementations SHOULD treat the combination of the "Finished" state and the availability of zero octets of application data as a signal that TLS negotiation has completed, and that the tunneled process can begin.

[EAPTLS] uses an empty application record to indicate that negotiation has finished. Methods which use "inner tunnel" methods should instead begin their "inner tunnel" negotiation by sending type-specific application data.

4. Security Considerations

[EAPTLS] 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 sends a 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 the NewSessionTicket. Then prior to authentication, the malicious client can abort the authentication session. The malicious client can then use 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.

5. 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 derivation of Key_Material, IV and Method-Id as defined above in Section ?

The labels above need to be added to the "TLS Exporter Labels" registry.

* TBD

6. References

6.1. 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", RC 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.
- [EAPTLS]
Mattsson, J., and Sethi, M., "Using EAP-TLS with TLS 1.3", draft-ietf-emu-eap-tls13-03, November 2018.
- [IANA]
<https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml#eap-numbers-4>

6.2. Informative References

- [PEAP]
Palekar, A. et al, "Protected EAP Protocol (PEAP)", draft-josefsson-pppext-eap-tls-eap-06.txt, March 2003.
- [MSPEAP]
<https://msdn.microsoft.com/en-us/library/cc238354.aspx>
- [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.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 26, 2020

O. Friel
R. Barnes
Cisco
R. Shekh-Yusef
Avaya
October 24, 2019

ACME Integrations
draft-friel-acme-integrations-02

Abstract

This document outlines multiple advanced use cases and integrations that ACME facilitates without any modifications or enhancements required to the base ACME specification. The use cases include ACME integration with EST, BRSKI and TEAP.

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 April 26, 2020.

Copyright Notice

Copyright (c) 2019 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	2
2. Terminology	3
3. ACME Integration with EST	3
4. ACME Integration with BRSKI	6
5. ACME Integration with BRSKI Default Cloud Registrar	8
6. ACME Integration with TEAP	10
7. ACME Integration with TEAP-BRSKI	13
8. IANA Considerations	16
9. Security Considerations	16
10. Informative References	16
Appendix A. Comments	17
Authors' Addresses	17

1. Introduction

ACME [RFC8555] defines a protocol that a certificate authority (CA) and an applicant can use to automate the process of domain name ownership validation and X.509 (PKIX) certificate issuance. The protocol is rich and flexible and enables multiple use cases that are not immediately obvious from reading the specification. This document explicitly outlines multiple advanced ACME use cases including:

- o ACME integration with EST [RFC7030]
- o ACME integration with BRSKI
[I-D.ietf-anima-bootstrapping-keyinfra]
- o ACME integration with BRSKI Default Cloud Registrar
[I-D.friel-anima-brski-cloud]
- o ACME integration with TEAP [RFC7170]
- o ACME integration with TEAP-BRSKI [I-D.lear-eap-teap-brski]

The integrations with EST, BRSKI (which is based upon EST), and TEAP enable automated certificate enrolment for devices. ACME for subdomains [I-D.friel-acme-subdomains] outlines how ACME can be used by a client to obtain a certificate for a subdomain identifier from a certificate authority where client has fulfilled a challenge against a parent domain but does not need to fulfil a challenge against the explicit subdomain. This is a useful optimisation when ACME is used to issue certificates for large numbers of devices as it reduces the

domain ownership proof traffic (DNS or HTTP) and ACME traffic overhead, but is not a necessary requirement.

2. 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.

The following terms are used in this document:

- o BRSKI: Bootstrapping Remote Secure Key Infrastructures [I-D.ietf-anima-bootstrapping-keyinfra]
- o CA: Certificate Authority
- o CMC: Certificate Management over CMS
- o CSR: Certificate Signing Request
- o EST: Enrollment over Secure Transport [RFC7030]
- o FQDN: Fully Qualified Domain Name
- o RA: PKI Registration Authority
- o TEAP: Tunneled Extensible Authentication Protocol [RFC7170]

3. ACME Integration with EST

EST [RFC7030] defines a mechanism for clients to enroll with a PKI Registration Authority by sending CMC messages over HTTP. EST section 1 states:

"Architecturally, the EST service is located between a Certification Authority (CA) and a client. It performs several functions traditionally allocated to the Registration Authority (RA) role in a PKI."

EST section 1.1 states that:

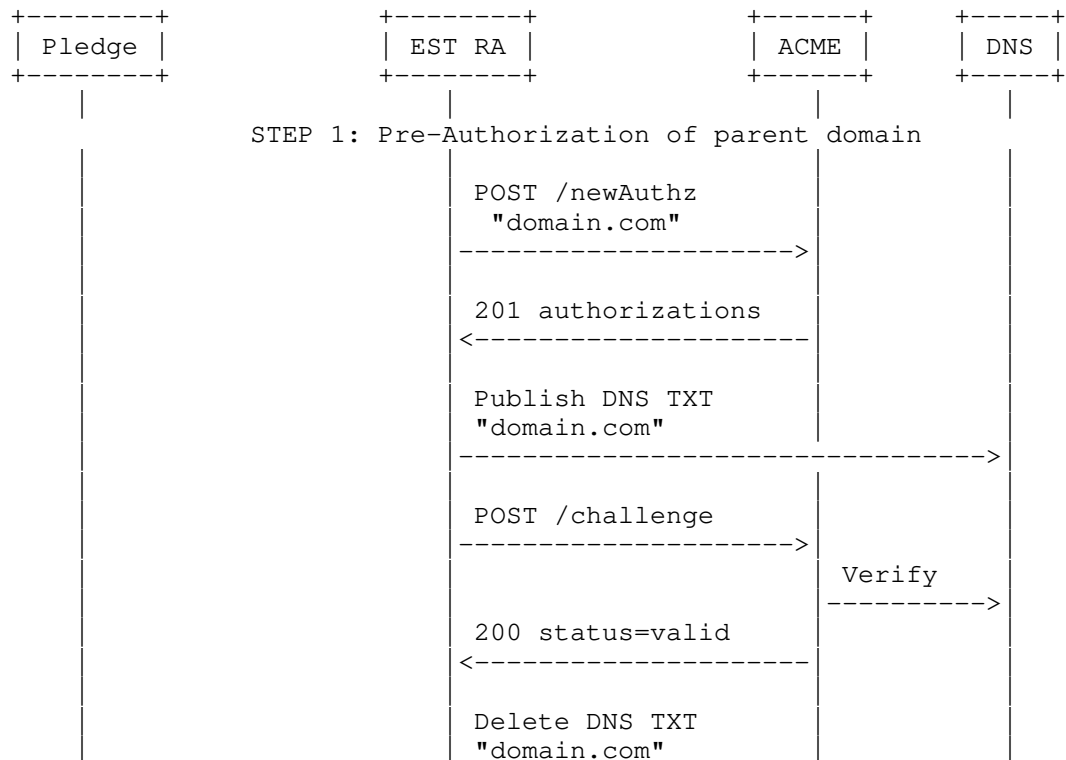
"For certificate issuing services, the EST CA is reached through the EST server; the CA could be logically "behind" the EST server or embedded within it."

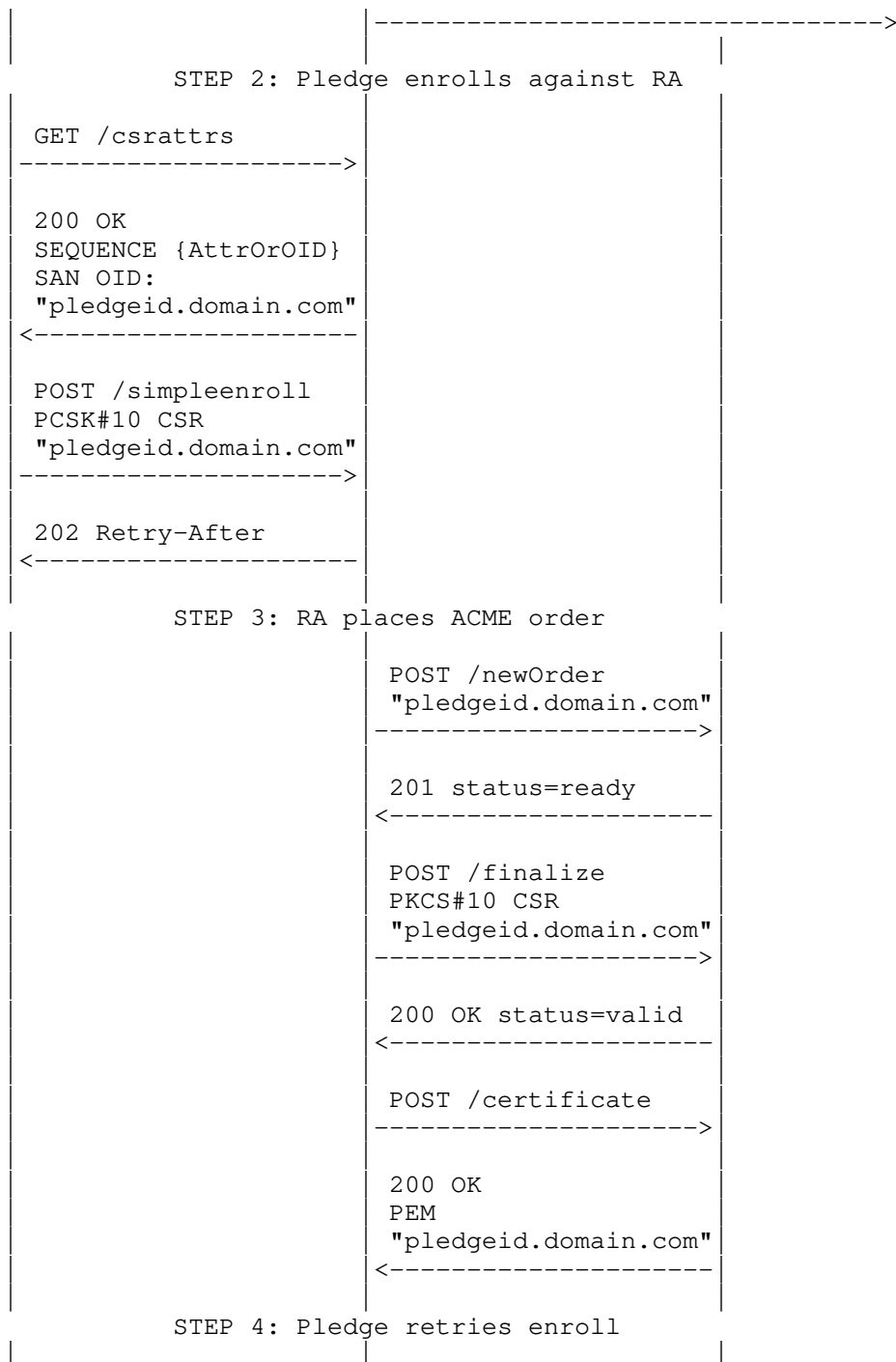
When the CA is logically "behind" the EST RA, EST does not specify how the RA communicates with the CA. EST section 1 states:

"The nature of communication between an EST server and a CA is not described in this document."

This section outlines how ACME could be used for communication between the EST RA and the CA. The example call flow leverages [I-D.friel-acme-subdomains] and shows the RA proving ownership of a parent domain, with individual client certificates being subdomains under that parent domain. This is an optimisation that reduces DNS and ACME traffic overhead. The RA could of course prove ownership of every explicit client certificate identifier.

The call flow illustrates the client calling the EST /csrattrs API before calling the EST /simpleenroll API. This enables the EST server to indicate to the client what attributes it expects the client to include in the CSR request send in the /simpleenroll API. For example, EST servers could use this mechanism to tell the client what fields to include in the CSR Subject and Subject Alternative Name fields.





```

POST /simpleenroll
PCSK#10 CSR
"pledgeid.domain.com"
----->

200 OK
PKCS#7
"pledgeid.domain.com"
<-----

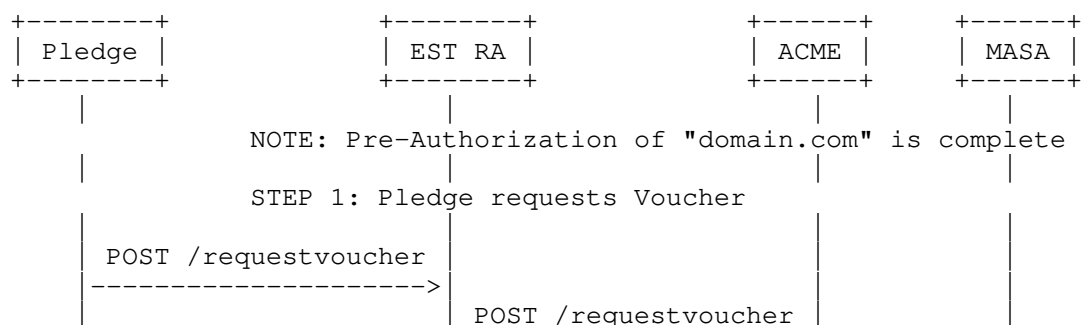
```

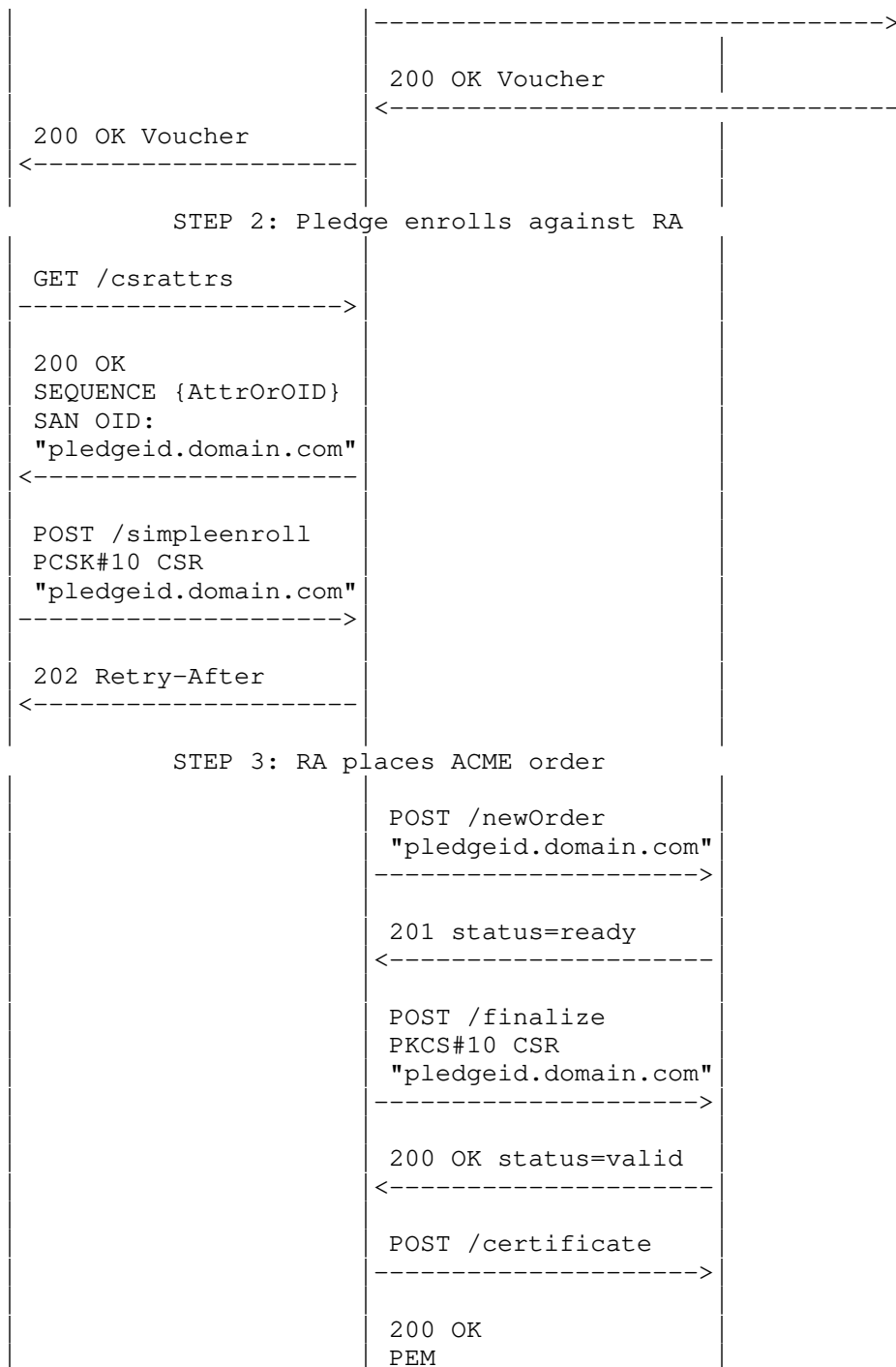
4. ACME Integration with BRSKI

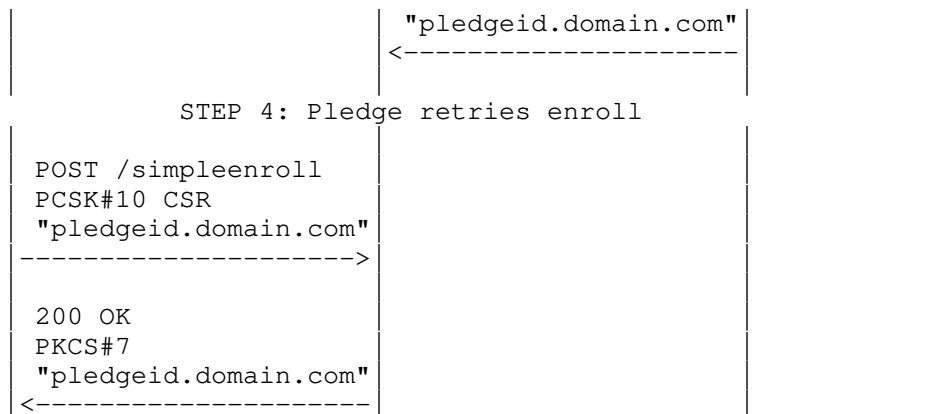
BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] is based upon EST [RFC7030] and defines how to autonomically bootstrap PKI trust anchors into devices via means of signed vouchers. EST certificate enrollment may then optionally take place after trust has been established. BRSKI voucher exchange and trust establishment are based on EST extensions and the certificate enrollment part of BRSKI is fully based on EST. Similar to EST, BRSKI does not define how the EST RA communicates with the CA. Therefore, the mechanisms outlined in the previous section for using ACME as the communications protocol between the EST RA and the CA are equally applicable to BRSKI.

The following call flow shows how ACME may be integrated into a full BRSKI voucher plus EST enrollment workflow. For brevity, it assumes that the EST RA has previously proven ownership of a parent domain and that pledge certificate identifiers are a subdomain of that parent domain. The domain ownership exchanges between the RA, ACME and DNS are not shown. Similarly, not all BRSKI interactions are shown and only the key protocol flows involving voucher exchange and EST enrollment are shown.

Similar to the EST section above, the client calls EST /csrattrs API before calling the EST /simpleenroll API. This enables the server to indicate what fields the pledge should include in the CSR that the client sends in the /simpleenroll API.



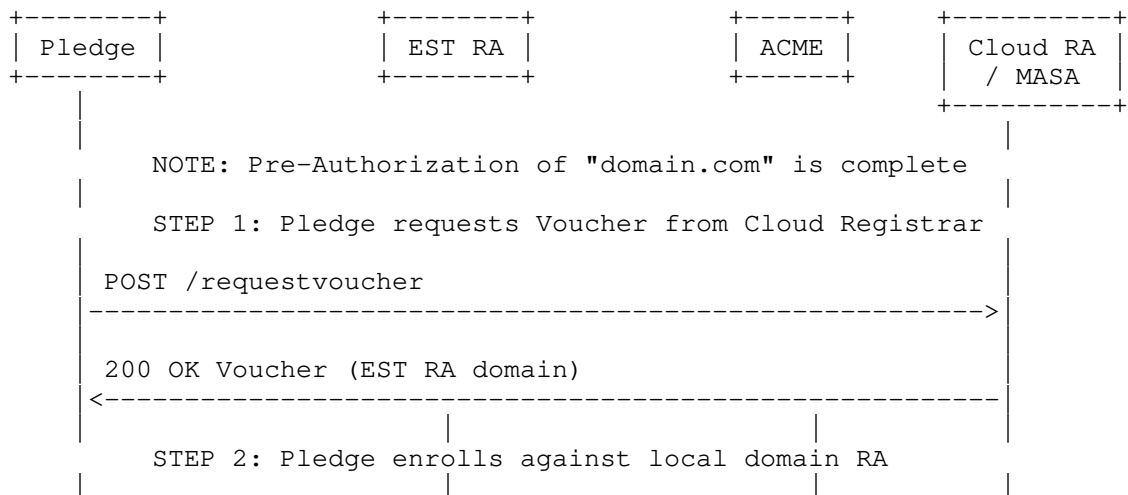


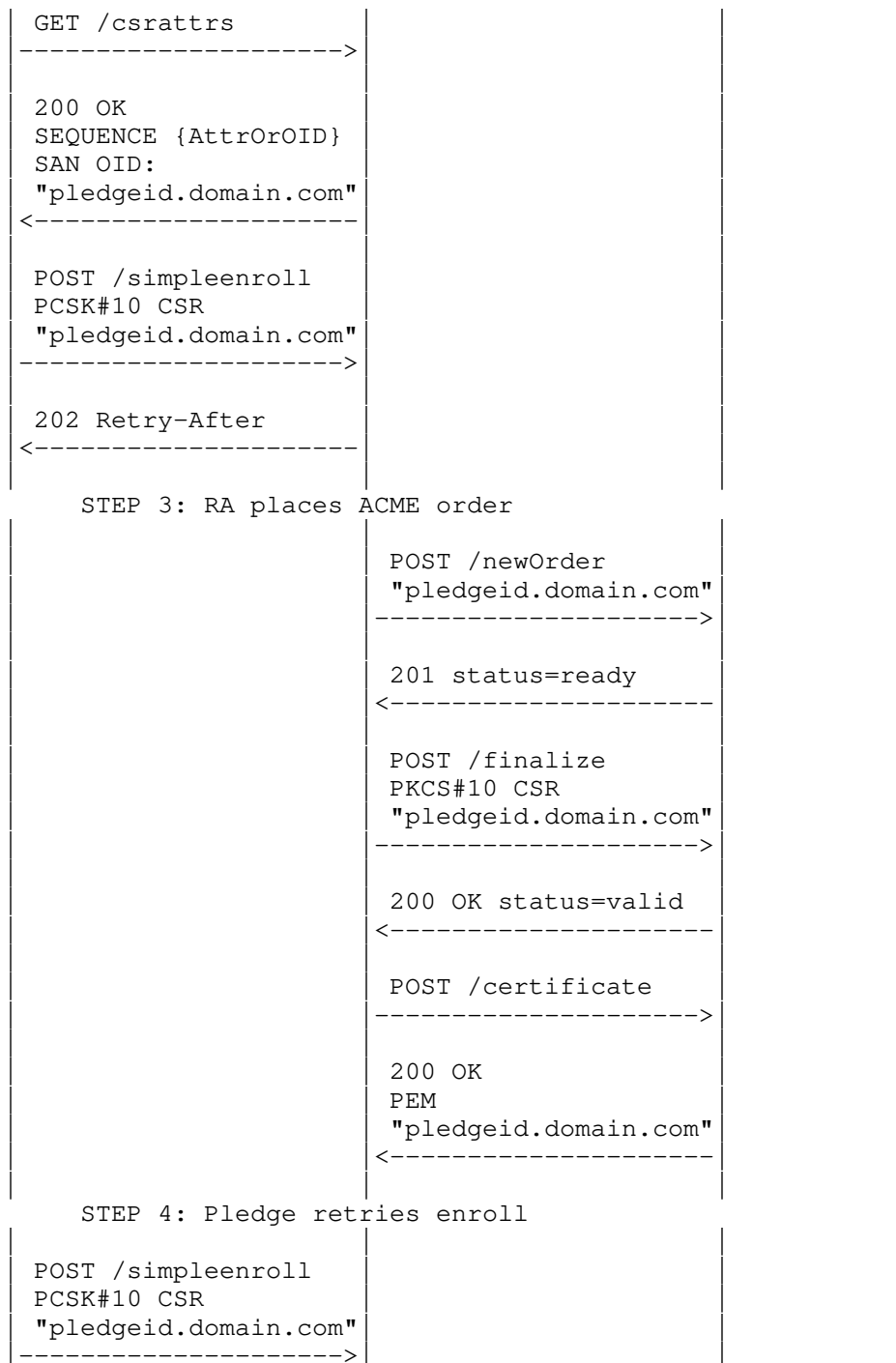


5. ACME Integration with BRSKI Default Cloud Registrar

BRSKI Cloud Registrar [I-D.friel-anima-brski-cloud] specifies the behaviour of a BRSKI Cloud Registrar, and how a pledge can interact with a BRSKI Cloud Registrar when bootstrapping. Similar to the local domain registrar BRSKI flow, ACME can be easily integrated with a cloud registrar bootstrap flow.

BRSKI cloud registrar is flexible and allows for multiple different local domain discovery and redirect scenarios. In the example illustrated here, the extension to [RFC8366] Vouchers which is defined in [[TODO ID-TBD]] and allows the specification of a bootstrap DNS domain is leveraged. This extension allows the cloud registrar to specify the local domain RA that the pledge should connect to for the purposes of EST enrollment.





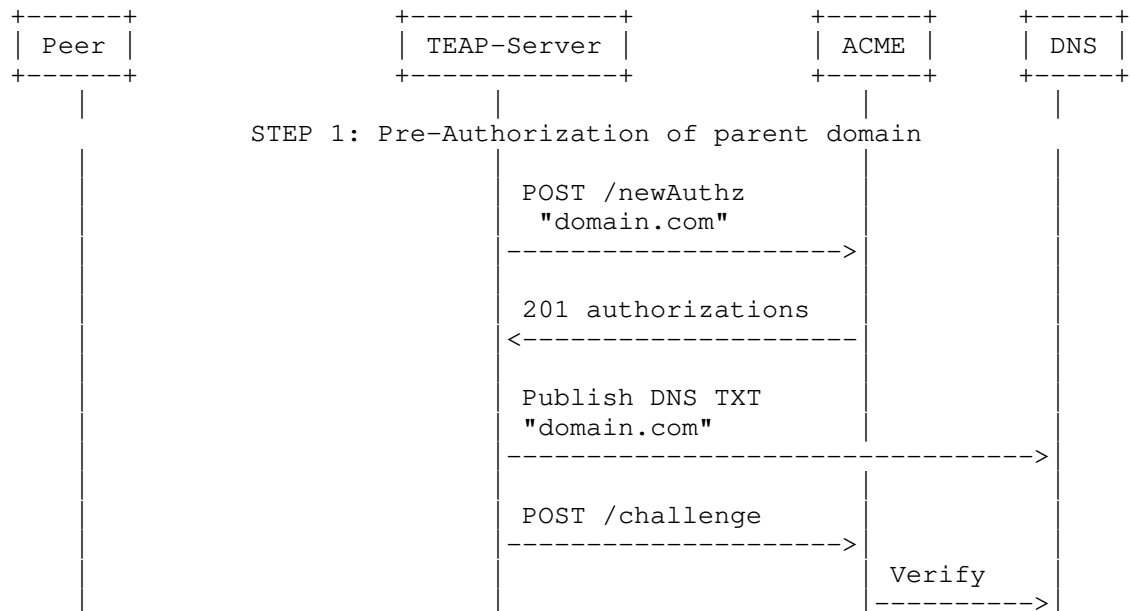
200 OK PKCS#7 "pledgeid.domain.com" <-----			
---	--	--	--

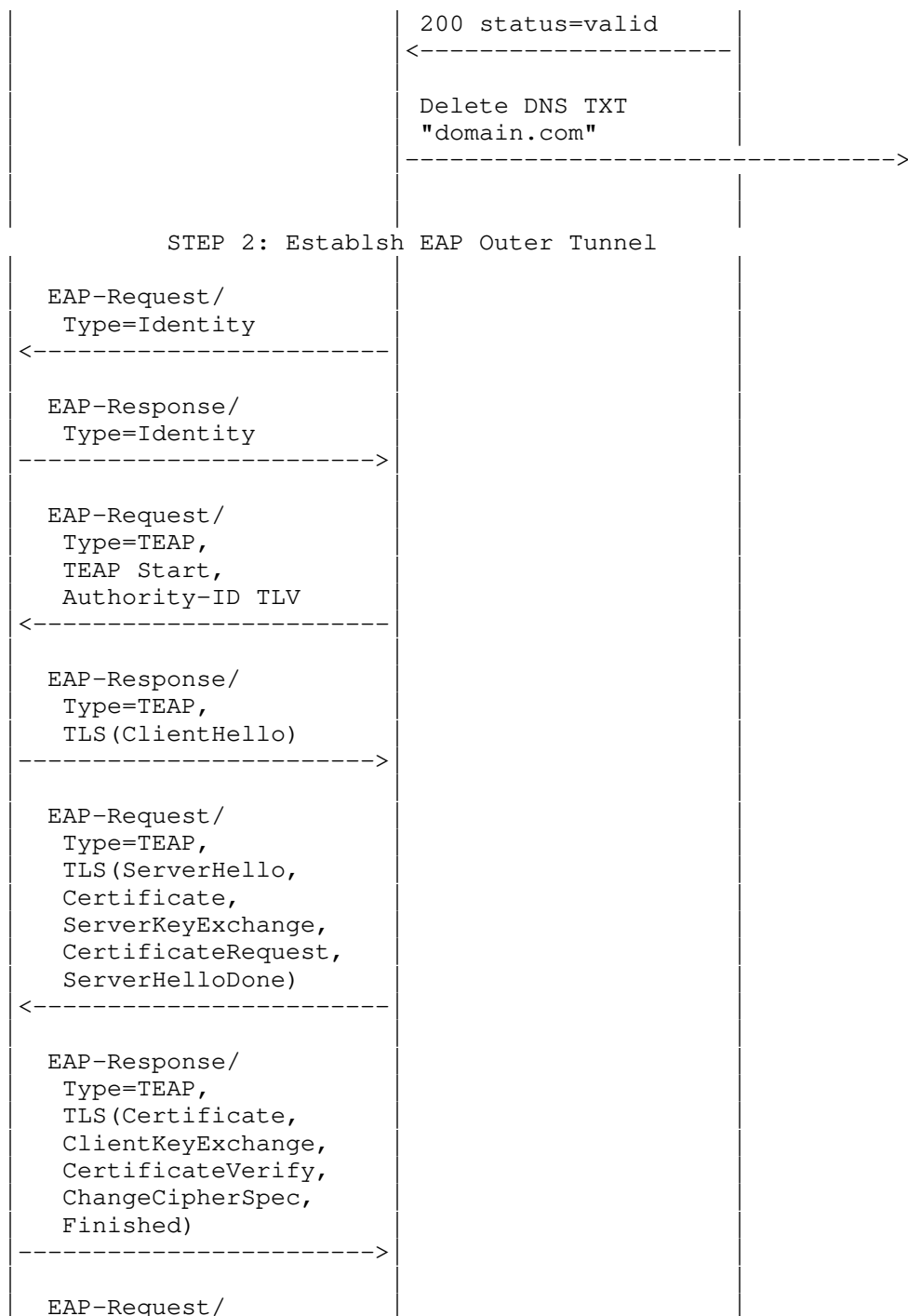
6. ACME Integration with TEAP

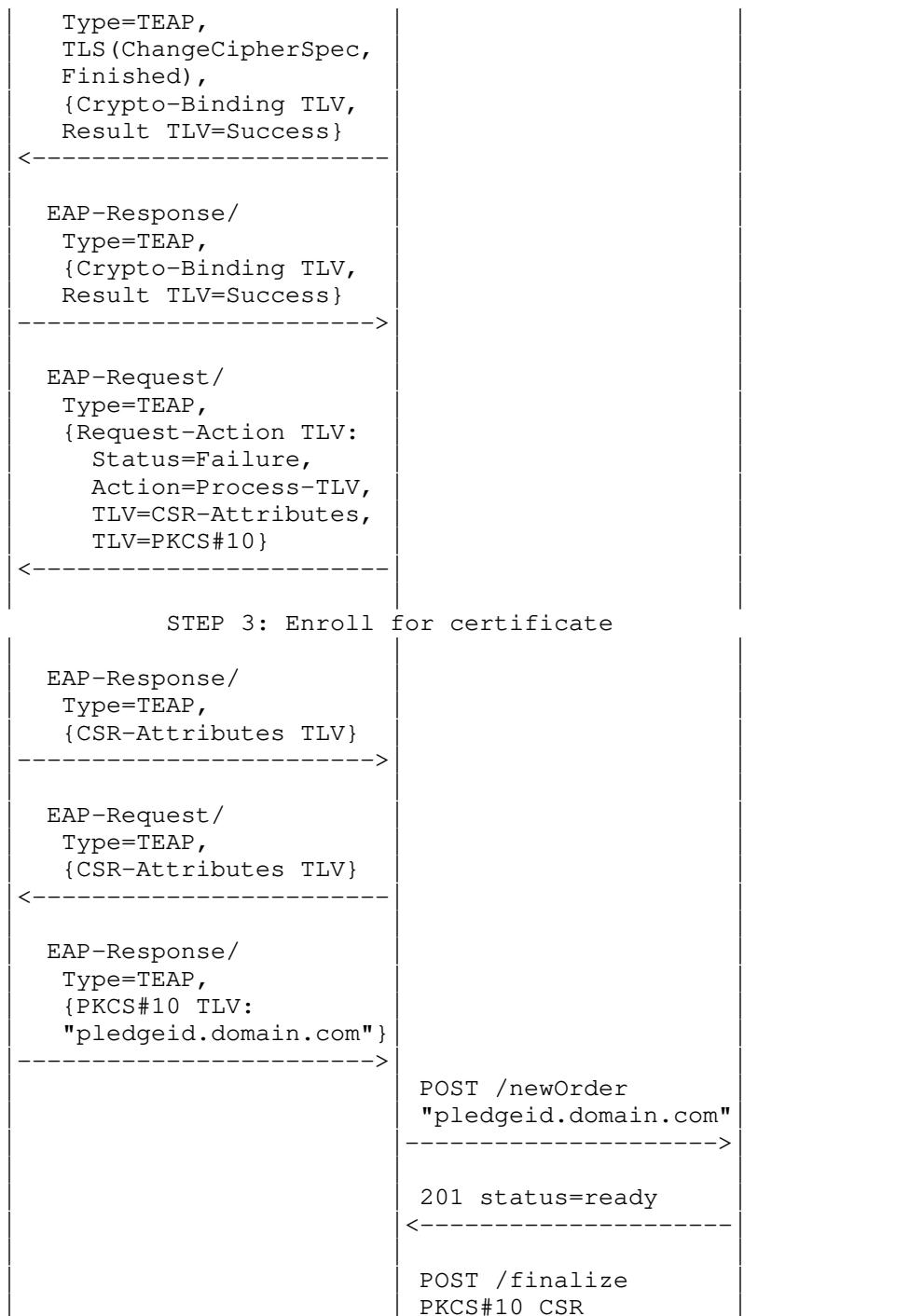
TEAP [RFC7170] defines a tunnel-based EAP method that enables secure communication between a peer and a server by using TLS to establish a mutually authenticated tunnel. TEAP enables certificate provisioning within the tunnel. TEAP does not define how the TEAP server communicates with the CA.

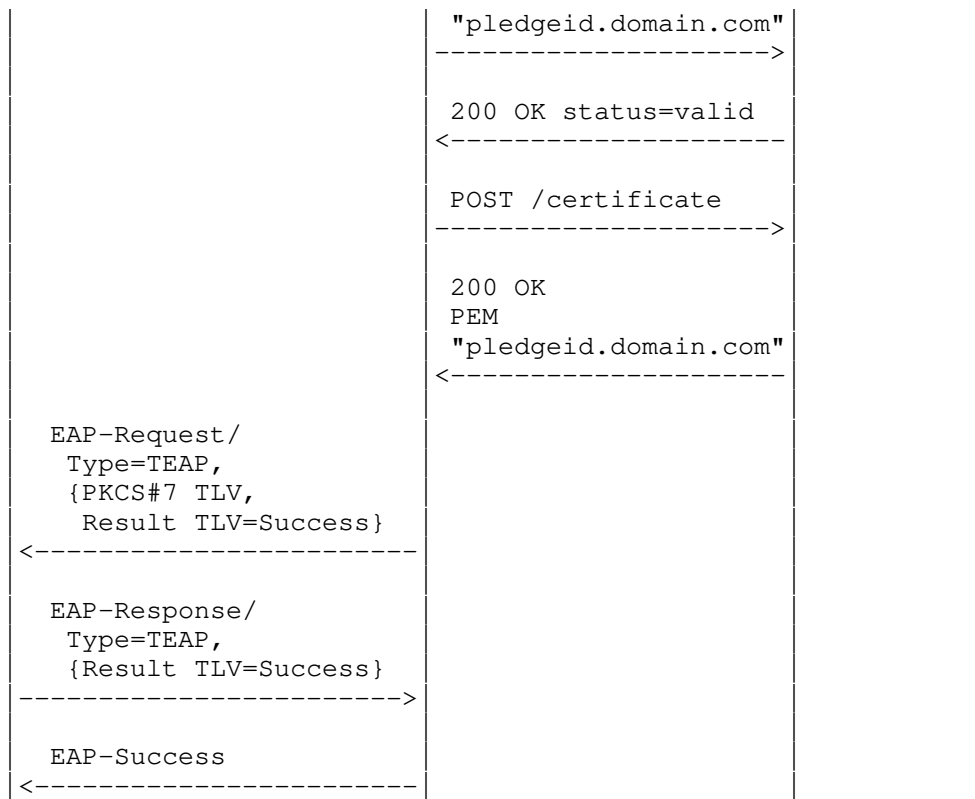
This section outlines how ACME could be used for communication between the TEAP server and the CA. The example call flow leverages [I-D.friel-acme-subdomains] and shows the TEAP server proving ownership of a parent domain, with individual client certificates being subdomains under that parent domain.

The example illustrates the TEAP server sending a Request-Action TLV including a CSR-Attributes TLV instructing the peer to send a CSR-Attributes TLV to the server. This enables the server to indicate what fields the peer should include in the CSR that the peer sends in the PKCS#10 TLV. For example, the TEAP server could instruct the peer what Subject or SAN entries to include in its CSR.







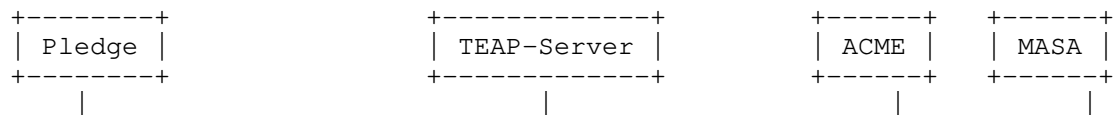


7. ACME Integration with TEAP-BRSKI

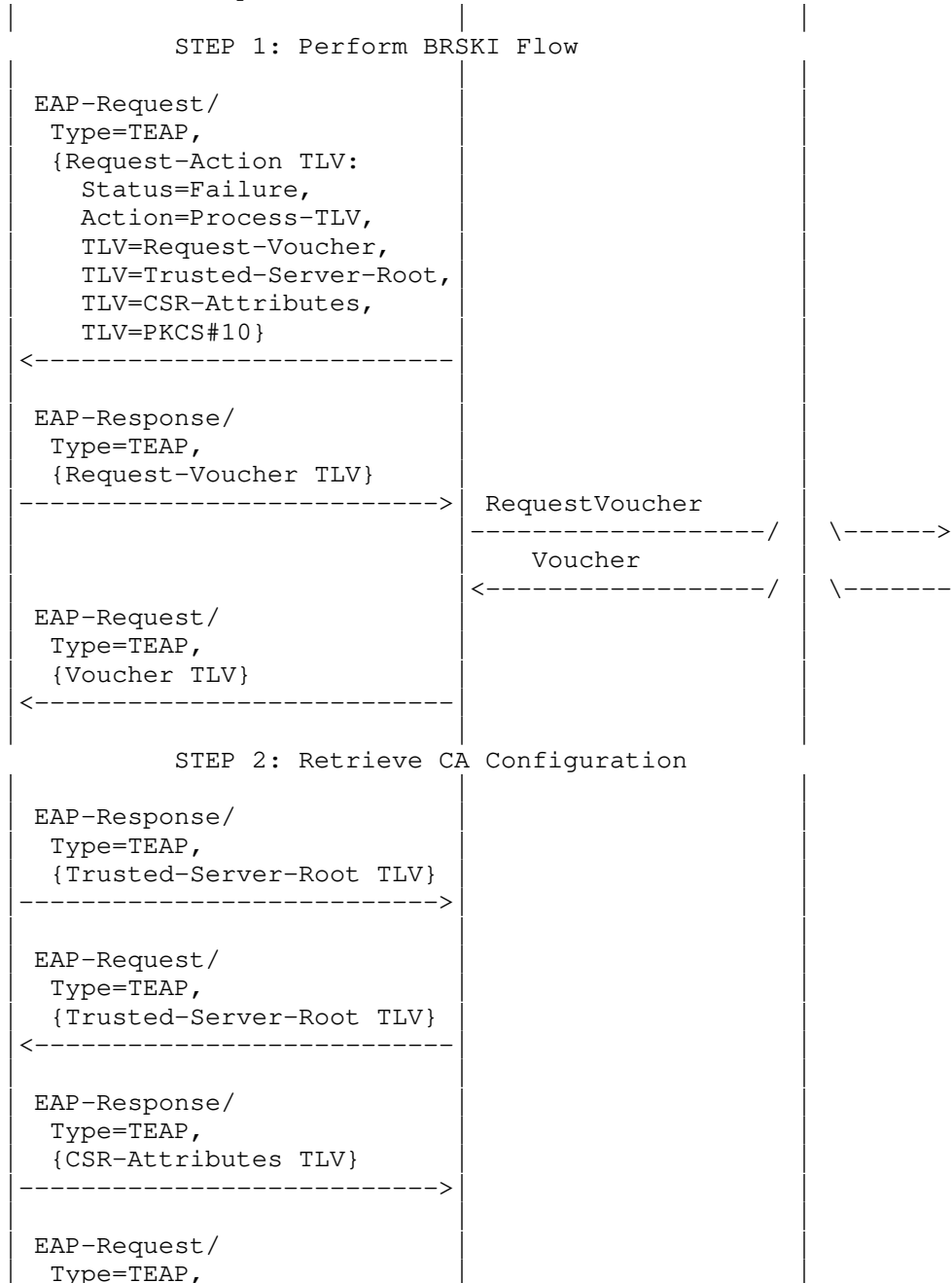
TEAP-BRSKI [I-D.lear-eap-teap-brski] defines how to execute BRSKI at layer 2 inside a TEAP tunnel. Similar to the TEAP proposal in the previous section, BRSKI-TEAP leverages the existing TEAP PKXS#10 and PKCS#7 mechanisms for certificate enrollment, and does not define how the TEAP server communicates with the CA.

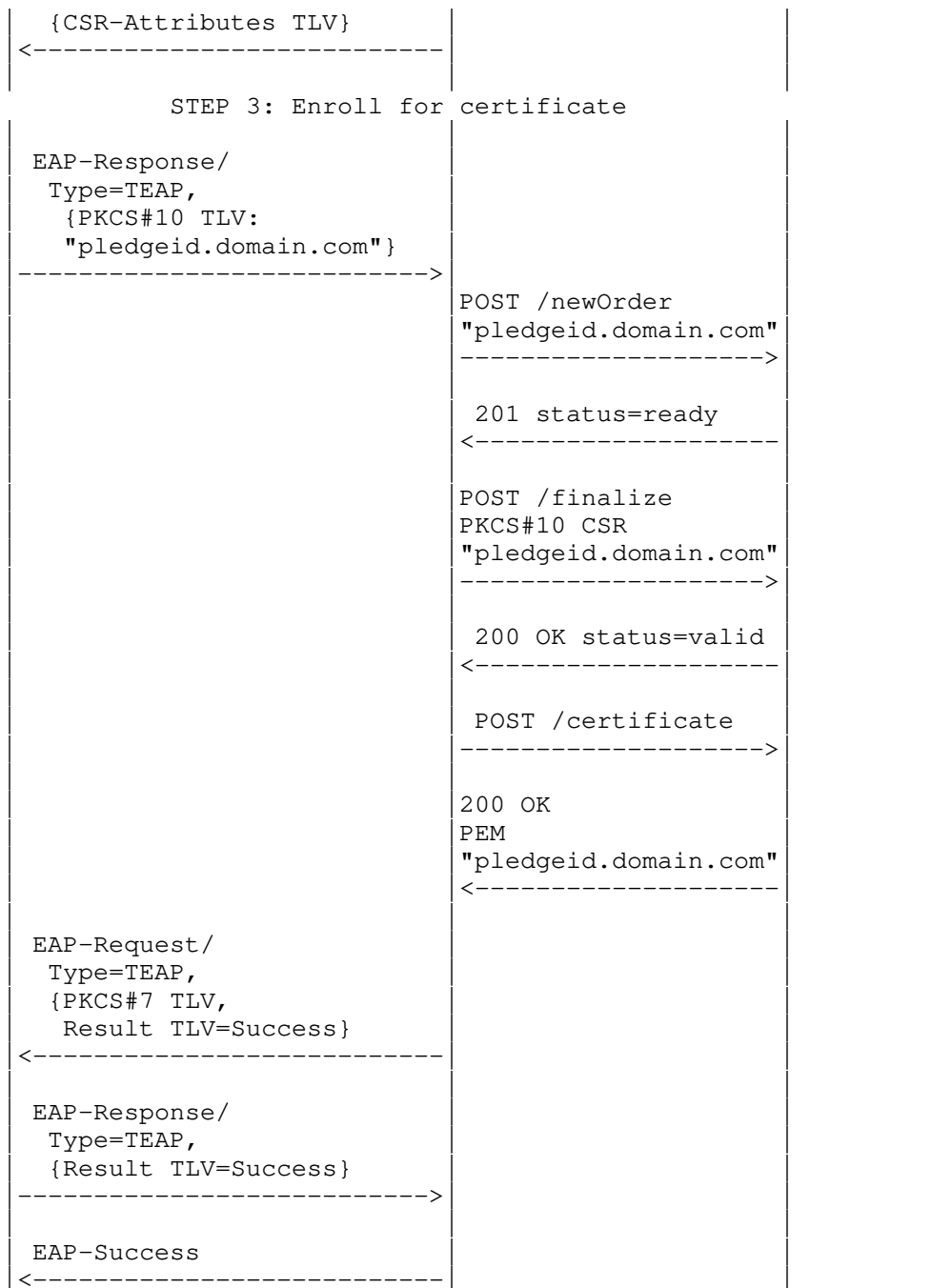
This section outlines how ACME could be used for communication between the TEAP server and the CA, and how this fits in with the TEAP-BRSKI proposal.

Similar to baseline TEAP, the TEAP server can use the CSR-Attributes TLV to tell the peer what attributes to include in its CSR request.



NOTE: Pre-Authorization of "domain.com" is complete and EAP outer tunnel is established as outlined in the previous section





8. IANA Considerations

[todo]

9. Security Considerations

[todo]

10. Informative References

[I-D.friel-acme-subdomains]

Friel, O., Barnes, R., and T. Hollebeek, "ACME for Subdomains", draft-friel-acme-subdomains-00 (work in progress), October 2019.

[I-D.friel-anima-brski-cloud]

Friel, O., Shekh-Yusef, R., and M. Richardson, "BRSKI Cloud Registrar", draft-friel-anima-brski-cloud-01 (work in progress), October 2019.

[I-D.ietf-anima-bootstrapping-keyinfra]

Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-28 (work in progress), September 2019.

[I-D.lear-eap-teap-brski]

Lear, E., Friel, O., Cam-Winget, N., and D. Harkins, "Bootstrapping Key Infrastructure over EAP", draft-lear-eap-teap-brski-04 (work in progress), September 2019.

[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>>.

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

[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>>.

- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <<https://www.rfc-editor.org/info/rfc8555>>.

Appendix A. Comments

Authors' Addresses

Owen Friel
Cisco

Email: ofriel@cisco.com

Richard Barnes
Cisco

Email: rlb@ipv.sx

Rifaat Shekh-Yusef
Avaya

Email: rifaat.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 8, 2020

J. Arkko
K. Norrman
V. Torvinen
Ericsson
November 5, 2019

Perfect-Forward Secrecy for the Extensible Authentication Protocol
Method for Authentication and Key Agreement (EAP-AKA' PFS)
draft-ietf-emu-aka-pfs-01

Abstract

Many different attacks have been reported as part of revelations associated with pervasive surveillance. Some of the reported attacks involved compromising smart cards, such as attacking SIM card manufacturers and operators in an effort to compromise shared secrets stored on these cards. Since the publication of those reports, manufacturing and provisioning processes have gained much scrutiny and have improved. However, the danger of resourceful attackers for these systems is still a concern.

This specification is an optional extension to the EAP-AKA' authentication method which was defined in RFC 5448 (to be superseded by draft-ietf-emu-rfc5448bis). The extension, when negotiated, provides Perfect Forward Secrecy for the session key generated as a part of the authentication run in EAP-AKA'. This prevents an attacker who has gained access to the long-term pre-shared secret in a SIM card from being able to decrypt all past communications. In addition, if the attacker stays merely a passive eavesdropper, the extension prevents attacks against future sessions. This forces attackers to use active attacks instead.

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 May 8, 2020.

Copyright Notice

Copyright (c) 2019 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
2. Protocol Design and Deployment Objectives	4
3. Background	5
3.1. AKA	5
3.2. EAP-AKA' Protocol	6
3.3. Attacks Against Long-Term Shared Secrets in Smart Cards .	8
4. Requirements Language	8
5. Protocol Overview	8
6. Extensions to EAP-AKA'	11
6.1. AT_PUB_ECDHE	11
6.2. AT_KDF_PFS	11
6.3. New Key Derivation Function	14
6.4. ECDHE Groups	15
6.5. Message Processing	15
6.5.1. EAP-Request/AKA'-Identity	15
6.5.2. EAP-Response/AKA'-Identity	16
6.5.3. EAP-Request/AKA'-Challenge	16
6.5.4. EAP-Response/AKA'-Challenge	16
6.5.5. EAP-Request/AKA'-Reauthentication	17
6.5.6. EAP-Response/AKA'-Reauthentication	17
6.5.7. EAP-Response/AKA'-Synchronization-Failure	17
6.5.8. EAP-Response/AKA'-Authentication-Reject	17
6.5.9. EAP-Response/AKA'-Client-Error	18
6.5.10. EAP-Request/AKA'-Notification	18
6.5.11. EAP-Response/AKA'-Notification	18
7. Security Considerations	18
8. IANA Considerations	22
9. References	22
9.1. Normative References	22

9.2. Informative References	23
Appendix A. Change Log	24
Appendix B. Acknowledgments	25
Authors' Addresses	25

1. Introduction

Many different attacks have been reported as part of revelations associated with pervasive surveillance. Some of the reported attacks involved compromising smart cards, such as attacking SIM card manufacturers and operators in an effort to compromise shared secrets stored on these cards. Such attacks are conceivable, for instance, during the manufacturing process of cards, or during the transfer of cards and associated information to the operator. Since the publication of reports about such attacks, manufacturing and provisioning processes have gained much scrutiny and have improved.

However, the danger of resourceful attackers attempting to gain information about SIM cards is still a concern. They are a high-value target and concern a large number of people. Note that the attacks are largely independent of the used authentication technology; the issue is not vulnerabilities in algorithms or protocols, but rather the possibility of someone gaining unlawful access to key material. While the better protection of manufacturing and other processes is essential in protecting against this, there is one question that we as protocol designers can ask. Is there something that we can do to limit the consequences of attacks, should they occur?

The authors want to provide a public specification of an extension that helps defend against one aspect of pervasive surveillance. This is important, given the large number of users such practices may affect. It is also a stated goal of the IETF to ensure that we understand the surveillance concerns related to IETF protocols and take appropriate countermeasures [RFC7258]. This document does that for EAP-AKA'.

This specification is an optional extension to the EAP-AKA' authentication method [RFC5448] (to be superseded by [I-D.ietf-emu-rfc5448bis]). The extension, when negotiated, provides Perfect Forward Secrecy for the session key generated as a part of the authentication run in EAP-AKA'. This prevents an attacker who has gained access to the long-term pre-shared secret in a SIM card from being able to decrypt all past communications. In addition, if the attacker stays merely a passive eavesdropper, the extension prevents attacks against future sessions. This forces attackers to use active attacks instead. As with other protocols, an active attacker with access to the long-term key material will of course be

able to attack all future communications, but risks detection, particularly if done at scale.

Attacks against AKA authentication via compromising the long-term secrets in the SIM cards have been an active discussion topic in many contexts. Perfect forward secrecy is on the list of features for the next release of 3GPP (5G Phase 2), and this document provides a basis for providing this feature in a particular fashion.

It should also be noted that 5G network architecture includes the use of the EAP framework for authentication. While any methods can be run, the default authentication method within that context will be EAP-AKA'. As a result, improvements in EAP-AKA' security have a potential to improve security for large number of users.

2. Protocol Design and Deployment Objectives

This extension specified here re-uses large portions of the current structure of 3GPP interfaces and functions, with the rationale that this will make the construction more easily adopted. In particular, the construction maintains the interface between the Universal Subscriber Identification Module (USIM) and the mobile terminal intact. As a consequence, there is no need to roll out new credentials to existing subscribers. The work is based on an earlier paper [TrustCom2015], and uses much of the same material, but applied to EAP rather than the underlying AKA method.

It has been a goal to implement this change as an extension of the widely supported EAP-AKA' method, rather than a completely new authentication method. The extension is implemented as a set of new, optional attributes, that are provided alongside the base attributes in EAP-AKA'. Old implementations can ignore these attributes, but their presence will nevertheless be verified as part of base EAP-AKA' integrity verification process, helping protect against bidding down attacks. This extension does not increase the number of rounds necessary to complete the protocol.

The use of this extension is at the discretion of the authenticating parties. It should be noted that PFS and defenses against passive attacks are by no means a panacea, but they can provide a partial defense that increases the cost and risk associated with pervasive surveillance.

While adding perfect forward secrecy to the existing mobile network infrastructure can be done in multiple different ways, the authors believe that the approach chosen here is relatively easily deployable. In particular:

- o As noted above, no new credentials are needed; there is no change to SIM cards.
- o PFS property can be incorporated into any current or future system that supports EAP, without changing any network functions beyond the EAP endpoints.
- o Key generation happens at the endpoints, enabling highest grade key material to be used both by the endpoints and the intermediate systems (such as access points that are given access to specific keys).
- o While EAP-AKA' is just one EAP method, for practical purposes perfect forward secrecy being available for both EAP-TLS [RFC5216] [I-D.mattsson-eap-tls13] and EAP-AKA' ensures that for many practical systems perfect forward secrecy can be enabled for either all or significant fraction of users.

3. Background

3.1. AKA

AKA is based on challenge-response mechanisms and symmetric cryptography. AKA typically runs in a UMTS Subscriber Identity Module (USIM) or a CDMA2000 (Removable) User Identity Module ((R)UIM). In contrast with its earlier GSM counterparts, 3rd generation AKA provides long key lengths and mutual authentication.

AKA works in the following manner:

- o The identity module and the home environment have agreed on a secret key beforehand.
- o The actual authentication process starts by having the home environment produce an authentication vector, based on the secret key and a sequence number. The authentication vector contains a random part RAND, an authenticator part AUTN used for authenticating the network to the identity module, an expected result part XRES, a 128-bit session key for integrity check IK, and a 128-bit session key for encryption CK.
- o The authentication vector is passed to the serving network, which uses it to authenticate the device.
- o The RAND and the AUTN are delivered to the identity module.
- o The identity module verifies the AUTN, again based on the secret key and the sequence number. If this process is successful (the

AUTN is valid and the sequence number used to generate AUTN is within the correct range), the identity module produces an authentication result RES and sends it to the serving network.

- o The serving network verifies the correct result from the identity module. If the result is correct, IK and CK can be used to protect further communications between the identity module and the home environment.

3.2. EAP-AKA' Protocol

When AKA (and AKA') are embedded into EAP, the authentication on the network side is moved to the home environment; the serving network performs the role of a pass-through authenticator. Figure 1 describes the basic flow in the EAP-AKA' authentication process. The definition of the full protocol behaviour, along with the definition of attributes AT_RANDOM, AT_AUTN, AT_MAC, and AT_RES can be found in [I-D.ietf-emu-rfc5448bis] and [RFC4187].

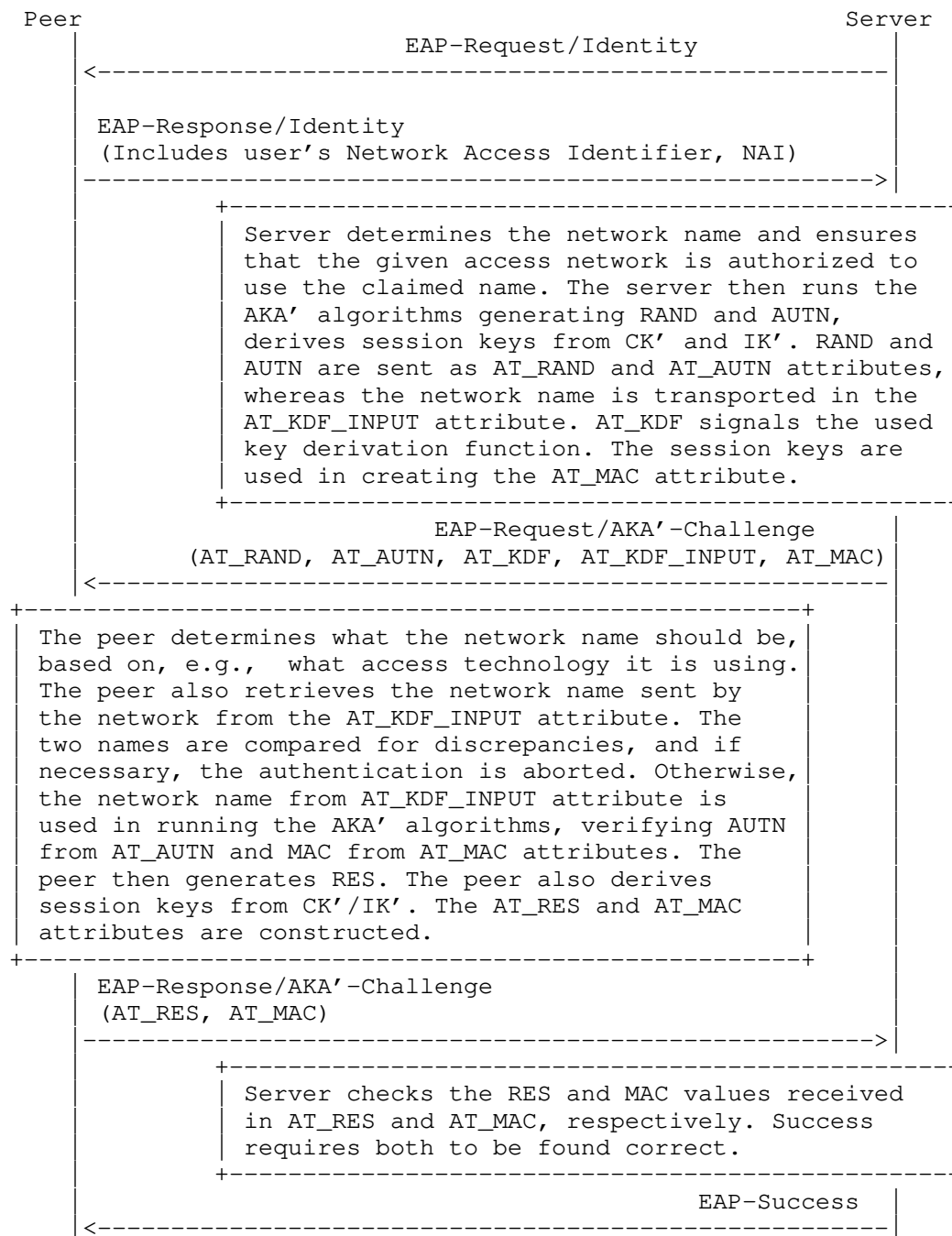


Figure 1: EAP-AKA' Authentication Process

3.3. Attacks Against Long-Term Shared Secrets in Smart Cards

Current 3GPP systems use (U)SIM pre-shared key based protocols and Authentication and Key Agreement (AKA) to authenticate subscribers. The general security properties and potential vulnerabilities of AKA and EAP-AKA' are discussed in [I-D.ietf-emu-rfc5448bis].

An important vulnerability in that discussion relates to the recent reports of compromised long term pre-shared keys used in AKA [Heist2015]. These attacks are not specific to AKA or EAP-AKA', as all security systems fail at least to some extent if key material is stolen. However, the reports indicate a need to look into solutions that can operate at least to an extent under these types of attacks. It is noted in [Heist2015] that some security can be retained even in the face of the attacks by providing Perfect Forward Security (PFS) [DOW1992] for the session key. If AKA would have provided PFS, compromising the pre-shared key would not be sufficient to perform passive attacks; the attacker is, in addition, forced to be a Man-In-The-Middle (MITM) during the AKA run and subsequent communication between the parties.

4. 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.

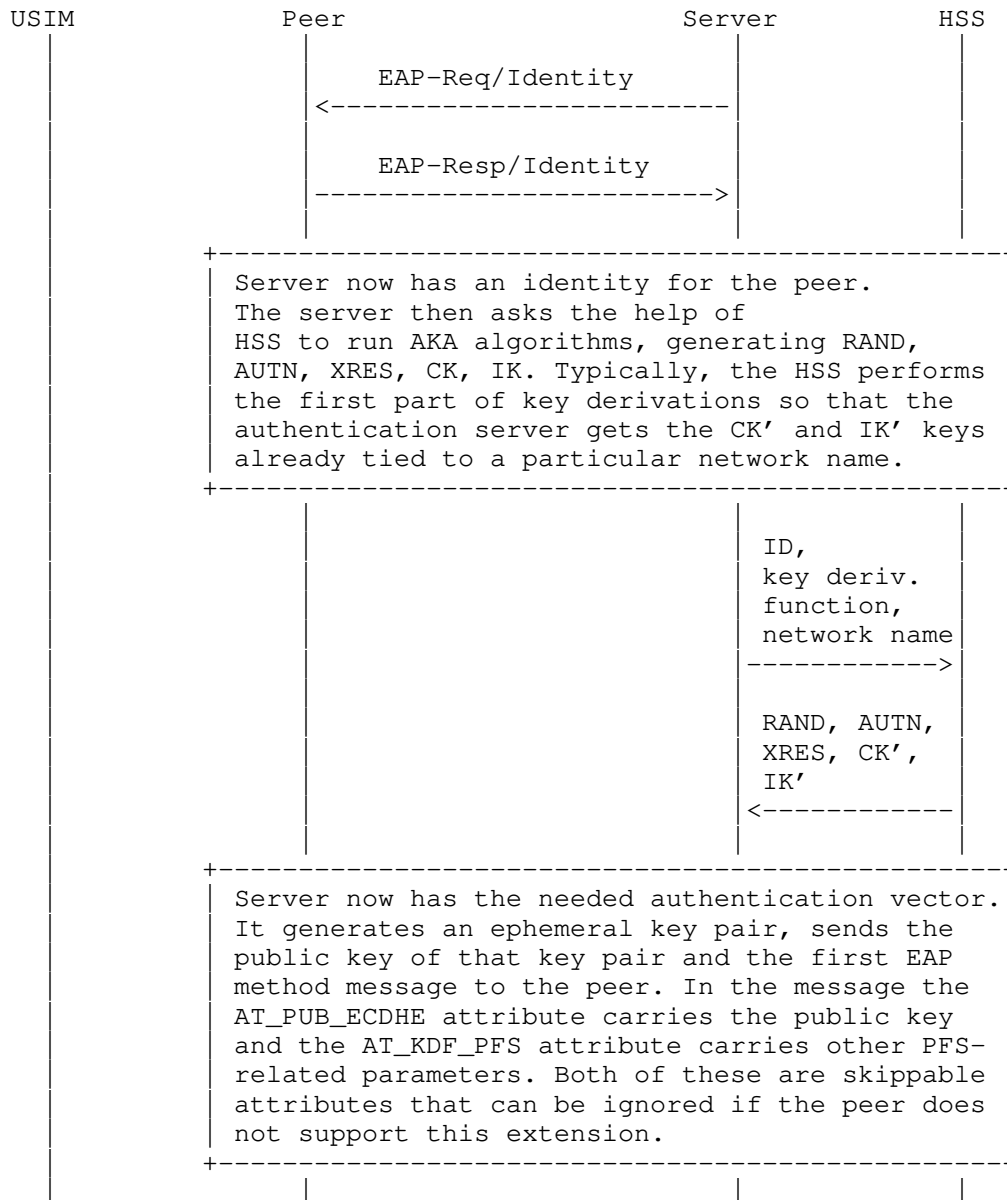
5. Protocol Overview

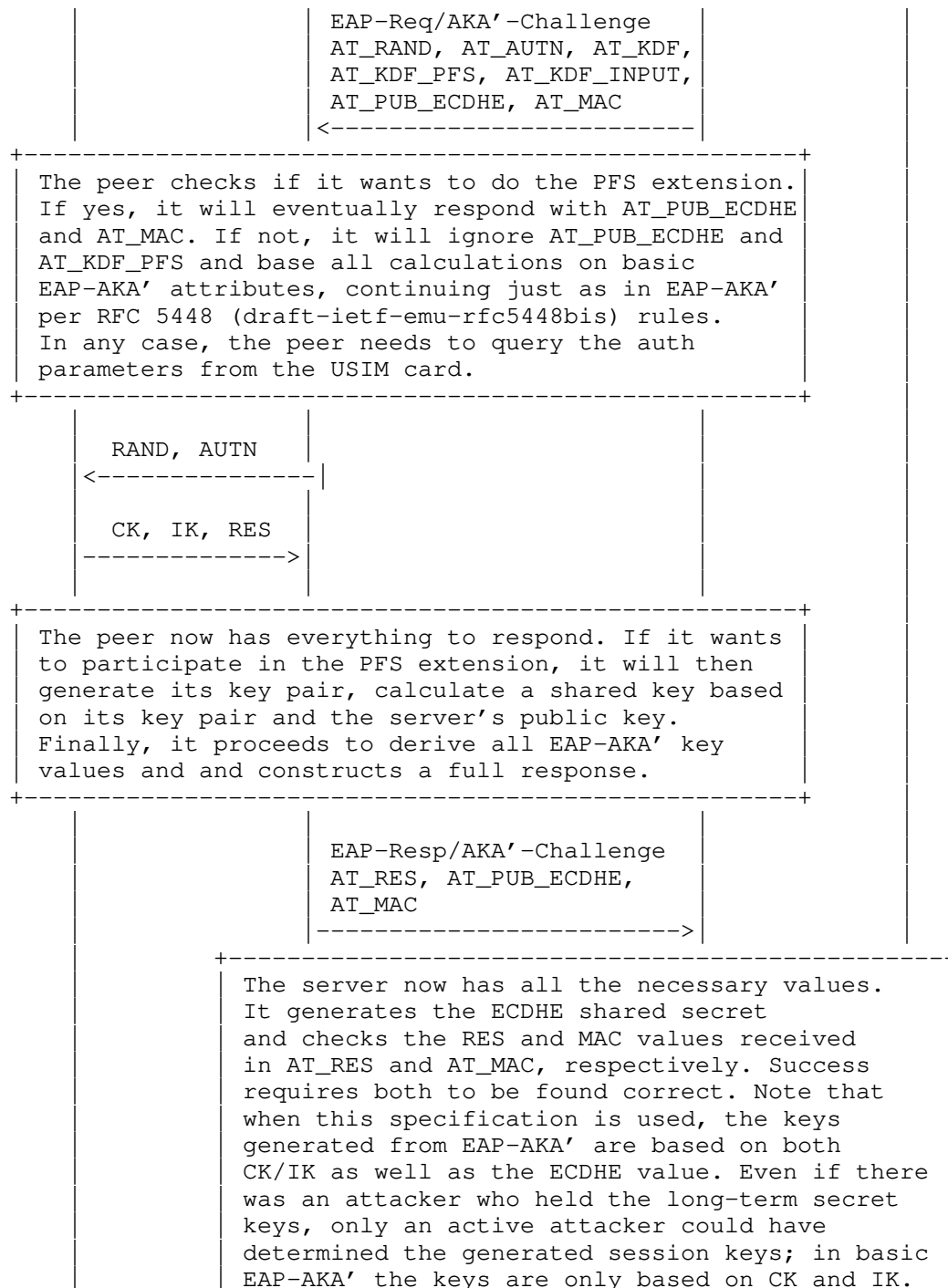
Introducing PFS for EAP-AKA' can be achieved by using an Elliptic Curve Diffie-Hellman (ECDH) exchange [RFC7748]. In EAP-AKA' PFS this exchange is run in an ephemeral manner, i.e., using temporary keys as specified in [RFC8031] Section 2. This method is referred to as ECDHE, where the last 'E' stands for Ephemeral.

The enhancements in the EAP-AKA' PFS protocol are compatible with the signaling flow and other basic structures of both AKA and EAP-AKA'. The intent is to implement the enhancement as optional attributes that legacy implementations can ignore.

The purpose of the protocol is to achieve mutual authentication between the EAP server and peer, and to establish keying material for secure communication between the two. This document specifies the calculation of key material, providing new properties that are not present in key material provided by EAP-AKA' in its original form.

Figure 2 below describes the overall process. Since our goal has been to not require new infrastructure or credentials, the flow diagrams also show the conceptual interaction with the USIM card and the 3GPP authentication server (HSS). The details of those interactions are outside the scope of this document, however, and the reader is referred to the 3GPP specifications .





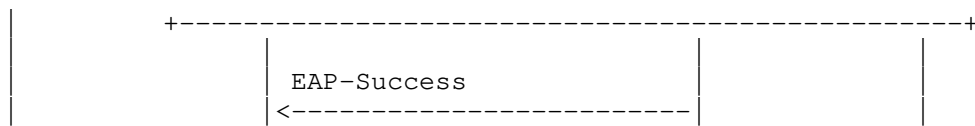


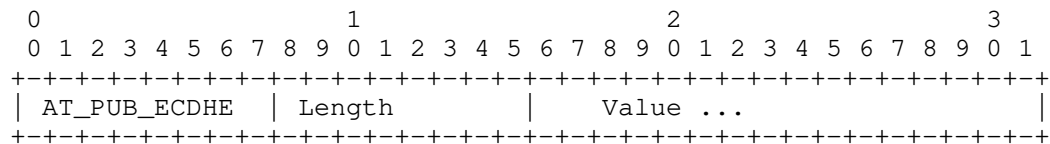
Figure 2: EAP-AKA' PFS Authentication Process

6. Extensions to EAP-AKA'

6.1. AT_PUB_ECDHE

The AT_PUB_ECDHE carries an ECDHE value.

The format of the AT_PUB_ECDHE attribute is shown below.



The fields are as follows:

AT_PUB_ECDHE

This is set to TBA1 BY IANA.

Length

The length of the attribute, set as other attributes in EAP-AKA [RFC4187].

Value

This value is the sender's ECDHE public value. For Curve25519, the length of this value is 32 bytes, encoded in binary as specified [RFC7748] Section 6.1.

To retain the security of the keys, the sender SHALL generate a fresh value for each run of the protocol.

6.2. AT_KDF_PFS

The AT_KDF_PFS indicates the used or desired key generation function, if the Perfect Forward Secrecy extension is taken into use. It will also at the same time indicate the used or desired ECDHE group. A new attribute is needed to carry this information, as AT_KDF carries

the legacy KDF value for those EAP peers that cannot or do not want to use this extension.

The format of the AT_KDF_PFS attribute is shown below.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
AT_KDF_PFS										Length										Key Derivation Function																			

The fields are as follows:

AT_KDF_PFS

This is set to TBA2 BY IANA.

Length

The length of the attribute, MUST be set to 1.

Key Derivation Function

An enumerated value representing the key derivation function that the server (or peer) wishes to use. See Section 6.3 for the functions specified in this document. Note: This field has a different name space than the similar field in the AT_KDF attribute Key Derivation Function defined in [I-D.ietf-emu-rfc5448bis].

Servers MUST send one or more AT_KDF_PFS attributes in the EAP-Request/AKA'-Challenge message. These attributes represent the desired functions ordered by preference, the most preferred function being the first attribute. The most preferred function is the only one that the server includes a public key value for, however. So for a set of AT_KDF_PFS attributes, there is always only one AT_PUB_ECDHE attribute.

Upon receiving a set of these attributes:

- o If the peer supports and is willing to use the key derivation function indicated by the first AT_KDF_PFS attribute, and is willing and able to use the extension defined in this specification, the function is taken into use without any further negotiation.

- o If the peer does not support this function or is unwilling to use it, it responds to the server with an indication that a different function is needed. Similarly with the negotiation process defined in [I-D.ietf-emu-rfc5448bis] for AT_KDF, the peer sends EAP-Response/AKA'-Challenge message that contains only one attribute, AT_KDF_PFS with the value set to the desired alternative function from among the ones suggested by the server earlier. If there is no suitable alternative, the peer has a choice of either falling back to EAP-AKA' or behaving as if AUTN had been incorrect and failing authentication (see Figure 3 of [RFC4187]). The peer MUST fail the authentication if there are any duplicate values within the list of AT_KDF_PFS attributes (except where the duplication is due to a request to change the key derivation function; see below for further information).
- o If the peer does not recognize the extension defined in this specification or is unwilling to use it, it ignores the AT_KDF_PFS attribute.

Upon receiving an EAP-Response/AKA'-Challenge with AT_KDF_PFS from the peer, the server checks that the suggested AT_KDF_PFS value was one of the alternatives in its offer. The first AT_KDF_PFS value in the message from the server is not a valid alternative. If the peer has replied with the first AT_KDF_PFS value, the server behaves as if AT_MAC of the response had been incorrect and fails the authentication. For an overview of the failed authentication process in the server side, see Section 3 and Figure 2 in [RFC4187]. Otherwise, the server re-sends the EAP-Response/AKA'-Challenge message, but adds the selected alternative to the beginning of the list of AT_KDF_PFS attributes, and retains the entire list following it. Note that this means that the selected alternative appears twice in the set of AT_KDF values. Responding to the peer's request to change the key derivation function is the only legal situation where such duplication may occur.

When the peer receives the new EAP-Request/AKA'-Challenge message, it MUST check that the requested change, and only the requested change occurred in the list of AT_KDF_PFS attributes. If yes, it continues. If not, it behaves as if AT_MAC had been incorrect and fails the authentication. If the peer receives multiple EAP-Request/AKA'-Challenge messages with differing AT_KDF_PFS attributes without having requested negotiation, the peer MUST behave as if AT_MAC had been incorrect and fail the authentication.

6.3. New Key Derivation Function

A new Key Derivation Function type is defined for "EAP-AKA' with ECDHE and Curve25519", represented by value 1. It represents a particular choice of key derivation function and at the same time selects an ECDHE group to be used.

The Key Derivation Function type value is only used in the AT_KDF_PFS attribute, and should not be confused with the different range of key derivation functions that can be represented in the AT_KDF attribute as defined in [I-D.ietf-emu-rfc5448bis].

Key derivation in this extension produces exactly the same keys for internal use within one authentication run as [I-D.ietf-emu-rfc5448bis] EAP-AKA' does. For instance, K_aut that is used in AT_MAC is still exactly as it was in EAP-AKA'. The only change to key derivation is in re-authentication keys and keys exported out of the EAP method, MSK and EMSK. As a result, EAP-AKA' attributes such as AT_MAC continue to be usable even when this extension is in use.

When the Key Derivation Function field in the AT_KDF_PFS attribute is set to 1 and the Key Derivation Function field in the AT_KDF attribute is also set to 1, the Master Key (MK) is derived as follows below.

```
MK           = PRF'(IK' | CK', "EAP-AKA'" | Identity)
MK_ECDHE     = PRF'(IK' | CK' | SHARED_SECRET, "EAP-AKA' PFS" | Identity)
K_encr       = MK[0..127]
K_aut        = MK[128..383]
K_re         = MK_ECDHE[0..255]
MSK          = MK_ECDHE[256..767]
EMSK         = MK_ECDHE[768..1279]
```

Where SHARED_SECRET is the shared secret computed via ECDHE, as specified in Section 2 of [RFC8031] and Section 6.1 of [RFC7748].

Both the peer and the server MAY check for zero-value shared secret as specified in Section 6.1 of [RFC7748]. If such checking is performed and the SHARED_SECRET has a zero value, both parties MUST behave as if the current EAP-AKA' authentication process starts again from the beginning.

Note: The way that shared secret is tested for zero can, if performed inappropriately, provide an ability for attackers to listen to CPU power usage side channels. Refer to [RFC7748] for a description of how to perform this check in a way that it does not become a problem.

The rest of computation proceeds as defined in Section 3.3 of [I-D.ietf-emu-rfc5448bis].

For readability, an explanation of the notation used above is copied here: [n..m] denotes the substring from bit n to m. PRF' is a new pseudo-random function specified in [I-D.ietf-emu-rfc5448bis]. K_encr is the encryption key, 128 bits, K_aut is the authentication key, 256 bits, K_re is the re-authentication key, 256 bits, MSK is the Master Session Key, 512 bits, and EMSK is the Extended Master Session Key, 512 bits. MSK and EMSK are outputs from a successful EAP method run [RFC3748].

CK and IK are produced by the AKA algorithm. IK' and CK' are derived as specified in [I-D.ietf-emu-rfc5448bis] from IK and CK.

The value "EAP-AKA'" is an eight-characters-long ASCII string. It is used as is, without any trailing NUL characters. Similarly, "EAP-AKA' PFS" is a twelve-characters-long ASCII string, also used as is.

Identity is the peer identity as specified in Section 7 of [RFC4187].

6.4. ECDHE Groups

The selection of suitable groups for the elliptic curve computation is necessary. The choice of a group is made at the same time as deciding to use of particular key derivation function in AT_KDF_PFS. For "EAP-AKA' with ECDHE and Curve25519" the group is the Curve25519 group specified in [RFC8031].

6.5. Message Processing

This section specifies the changes related to message processing when this extension is used in EAP-AKA'. It specifies when a message may be transmitted or accepted, which attributes are allowed in a message, which attributes are required in a message, and other message-specific details, where those details are different for this extension than the base EAP-AKA' or EAP-AKA protocol. Unless otherwise specified here, the rules from [I-D.ietf-emu-rfc5448bis] or [RFC4187] apply.

6.5.1. EAP-Request/AKA'-Identity

No changes, except that the AT_KDF_PFS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.2. EAP-Response/AKA'-Identity

No changes, except that the AT_KDF_PFS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.3. EAP-Request/AKA'-Challenge

The server sends the EAP-Request/AKA'-Challenge on full authentication as specified by [RFC4187] and [I-D.ietf-emu-rfc5448bis]. The attributes AT_RANDOM, AT_AUTN, and AT_MAC MUST be included and checked on reception as specified in [RFC4187]. They are also necessary for backwards compatibility.

In EAP-Request/AKA'-Challenge, there is no message-specific data covered by the MAC for the AT_MAC attribute. The AT_KDF_PFS and AT_PUB_ECDHE attributes MUST be included. The AT_PUB_ECDHE attribute carries the server's public Diffie-Hellman key. If either AT_KDF_PFS or AT_PUB_ECDHE is missing on reception, the peer MUST treat them as if neither one was sent, and the assume that the extension defined in this specification is not in use.

The AT_RESULT_IND, AT_CHECKCODE, AT_IV, AT_ENCR_DATA, AT_PADDING, AT_NEXT_PSEUDONYM, AT_NEXT_REAUTH_ID and other attributes may be included as specified in Section 9.3 of [RFC4187].

When processing this message, the peer MUST process AT_RANDOM, AT_AUTN, AT_KDF_PFS, AT_PUB_ECDHE before processing other attributes. Only if these attributes are verified to be valid, the peer derives keys and verifies AT_MAC. If the peer is unable or unwilling to perform the extension specified in this document, it proceeds as defined in [I-D.ietf-emu-rfc5448bis]. Finally, the operation in case an error occurs is specified in Section 6.3.1. of [RFC4187].

6.5.4. EAP-Response/AKA'-Challenge

The peer sends EAP-Response/AKA'-Challenge in response to a valid EAP-Request/AKA'-Challenge message, as specified by [RFC4187] and [I-D.ietf-emu-rfc5448bis]. If the peer supports and is willing to perform the extension specified in this protocol, and the server had made a valid request involving the attributes specified in Section 6.5.3, the peer responds per the rules specified below. Otherwise, the peer responds as specified in [RFC4187] and [I-D.ietf-emu-rfc5448bis] and ignores the attributes related to this extension. If the peer has not received attributes related to this extension from the Server, and has a policy that requires it to always use this extension, it behaves as if AUTN had been incorrect and fails the authentication.

The AT_MAC attribute MUST be included and checked as specified in [I-D.ietf-emu-rfc5448bis]. In EAP-Response/AKA'-Challenge, there is no message-specific data covered by the MAC. The AT_PUB_ECDHE attribute MUST be included, and carries the peer's public Diffie-Hellman key.

The AT_RES attribute MUST be included and checked as specified in [RFC4187]. When processing this message, the Server MUST process AT_RES before processing other attributes. Only if these attribute is verified to be valid, the Server derives keys and verifies AT_MAC.

If the Server has proposed the use of the extension specified in this protocol, but the peer ignores and continues the basic EAP-AKA' authentication, the Server makes policy decision of whether this is allowed. If this is allowed, it continues the EAP-AKA' authentication to completion. If it is not allowed, the Server MUST behave as if authentication failed.

The AT_CHECKCODE, AT_RESULT_IND, AT_IV, AT_ENCR_DATA and other attributes may be included as specified in Section 9.4 of [RFC4187].

6.5.5. EAP-Request/AKA'-Reauthentication

No changes, but note that the re-authentication process uses the keys generated in the original EAP-AKA' authentication, which, if the extension specified in this documents is in use, employs key material from the Diffie-Hellman procedure.

6.5.6. EAP-Response/AKA'-Reauthentication

No changes, but as discussed in Section 6.5.5, re-authentication is based on the key material generated by EAP-AKA' and the extension defined in this document.

6.5.7. EAP-Response/AKA'-Synchronization-Failure

No changes, except that the AT_KDF_PFS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.8. EAP-Response/AKA'-Authentication-Reject

No changes, except that the AT_KDF_PFS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.9. EAP-Response/AKA'-Client-Error

No changes, except that the AT_KDF_PFS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.10. EAP-Request/AKA'-Notification

No changes.

6.5.11. EAP-Response/AKA'-Notification

No changes.

7. Security Considerations

This section deals only with the changes to security considerations as they differ from EAP-AKA', or as new information has been gathered since the publication of [I-D.ietf-emu-rfc5448bis].

The possibility of attacks against key storage offered in SIM or other smart cards has been a known threat. But as the discussion in Section 3.3 shows, the likelihood of practically feasible attacks has increased. Many of these attacks can be best dealt with improved processes, e.g., limiting the access to the key material within the factory or personnel, etc. But not all attacks can be entirely ruled out for well-resourced adversaries, irrespective of what the technical algorithms and protection measures are.

This extension can provide assistance in situations where there is a danger of attacks against the key material on SIM cards by adversaries that can not or who are unwilling to mount active attacks against large number of sessions. This extension is most useful when used in a context where EAP keys are used without further mixing that can provide Perfect Forward Secrecy. For instance, when used with IKEv2 [RFC7296], the session keys produced by IKEv2 have this property, so better characteristics of EAP keys is not that useful. However, typical link layer usage of EAP does not involve running Diffie-Hellman, so using EAP to authenticate access to a network is one situation where the extension defined in this document can be helpful.

This extension generates keying material using the ECDHE exchange in order to gain the PFS property. This means that once an EAP-AKA' authentication run ends, the session that it was used to protect is closed, and the corresponding keys are forgotten, even someone who has recorded all of the data from the authentication run and session and gets access to all of the AKA long-term keys cannot reconstruct

the keys used to protect the session or any previous session, without doing a brute force search of the session key space.

Even if a compromise of the long-term keys has occurred, PFS is still provided for all future sessions, as long as the attacker does not become an active attacker. Of course, as with other protocols, if the attacker has learned the keys and does become an active attacker, there is no protection that that can be provided for future sessions. Among other things, such an active attacker can impersonate any legitimate endpoint in EAP-AKA', become a MITM in EAP-AKA' or the extension defined in this document, retrieve all keys, or turn off PFS. Still, past sessions where PFS was in use remain protected.

Achieving PFS requires that when a connection is closed, each endpoint MUST forget not only the ephemeral keys used by the connection but also any information that could be used to recompute those keys.

The following security properties of EAP-AKA' are impacted through this extension:

Protected ciphersuite negotiation

EAP-AKA' has a negotiation mechanism for selecting the key derivation functions, and this mechanism has been extended by the extension specified in this document. The resulting mechanism continues to be secure against bidding down attacks.

There are two specific needs in the negotiation mechanism:

Negotiating key derivation function within the extension

The negotiation mechanism allows changing the offered key derivation function, but the change is visible in the final EAP-Request/KA'-Challenge message that the server sends to the peer. This message is authenticated via the AT_MAC attribute, and carries both the chosen alternative and the initially offered list. The peer refuses to accept a change it did not initiate. As a result, both parties are aware that a change is being made and what the original offer was.

Negotiating the use of this extension

This extension is offered by the server through presenting the AT_KDF_PFS and AT_PUB_ECDHE attributes in the EAP-Request/KA'-Challenge message. These attributes are protected by AT_MAC, so attempts to change or omit them by an adversary will be detected.

Except of course, if the adversary holds the long-term shared secret and is willing to engage in an active attack. Such an attack can, for instance, forge the negotiation process so that no PFS will be provided. However, as noted above, an attacker with these capabilities will in any case be able to impersonate any party in the protocol and perform MITM attacks. That is not a situation that can be improved by a technical solution. However, as discussed in the introduction, even an attacker with access to the long-term keys is required to be a MITM on each AKA run and subsequent communication, which makes mass surveillance more laborous.

The security properties of the extension also depend on a policy choice. As discussed in Section 6.5.4, both the peer and the server make a policy decision of what to do when it was willing to perform the extension specified in this protocol, but the other side does not wish to use the extension. Allowing this has the benefit of allowing backwards compatibility to equipment that did not yet support the extension. When the extension is not supported or negotiated by the parties, no PFS can obviously be provided.

If turning off the extension specified in this protocol is not allowed by policy, the use of legacy equipment that does not support this protocol is no longer possible. This may be appropriate when, for instance, support for the extension is sufficiently widespread, or required in a particular version of a mobile network.

Key derivation

This extension provides key material that is based on the Diffie-Hellman keys, yet bound to the authentication through the (U)SIM card. This means that subsequent payload communications between the parties are protected with keys that are not solely based on information in the clear (such as the RAND) and information derivable from the long-term shared secrets on the (U)SIM card. As a result, if anyone successfully recovers shared secret information, they are unable to decrypt communications protected by the keys generated through this extension. Note that the recovery of shared secret information could occur either before or after the time that the protected communications are used. When this extension is used, communications at time t_0 can be protected if at some later time t_1 an adversary learns of long-term shared secret and has access to a recording of the encrypted communications.

Obviously, this extension is still vulnerable to attackers that are willing to perform an active attack and who at the time of the attack have access to the long-term shared secret.

This extension does not change the properties related to re-authentication. No new Diffie-Hellman run is performed during the re-authentication allowed by EAP-AKA'. However, if this extension was in use when the original EAP-AKA' authentication was performed, the keys used for re-authentication (K_{re}) are based on the Diffie-Hellman keys, and hence continue to be equally safe against exposure of the long-term secrets as the original authentication.

In addition, it is worthwhile to discuss Denial-of-Service attacks and their impact on this protocol. The calculations involved in public key cryptography require computing power, which could be used in an attack to overpower either the peer or the server. While some forms of Denial-of-Service attacks are always possible, the following factors help mitigate the concerns relating to public key cryptography and EAP-AKA' PFS.

- o In 5G context, other parts of the connection setup involve public key cryptography, so while performing additional operations in EAP-AKA' is an additional concern, it does not change the overall situation. As a result, the relevant system components need to be dimensioned appropriately, and detection and management mechanisms to reduce the effect of attacks need to be in place.
- o This specification is constructed so that a separation between the USIM and Peer on client side and the Server and HSS on network side is possible. This ensures that the most sensitive (or legacy) system components can not be the target of the attack. For instance, EAP-AKA' and public key cryptography takes place in the phone and not the low-power SIM card.
- o EAP-AKA' has been designed so that the first actual message in the authentication process comes from the Server, and that this message will not be sent unless the user has been identified as an active subscriber of the operator in question. While the initial identity can be spoofed before authentication has succeeded, this reduces the efficiency of an attack.
- o Finally, this memo specifies an order in which computations and checks must occur. When processing the EAP-Request/AKA'-Challenge message, for instance, the AKA authentication must be checked and succeed before the peer proceeds to calculating or processing the PFS related parameters (see Section 6.5.4). The same is true of EAP-Response/AKA'-Challenge (see Section 6.5.4). This ensures

that the parties need to show possession of the long-term secret in some way, and only then will the PFS calculations become active. This limits the Denial-of-Service to specific, identified subscribers. While botnets and other forms of malicious parties could take advantage of actual subscribers and their key material, at least such attacks are (a) limited in terms of subscribers they control, and (b) identifiable for the purposes of blocking the affected subscribers.

8. IANA Considerations

This extension of EAP-AKA' shares its attribute space and subtypes with EAP-SIM [RFC4186], EAP-AKA [RFC4186], and EAP-AKA' [I-D.ietf-emu-rfc5448bis].

Two new Attribute Type value (TBA1, TBA2) in the skippable range need to be assigned for AT_PUB_ECDHE (Section 6.1) and AT_KDF_PFS (Section 6.2 in the EAP-AKA and EAP-SIM Parameters registry under Attribute Types.

Also, a new registry should be created to represent Diffie-Hellman Key Derivation Function types. The "EAP-AKA' with ECDHE and Curve25519" type (1, see Section 6.3) needs to be assigned, along with one reserved value. The initial contents of this namespace are therefore as below; new values can be created through the Specification Required policy [RFC8126].

Value	Description	Reference
0	Reserved	[TBD BY IANA: THIS RFC]
1	EAP-AKA' with ECDHE and Curve25519	[TBD BY IANA: THIS RFC]
2-65535	Unassigned	

9. References

9.1. Normative References

- [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-05 (work in progress),
 July 2019.
- [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>>.
- [RFC4187] Arkko, J. and H. Haverinen, "Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)", RFC 4187, DOI 10.17487/RFC4187, January 2006, <<https://www.rfc-editor.org/info/rfc4187>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8031] Nir, Y. and S. Josefsson, "Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement", RFC 8031, DOI 10.17487/RFC8031, December 2016, <<https://www.rfc-editor.org/info/rfc8031>>.
- [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>>.

9.2. Informative References

- [DOW1992] Diffie, W., vanOorschot, P., and M. Wiener, "Authentication and Authenticated Key Exchanges", June 1992, in Designs, Codes and Cryptography 2 (2): pp. 107-125.
- [Heist2015] Scahill, J. and J. Begley, "The great SIM heist", February 2015, in <https://firstlook.org/theintercept/2015/02/19/great-sim-heist/> .
- [I-D.mattsson-eap-tls13] Mattsson, J. and M. Sethi, "Using EAP-TLS with TLS 1.3", draft-mattsson-eap-tls13-02 (work in progress), March 2018.

- [RFC4186] Haverinen, H., Ed. and J. Salowey, Ed., "Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)", RFC 4186, DOI 10.17487/RFC4186, January 2006, <<https://www.rfc-editor.org/info/rfc4186>>.
- [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>>.
- [RFC5448] Arkko, J., Lehtovirta, V., and P. Eronen, "Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')", RFC 5448, DOI 10.17487/RFC5448, May 2009, <<https://www.rfc-editor.org/info/rfc5448>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [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>>.
- [TrustCom2015] Arkko, J., Norrman, K., Naslund, M., and B. Sahlin, "A USIM compatible 5G AKA protocol with perfect forward secrecy", August 2015 in Proceedings of the TrustCom 2015, IEEE.

Appendix A. Change Log

The -05 version is merely a refresh while the draft was waiting for WG adoption.

The -04 version of this draft made only editorial changes.

The -03 version of this draft changed the naming of various protocol components, values, and notation to match with the use of ECDH in ephemeral mode. The AT_KDF_PFS negotiation process was clarified in that exactly one key is ever sent in AT_KDF_ECDHE. The option of checking for zero key values IN ECDHE was added. The format of the actual key in AT_PUB_ECDHE was specified. Denial-of-service considerations for the PFS process have been updated. Bidding down attacks against this extension itself are discussed extensively. This version also addressed comments from reviewers, including the

August review from Mohit Sethi, and comments made during IETF-102 discussion.

Appendix B. Acknowledgments

The authors would like to note that the technical solution in this document came out of the TrustCom paper [TrustCom2015], whose authors were J. Arkko, K. Norrman, M. Naslund, and B. Sahlin. This document uses also a lot of material from [RFC4187] by J. Arkko and H. Haverinen as well as [RFC5448] by J. Arkko, V. Lehtovirta, and P. Eronen.

The authors would also like to thank Tero Kivinen, John Mattson, Mohit Sethi, Vesa Lehtovirta, Joseph Salowey, Kathleen Moriarty, Zhang Fu, Bengt Sahlin, Ben Campbell, Prajwol Kumar Nakarmi, Goran Rune, Tim Evans, Helena Vahidi Mazinani, Anand R. Prasad, and many other people at the GSMA and 3GPP groups for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Karl Norrman
Ericsson
Stockholm 16483
Sweden

Email: karl.norrman@ericsson.com

Vesa Torvinen
Ericsson
Jorvas 02420
Finland

Email: vesa.torvinen@ericsson.com

Network Working Group
Internet-Draft
Updates: RFC5448 (if approved)
Intended status: Informational
Expires: 7 September 2022

J. Arkko
K. Norrman
V. Torvinen
Ericsson
March 2022

Forward Secrecy for the Extensible Authentication Protocol Method for
Authentication and Key Agreement (EAP-AKA' FS)
draft-ietf-emu-aka-pfs-06

Abstract

Many different attacks have been reported as part of revelations associated with pervasive surveillance. Some of the reported attacks involved compromising smart cards, such as attacking SIM card manufacturers and operators in an effort to compromise shared secrets stored on these cards. Since the publication of those reports, manufacturing and provisioning processes have gained much scrutiny and have improved. However, the danger of resourceful attackers for these systems is still a concern.

This specification is an optional extension to the EAP-AKA' authentication method which was defined in [RFC9048]. The extension, when negotiated, provides Forward Secrecy for the session key generated as a part of the authentication run in EAP-AKA'. This prevents an attacker who has gained access to the long-term pre-shared secret in a SIM card from being able to decrypt any past communications. In addition, if the attacker stays merely a passive eavesdropper, the extension prevents attacks against future sessions. This forces attackers to use active attacks instead.

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 2 September 2022.

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

Table of Contents

1. Introduction	3
2. Protocol Design and Deployment Objectives	4
3. Background	5
3.1. AKA	5
3.2. EAP-AKA' Protocol	6
3.3. Attacks Against Long-Term Shared Secrets in Smart Cards	7
4. Requirements Language	8
5. Protocol Overview	8
6. Extensions to EAP-AKA'	10
6.1. AT_PUB_ECDHE	10
6.2. AT_KDF_FS	11
6.3. New Key Derivation Functions	13
6.4. ECDHE Groups	15
6.5. Message Processing	15
6.5.1. EAP-Request/AKA'-Identity	15
6.5.2. EAP-Response/AKA'-Identity	15
6.5.3. EAP-Request/AKA'-Challenge	15
6.5.4. EAP-Response/AKA'-Challenge	16
6.5.5. EAP-Request/AKA'-Reauthentication	17
6.5.6. EAP-Response/AKA'-Reauthentication	17
6.5.7. EAP-Response/AKA'-Synchronization-Failure	17
6.5.8. EAP-Response/AKA'-Authentication-Reject	17
6.5.9. EAP-Response/AKA'-Client-Error	17
6.5.10. EAP-Request/AKA'-Notification	17
6.5.11. EAP-Response/AKA'-Notification	17
7. Security Considerations	17
8. IANA Considerations	21
9. References	22
9.1. Normative References	22
9.2. Informative References	23
Appendix A. Change Log	24

Appendix B. Acknowledgments	25
Authors' Addresses	25

1. Introduction

Many different attacks have been reported as part of revelations associated with pervasive surveillance. Some of the reported attacks involved compromising smart cards, such as attacking SIM card manufacturers and operators in an effort to compromise shared secrets stored on these cards. Such attacks are conceivable, for instance, during the manufacturing process of cards, or during the transfer of cards and associated information to the operator. Since the publication of reports about such attacks, manufacturing and provisioning processes have gained much scrutiny and have improved.

However, the danger of resourceful attackers attempting to gain information about SIM cards is still a concern. They are a high-value target and concern a large number of people. Note that the attacks are largely independent of the used authentication technology; the issue is not vulnerabilities in algorithms or protocols, but rather the possibility of someone gaining unlawful access to key material. While the better protection of manufacturing and other processes is essential in protecting against this, there is one question that we as protocol designers can ask. Is there something that we can do to limit the consequences of attacks, should they occur?

The authors want to provide a public specification of an extension that helps defend against one aspect of pervasive surveillance. This is important, given the large number of users such practices may affect. It is also a stated goal of the IETF to ensure that we understand the surveillance concerns related to IETF protocols and take appropriate countermeasures [RFC7258]. This document does that for EAP-AKA'.

This specification is an optional extension to the EAP-AKA' authentication method [RFC9048]. While optional, the use of this extension is RECOMMENDED.

The extension, when negotiated, provides Forward Secrecy for the session key generated as a part of the authentication run in EAP-AKA'. This prevents an attacker who has gained access to the long-term pre-shared secret in a SIM card from being able to decrypt any past communications. In addition, if the attacker stays merely a passive eavesdropper, the extension prevents attacks against future sessions. This forces attackers to use active attacks instead. This is beneficial, because active attacks demand much more resources to launch, and can generally be detected much easier. As with other

protocols, an active attacker with access to the long-term key material will of course be able to attack all future communications, but risks detection, particularly if done at scale. The attacker is forced to attempt to exfiltrate key material, if it can, on a continuous basis, as opposed to learning it once [RFC7624].

Attacks against AKA authentication via compromising the long-term secrets in the SIM cards have been an active discussion topic in many contexts. Forward secrecy is on the list of features for the next release of 3GPP (5G Phase 2), and this document provides a basis for providing this feature in a particular fashion.

It should also be noted that 5G network architecture includes the use of the EAP framework for authentication. While any methods can be run, the default authentication method within that context will be EAP-AKA'. As a result, improvements in EAP-AKA' security have a potential to improve security for large number of users.

2. Protocol Design and Deployment Objectives

This extension specified here re-uses large portions of the current structure of 3GPP interfaces and functions, with the rationale that this will make the construction more easily adopted. In particular, the construction maintains the interface between the Universal Subscriber Identification Module (USIM) and the mobile terminal intact. As a consequence, there is no need to roll out new credentials to existing subscribers. The work is based on an earlier paper [TrustCom2015], and uses much of the same material, but applied to EAP rather than the underlying AKA method.

It has been a goal to implement this change as an extension of the widely supported EAP-AKA' method, rather than a completely new authentication method. The extension is implemented as a set of new, optional attributes, that are provided alongside the base attributes in EAP-AKA'. Old implementations can ignore these attributes, but their presence will nevertheless be verified as part of base EAP-AKA' integrity verification process, helping protect against bidding down attacks. This extension does not increase the number of rounds necessary to complete the protocol.

The use of this extension is at the discretion of the authenticating parties. It should be noted that FS and defenses against passive attacks are by no means a panacea, but they can provide a partial defense that increases the cost and risk associated with pervasive surveillance.

While adding forward secrecy to the existing mobile network infrastructure can be done in multiple different ways, the authors believe that the approach chosen here is relatively easily deployable. In particular:

- * As noted above, no new credentials are needed; there is no change to SIM cards.
- * FS property can be incorporated into any current or future system that supports EAP, without changing any network functions beyond the EAP endpoints.
- * Key generation happens at the endpoints, enabling highest grade key material to be used both by the endpoints and the intermediate systems (such as access points that are given access to specific keys).
- * While EAP-AKA' is just one EAP method, for practical purposes forward secrecy being available for both EAP-TLS [RFC5216] [RFC9190] and EAP-AKA' ensures that for many practical systems forward secrecy can be enabled for either all or significant fraction of users.

3. Background

3.1. AKA

AKA is based on challenge-response mechanisms and symmetric cryptography. AKA typically runs in a UMTS Subscriber Identity Module (USIM) or a CDMA2000 (Removable) User Identity Module ((R)UIM). In contrast with its earlier GSM counterparts, AKA provides long key lengths and mutual authentication.

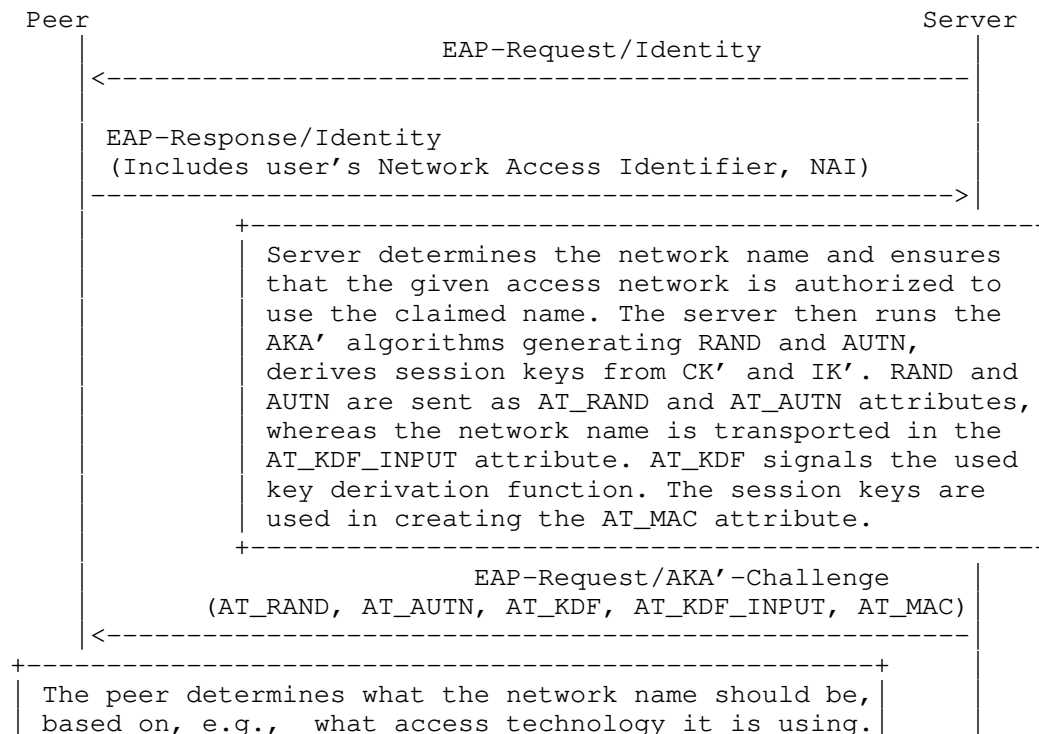
AKA works in the following manner:

- * The identity module and the home environment have agreed on a secret key beforehand.
- * The actual authentication process starts by having the home environment produce an authentication vector, based on the secret key and a sequence number. The authentication vector contains a random part RAND, an authenticator part AUTN used for authenticating the network to the identity module, an expected result part XRES, a 128-bit session key for integrity check IK, and a 128-bit session key for encryption CK.
- * The authentication vector is passed to the serving network, which uses it to authenticate the device.

- * The RAND and the AUTN are delivered to the identity module.
- * The identity module verifies the AUTN, again based on the secret key and the sequence number. If this process is successful (the AUTN is valid and the sequence number used to generate AUTN is within the correct range), the identity module produces an authentication result RES and sends it to the serving network.
- * The serving network verifies the correct result from the identity module. If the result is correct, IK and CK can be used to protect further communications between the identity module and the home environment.

3.2. EAP-AKA' Protocol

When AKA are embedded into EAP, the authentication on the network side is moved to the home environment; the serving network performs the role of a pass-through authenticator. Figure 1 describes the basic flow in the EAP-AKA' authentication process. The definition of the full protocol behaviour, along with the definition of attributes AT RAND, AT AUTN, AT MAC, and AT RES can be found in [RFC9048] and [RFC4187].



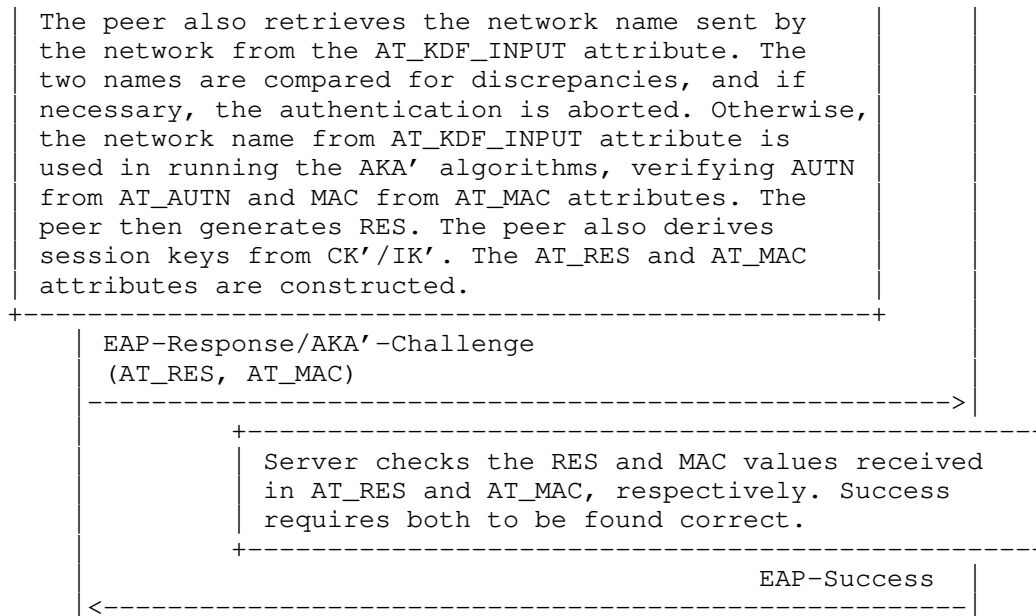


Figure 1: EAP-AKA' Authentication Process

3.3. Attacks Against Long-Term Shared Secrets in Smart Cards

Current 3GPP systems use SIM pre-shared key based protocols and Authentication and Key Agreement (AKA) to authenticate subscribers. The general security properties and potential vulnerabilities of AKA and EAP-AKA' are discussed in [RFC9048].

An important vulnerability in that discussion relates to the recent reports of compromised long term pre-shared keys used in AKA [Heist2015]. These attacks are not specific to AKA or EAP-AKA', as all security systems fail at least to some extent if key material is stolen. However, the reports indicate a need to look into solutions that can operate at least to an extent under these types of attacks. It is noted in [Heist2015] that some security can be retained even in the face of the attacks by providing Forward Secrecy (FS) [DOW1992] for the session key. If AKA would have provided FS, compromising the pre-shared key would not be sufficient to perform passive attacks; the attacker is, in addition, forced to be a Man-In-The-Middle (MITM) during the AKA run and subsequent communication between the parties.

4. 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.

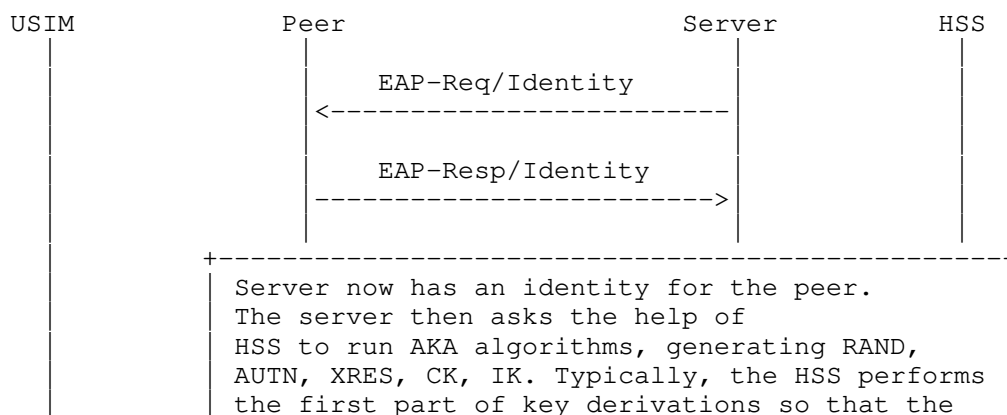
5. Protocol Overview

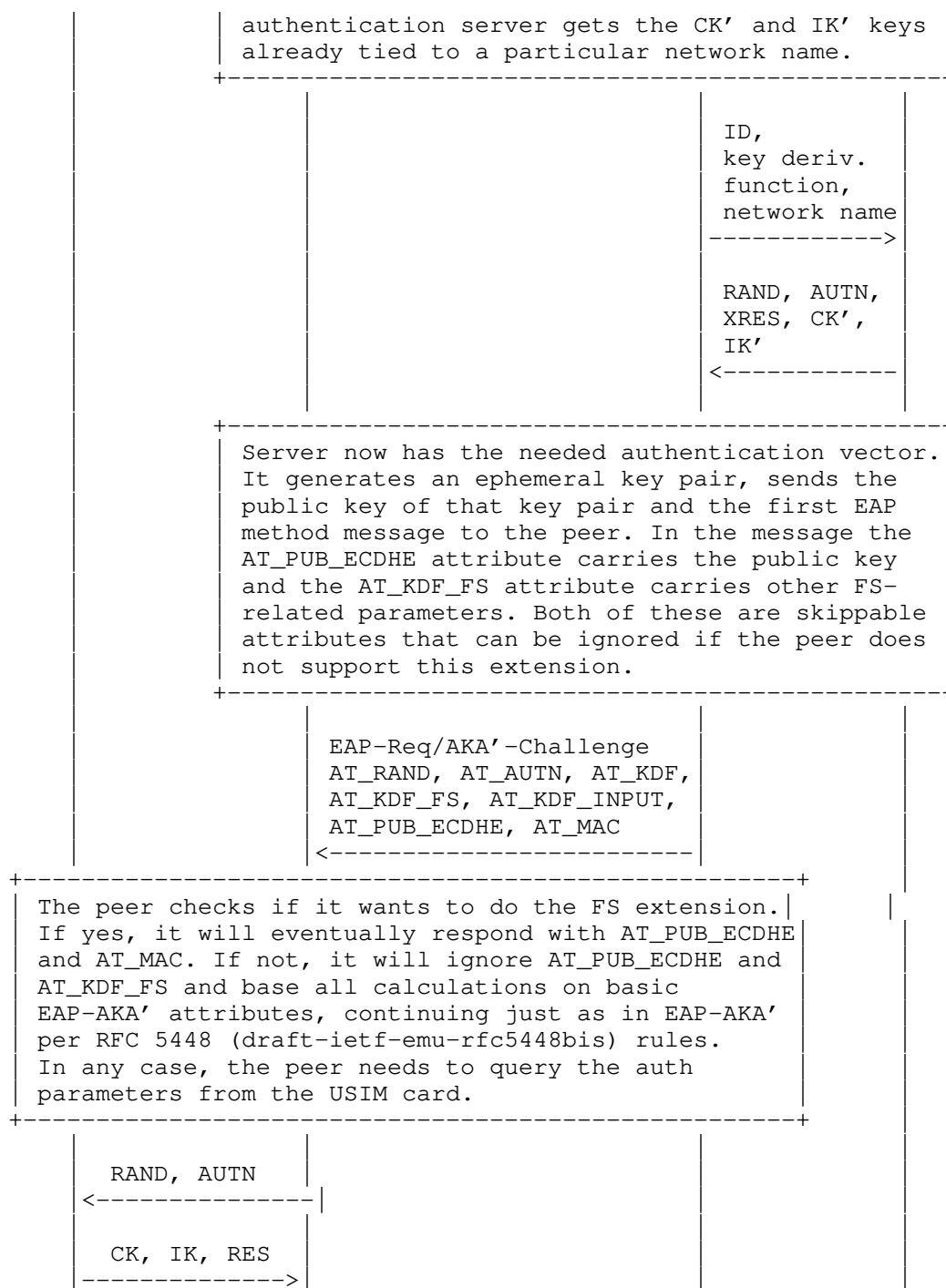
Introducing FS for EAP-AKA' can be achieved by using an Elliptic Curve Diffie-Hellman (ECDH) exchange [RFC7748]. In EAP-AKA' FS this exchange is run in an ephemeral manner, i.e., both sides generate temporary keys as specified in [RFC7748]. This method is referred to as ECDHE, where the last 'E' stands for Ephemeral.

The enhancements in the EAP-AKA' FS protocol are compatible with the signaling flow and other basic structures of both AKA and EAP-AKA'. The intent is to implement the enhancement as optional attributes that legacy implementations can ignore.

The purpose of the protocol is to achieve mutual authentication between the EAP server and peer, and to establish keying material for secure communication between the two. This document specifies the calculation of key material, providing new properties that are not present in key material provided by EAP-AKA' in its original form.

Figure 2 below describes the overall process. Since our goal has been to not require new infrastructure or credentials, the flow diagrams also show the conceptual interaction with the USIM card and the 3GPP authentication server (HSS). The details of those interactions are outside the scope of this document, however, and the reader is referred to the 3GPP specifications .





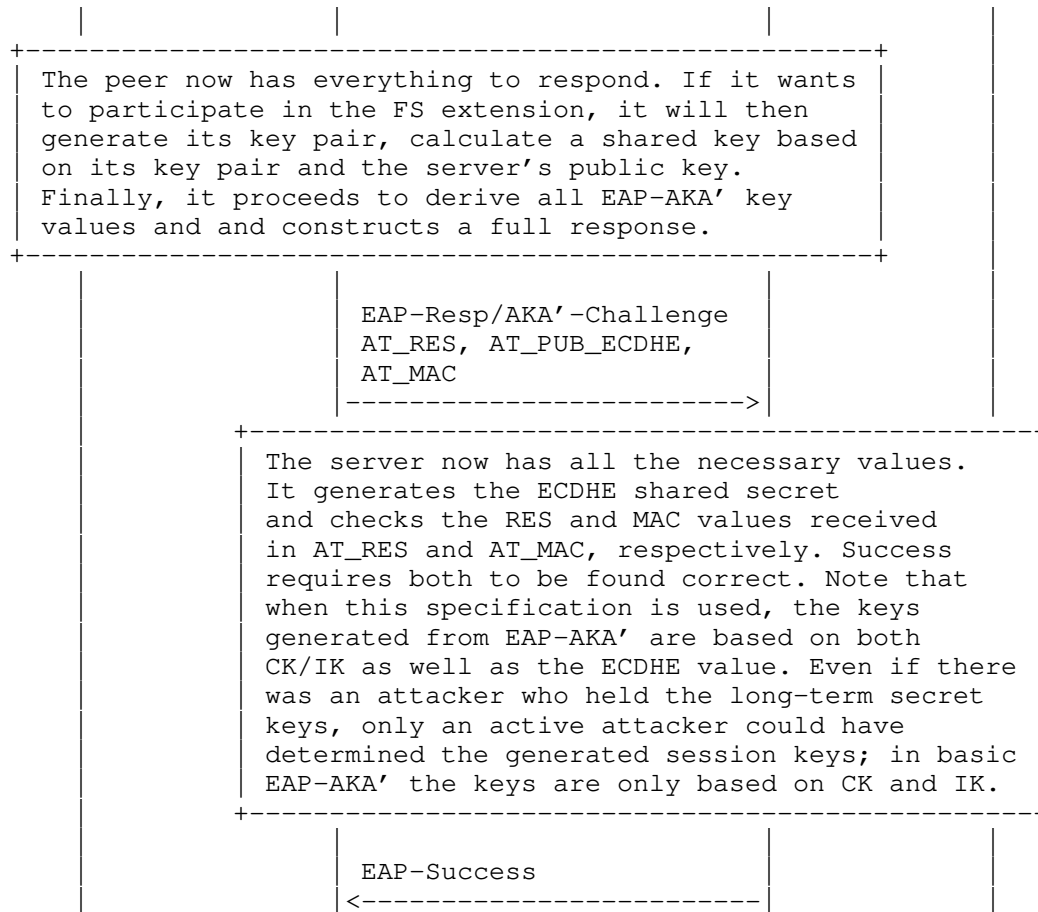


Figure 2: EAP-AKA' FS Authentication Process

6. Extensions to EAP-AKA'

6.1. AT_PUB_ECDHE

The AT_PUB_ECDHE carries an ECDHE value.

The format of the AT_PUB_ECDHE attribute is shown below.

```

      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
+-----+-----+-----+-----+-----+-----+-----+-----+
| AT_PUB_ECDHE | Length | Value ... |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The fields are as follows:

AT_PUB_ECDHE

This is set to TBA1 BY IANA.

Length

The length of the attribute, set as other attributes in EAP-AKA [RFC4187].

Value

This value is the sender's ECDHE public value. It is calculated as follows:

- * For X25519/Curve25519, the length of this value is 32 bytes, encoded in binary as specified [RFC7748] Section 6.1.
- * For P-256, the length of this value is 33 bytes, encoded in binary as specified in [FIPS186-4], using the compressed form from Section 2.7.1 of [SEC2].

To retain the security of the keys, the sender SHALL generate a fresh value for each run of the protocol.

6.2. AT_KDF_FS

The AT_KDF_FS indicates the used or desired key generation function, if the Forward Secrecy extension is taken into use. It will also at the same time indicate the used or desired ECDHE group. A new attribute is needed to carry this information, as AT_KDF carries the legacy KDF value for those EAP peers that cannot or do not want to use this extension.

The format of the AT_KDF_FS attribute is shown below.

```

      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
+-----+-----+-----+-----+-----+-----+-----+-----+
| AT_KDF_FS | Length | Key Derivation Function |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The fields are as follows:

AT_KDF_FS

This is set to TBA2 BY IANA.

Length

The length of the attribute, MUST be set to 1.

Key Derivation Function

An enumerated value representing the key derivation function that the server (or peer) wishes to use. See Section 6.3 for the functions specified in this document. Note: This field has a different name space than the similar field in the AT_KDF attribute Key Derivation Function defined in [RFC9048].

Servers MUST send one or more AT_KDF_FS attributes in the EAP-Request/AKA'-Challenge message. These attributes represent the desired functions ordered by preference, the most preferred function being the first attribute. The most preferred function is the only one that the server includes a public key value for, however. So for a set of AT_KDF_FS attributes, there is always only one AT_PUB_ECDHE attribute.

Upon receiving a set of these attributes:

- * If the peer supports and is willing to use the key derivation function indicated by the first AT_KDF_FS attribute, and is willing and able to use the extension defined in this specification, the function is taken into use without any further negotiation.
- * If the peer does not support this function or is unwilling to use it, it responds to the server with an indication that a different function is needed. Similarly with the negotiation process defined in [RFC9048] for AT_KDF, the peer sends EAP-Response/AKA'-Challenge message that contains only one attribute, AT_KDF_FS with the value set to the desired alternative function from among the ones suggested by the server earlier. If there is no suitable alternative, the peer has a choice of either falling back to EAP-AKA' or behaving as if AUTN had been incorrect and failing authentication (see Figure 3 of [RFC4187]). The peer MUST fail the authentication if there are any duplicate values within the list of AT_KDF_FS attributes (except where the duplication is due to a request to change the key derivation function; see below for further information).

- * If the peer does not recognize the extension defined in this specification or is unwilling to use it, it ignores the AT_KDF_FS attribute.

Upon receiving an EAP-Response/AKA'-Challenge with AT_KDF_FS from the peer, the server checks that the suggested AT_KDF_FS value was one of the alternatives in its offer. The first AT_KDF_FS value in the message from the server is not a valid alternative. If the peer has replied with the first AT_KDF_FS value, the server behaves as if AT_MAC of the response had been incorrect and fails the authentication. For an overview of the failed authentication process in the server side, see Section 3 and Figure 2 in [RFC4187]. Otherwise, the server re-sends the EAP-Response/AKA'-Challenge message, but adds the selected alternative to the beginning of the list of AT_KDF_FS attributes, and retains the entire list following it. Note that this means that the selected alternative appears twice in the set of AT_KDF values. Responding to the peer's request to change the key derivation function is the only legal situation where such duplication may occur.

When the peer receives the new EAP-Request/AKA'-Challenge message, it MUST check that the requested change, and only the requested change occurred in the list of AT_KDF_FS attributes. If yes, it continues. If not, it behaves as if AT_MAC had been incorrect and fails the authentication. If the peer receives multiple EAP-Request/AKA'-Challenge messages with differing AT_KDF_FS attributes without having requested negotiation, the peer MUST behave as if AT_MAC had been incorrect and fail the authentication.

6.3. New Key Derivation Functions

Two new Key Derivation Function types are defined for "EAP-AKA' with ECDHE and X25519", represented by value 1, and "EAP-AKA' with ECDHE and P-256", represented by value 2. These represent a particular choice of key derivation function and at the same time selects an ECDHE group to be used. The Key Derivation Function type value is only used in the AT_KDF_FS attribute, and should not be confused with the different range of key derivation functions that can be represented in the AT_KDF attribute as defined in [RFC9048].

Key derivation in this extension produces exactly the same keys for internal use within one authentication run as [RFC9048] EAP-AKA' does. For instance, K_aut that is used in AT_MAC is still exactly as it was in EAP-AKA'. The only change to key derivation is in re-authentication keys and keys exported out of the EAP method, MSK and EMSK. As a result, EAP-AKA' attributes such as AT_MAC continue to be usable even when this extension is in use.

When the Key Derivation Function field in the AT_KDF_FS attribute is set to 1 and the Key Derivation Function field in the AT_KDF attribute is also set to 1, the Master Key (MK) is derived as follows below.

```
MK          = PRF'(IK'|CK',"EAP-AKA'"|Identity)
MK_ECDHE    = PRF'(IK'|CK'|SHARED_SECRET,"EAP-AKA' FS"|Identity)
K_encr      = MK[0..127]
K_aut       = MK[128..383]
K_re        = MK_ECDHE[0..255]
MSK         = MK_ECDHE[256..767]
EMSK        = MK_ECDHE[768..1279]
```

Where SHARED_SECRET is the shared secret computed via ECDHE, as specified in Section 6.1 of Both the peer and the server MAY check for zero-value shared secret as specified in Section 6.1 of The rest of computation proceeds as defined in Section 3.3 of [RFC7748]. [RFC7748]. If such checking is performed and the SHARED_SECRET has a zero value, both parties MUST behave as if the current EAP-AKA' authentication process starts again from the beginning.

Note: The way that shared secret is tested for zero can, if performed inappropriately, provide an ability for attackers to listen to CPU power usage side channels. Refer to [RFC7748] for a description of how to perform this check in a way that it does not become a problem.

[RFC9048].

For readability, an explanation of the notation used above is copied here: [n..m] denotes the substring from bit n to m. PRF' is a new pseudo-random function specified in [RFC9048]. K_encr is the encryption key, 128 bits, K_aut is the authentication key, 256 bits, K_re is the re-authentication key, 256 bits, MSK is the Master Session Key, 512 bits, and EMSK is the Extended Master Session Key, 512 bits. MSK and EMSK are outputs from a successful EAP method run [RFC3748].

CK and IK are produced by the AKA algorithm. IK' and CK' are derived as specified in [RFC9048] from IK and CK.

The value "EAP-AKA'" is an eight-characters-long ASCII string. It is used as is, without any trailing NUL characters. Similarly, "EAP-AKA' FS" is an eleven-characters-long ASCII string, also used as is.

Identity is the peer identity as specified in Section 7 of [RFC4187].

6.4. ECDHE Groups

The selection of suitable groups for the elliptic curve computation is necessary. The choice of a group is made at the same time as deciding to use of particular key derivation function in AT_KDF_FS.

For "EAP-AKA' with ECDHE and X25519" the group is the Curve25519 group specified in [RFC7748]. The support for this group is REQUIRED.

For "EAP-AKA' with ECDHE and P-256" the group is the NIST P-256 group (SEC group secp256r1), specified in [FIPS186-4]. The support for this group is OPTIONAL.

6.5. Message Processing

This section specifies the changes related to message processing when this extension is used in EAP-AKA'. It specifies when a message may be transmitted or accepted, which attributes are allowed in a message, which attributes are required in a message, and other message-specific details, where those details are different for this extension than the base EAP-AKA' or EAP-AKA protocol. Unless otherwise specified here, the rules from [RFC9048] or [RFC4187] apply.

6.5.1. EAP-Request/AKA'-Identity

No changes, except that the AT_KDF_FS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.2. EAP-Response/AKA'-Identity

No changes, except that the AT_KDF_FS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.3. EAP-Request/AKA'-Challenge

The server sends the EAP-Request/AKA'-Challenge on full authentication as specified by [RFC4187] and [RFC9048]. The attributes AT_RANDOM, AT_AUTN, and AT_MAC MUST be included and checked on reception as specified in [RFC4187]. They are also necessary for backwards compatibility.

In EAP-Request/AKA'-Challenge, there is no message-specific data covered by the MAC for the AT_MAC attribute. The AT_KDF_FS and AT_PUB_ECDHE attributes MUST be included. The AT_PUB_ECDHE attribute

carries the server's public Diffie-Hellman key. If either AT_KDF_FS or AT_PUB_ECDHE is missing on reception, the peer MUST treat them as if neither one was sent, and the assume that the extension defined in this specification is not in use.

The AT_RESULT_IND, AT_CHECKCODE, AT_IV, AT_ENCR_DATA, AT_PADDING, AT_NEXT_PSEUDONYM, AT_NEXT_REAUTH_ID and other attributes may be included as specified in Section 9.3 of [RFC4187].

When processing this message, the peer MUST process AT_RAND, AT_AUTN, AT_KDF_FS, AT_PUB_ECDHE before processing other attributes. Only if these attributes are verified to be valid, the peer derives keys and verifies AT_MAC. If the peer is unable or unwilling to perform the extension specified in this document, it proceeds as defined in [RFC9048]. Finally, the operation in case an error occurs is specified in Section 6.3.1. of [RFC4187].

6.5.4. EAP-Response/AKA'-Challenge

The peer sends EAP-Response/AKA'-Challenge in response to a valid EAP-Request/AKA'-Challenge message, as specified by [RFC4187] and [RFC9048]. If the peer supports and is willing to perform the extension specified in this protocol, and the server had made a valid request involving the attributes specified in Section 6.5.3, the peer responds per the rules specified below. Otherwise, the peer responds as specified in [RFC4187] and [RFC9048] and ignores the attributes related to this extension. If the peer has not received attributes related to this extension from the Server, and has a policy that requires it to always use this extension, it behaves as if AUTN had been incorrect and fails the authentication.

The AT_MAC attribute MUST be included and checked as specified in [RFC9048]. In EAP-Response/AKA'-Challenge, there is no message-specific data covered by the MAC. The AT_PUB_ECDHE attribute MUST be included, and carries the peer's public Diffie-Hellman key.

The AT_RES attribute MUST be included and checked as specified in [RFC4187]. When processing this message, the Server MUST process AT_RES before processing other attributes. Only if these attribute is verified to be valid, the Server derives keys and verifies AT_MAC.

If the Server has proposed the use of the extension specified in this protocol, but the peer ignores and continues the basic EAP-AKA' authentication, the Server makes policy decision of whether this is allowed. If this is allowed, it continues the EAP-AKA' authentication to completion. If it is not allowed, the Server MUST behave as if authentication failed.

The AT_CHECKCODE, AT_RESULT_IND, AT_IV, AT_ENCR_DATA and other attributes may be included as specified in Section 9.4 of [RFC4187].

6.5.5. EAP-Request/AKA'-Reauthentication

No changes, but note that the re-authentication process uses the keys generated in the original EAP-AKA' authentication, which, if the extension specified in this documents is in use, employs key material from the Diffie-Hellman procedure.

6.5.6. EAP-Response/AKA'-Reauthentication

No changes, but as discussed in Section 6.5.5, re-authentication is based on the key material generated by EAP-AKA' and the extension defined in this document.

6.5.7. EAP-Response/AKA'-Synchronization-Failure

No changes, except that the AT_KDF_FS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.8. EAP-Response/AKA'-Authentication-Reject

No changes, except that the AT_KDF_FS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.9. EAP-Response/AKA'-Client-Error

No changes, except that the AT_KDF_FS or AT_PUB_ECDHE attributes MUST NOT be added to this message. The appearance of these messages in a received message MUST be ignored.

6.5.10. EAP-Request/AKA'-Notification

No changes.

6.5.11. EAP-Response/AKA'-Notification

No changes.

7. Security Considerations

This section deals only with the changes to security considerations as they differ from EAP-AKA', or as new information has been gathered since the publication of [RFC9048].

The possibility of attacks against key storage offered in SIM or other smart cards has been a known threat. But as the discussion in Section 3.3 shows, the likelihood of practically feasible attacks has increased. Many of these attacks can be best dealt with improved processes, e.g., limiting the access to the key material within the factory or personnel, etc. But not all attacks can be entirely ruled out for well-resourced adversaries, irrespective of what the technical algorithms and protection measures are.

This extension can provide assistance in situations where there is a danger of attacks against the key material on SIM cards by adversaries that can not or who are unwilling to mount active attacks against large number of sessions. This extension is most useful when used in a context where EAP keys are used without further mixing that can provide Forward Secrecy. For instance, when used with IKEv2 [RFC7296], the session keys produced by IKEv2 have this property, so better characteristics of EAP keys is not that useful. However, typical link layer usage of EAP does not involve running Diffie-Hellman, so using EAP to authenticate access to a network is one situation where the extension defined in this document can be helpful.

This extension generates keying material using the ECDHE exchange in order to gain the FS property. This means that once an EAP-AKA' authentication run ends, the session that it was used to protect is closed, and the corresponding keys are forgotten, even someone who has recorded all of the data from the authentication run and session and gets access to all of the AKA long-term keys cannot reconstruct the keys used to protect the session or any previous session, without doing a brute force search of the session key space.

Even if a compromise of the long-term keys has occurred, FS is still provided for all future sessions, as long as the attacker does not become an active attacker. Of course, as with other protocols, if the attacker has learned the keys and does become an active attacker, there is no protection that that can be provided for future sessions. Among other things, such an active attacker can impersonate any legitimate endpoint in EAP-AKA', become a MITM in EAP-AKA' or the extension defined in this document, retrieve all keys, or turn off FS. Still, past sessions where FS was in use remain protected.

Achieving FS requires that when a connection is closed, each endpoint MUST forget not only the ephemeral keys used by the connection but also any information that could be used to recompute those keys.

The following security properties of EAP-AKA' are impacted through this extension:

Protected ciphersuite negotiation

EAP-AKA' has a negotiation mechanism for selecting the key derivation functions, and this mechanism has been extended by the extension specified in this document. The resulting mechanism continues to be secure against bidding down attacks.

There are two specific needs in the negotiation mechanism:

Negotiating key derivation function within the extension

The negotiation mechanism allows changing the offered key derivation function, but the change is visible in the final EAP-Request/AKA'-Challenge message that the server sends to the peer. This message is authenticated via the AT_MAC attribute, and carries both the chosen alternative and the initially offered list. The peer refuses to accept a change it did not initiate. As a result, both parties are aware that a change is being made and what the original offer was.

Negotiating the use of this extension

This extension is offered by the server through presenting the AT_KDF_FS and AT_PUB_ECDHE attributes in the EAP-Request/AKA'-Challenge message. These attributes are protected by AT_MAC, so attempts to change or omit them by an adversary will be detected.

Except of course, if the adversary holds the long-term shared secret and is willing to engage in an active attack. Such an attack can, for instance, forge the negotiation process so that no FS will be provided. However, as noted above, an attacker with these capabilities will in any case be able to impersonate any party in the protocol and perform MITM attacks. That is not a situation that can be improved by a technical solution. However, as discussed in the introduction, even an attacker with access to the long-term keys is required to be a MITM on each AKA run and subsequent communication, which makes mass surveillance more laborious.

The security properties of the extension also depend on a policy choice. As discussed in Section 6.5.4, both the peer and the server make a policy decision of what to do when it was willing to perform the extension specified in this protocol, but the other side does not wish to use the extension. Allowing this has the benefit of allowing backwards compatibility to equipment that did not yet support the extension. When the extension is not supported or negotiated by the parties, no FS can obviously be provided.

If turning off the extension specified in this protocol is not allowed by policy, the use of legacy equipment that does not support this protocol is no longer possible. This may be appropriate when, for instance, support for the extension is sufficiently widespread, or required in a particular version of a mobile network.

Key derivation

This extension provides key material that is based on the Diffie-Hellman keys, yet bound to the authentication through the SIM card. This means that subsequent payload communications between the parties are protected with keys that are not solely based on information in the clear (such as the RAND) and information derivable from the long-term shared secrets on the SIM card. As a result, if anyone successfully recovers shared secret information, they are unable to decrypt communications protected by the keys generated through this extension. Note that the recovery of shared secret information could occur either before or after the time that the protected communications are used. When this extension is used, communications at time t_0 can be protected if at some later time t_1 an adversary learns of long-term shared secret and has access to a recording of the encrypted communications.

Obviously, this extension is still vulnerable to attackers that are willing to perform an active attack and who at the time of the attack have access to the long-term shared secret.

This extension does not change the properties related to re-authentication. No new Diffie-Hellman run is performed during the re-authentication allowed by EAP-AKA'. However, if this extension was in use when the original EAP-AKA' authentication was performed, the keys used for re-authentication (K_{re}) are based on the Diffie-Hellman keys, and hence continue to be equally safe against expose of the long-term secrets as the original authentication.

In addition, it is worthwhile to discuss Denial-of-Service attacks and their impact on this protocol. The calculations involved in public key cryptography require computing power, which could be used in an attack to overpower either the peer or the server. While some forms of Denial-of-Service attacks are always possible, the following factors help mitigate the concerns relating to public key cryptography and EAP-AKA' FS.

- * In 5G context, other parts of the connection setup involve public key cryptography, so while performing additional operations in EAP-AKA' is an additional concern, it does not change the overall

situation. As a result, the relevant system components need to be dimensioned appropriately, and detection and management mechanisms to reduce the effect of attacks need to be in place.

- * This specification is constructed so that a separation between the USIM and Peer on client side and the Server and HSS on network side is possible. This ensures that the most sensitive (or legacy) system components can not be the target of the attack. For instance, EAP-AKA' and public key cryptography takes place in the phone and not the low-power SIM card.
- * EAP-AKA' has been designed so that the first actual message in the authentication process comes from the Server, and that this message will not be sent unless the user has been identified as an active subscriber of the operator in question. While the initial identity can be spoofed before authentication has succeeded, this reduces the efficiency of an attack.
- * Finally, this memo specifies an order in which computations and checks must occur. When processing the EAP-Request/AKA'-Challenge message, for instance, the AKA authentication must be checked and succeed before the peer proceeds to calculating or processing the FS related parameters (see Section 6.5.4). The same is true of EAP-Response/AKA'-Challenge (see Section 6.5.4). This ensures that the parties need to show possession of the long-term secret in some way, and only then will the FS calculations become active. This limits the Denial-of-Service to specific, identified subscribers. While botnets and other forms of malicious parties could take advantage of actual subscribers and their key material, at least such attacks are (a) limited in terms of subscribers they control, and (b) identifiable for the purposes of blocking the affected subscribers.

8. IANA Considerations

This extension of EAP-AKA' shares its attribute space and subtypes with EAP-SIM [RFC4186], EAP-AKA [RFC4186], and EAP-AKA' [RFC9048].

Two new Attribute Type value (TBA1, TBA2) in the skippable range need to be assigned for AT_PUB_ECDHE (Section 6.1) and AT_KDF_FS (Section 6.2 in the EAP-AKA and EAP-SIM Parameters registry under Attribute Types.

Also, a new registry should be created to represent Diffie-Hellman Key Derivation Function types. The "EAP-AKA' with ECDHE and X25519" and "EAP-AKA' with ECDHE and P-256" types (1 and 2, see Section 6.3) need to be assigned, along with one reserved value. The initial contents of this namespace are therefore as below; new values can be created through the Specification Required policy [RFC8126].

Value	Description	Reference
0	Reserved	[TBD BY IANA: THIS RFC]
1	EAP-AKA' with ECDHE and X25519	[TBD BY IANA: THIS RFC]
2	EAP-AKA' with ECDHE and P-256	[TBD BY IANA: THIS RFC]
3-65535	Unassigned	[TBD BY IANA: THIS RFC]

9. References

9.1. 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>>.
- [RFC4187] Arkko, J. and H. Haverinen, "Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)", RFC 4187, DOI 10.17487/RFC4187, January 2006, <<https://www.rfc-editor.org/info/rfc4187>>.
- [RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <<https://www.rfc-editor.org/info/rfc7624>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [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>>.
- [RFC9048] Arkko, J., Lehtovirta, V., Torvinen, V., and P. Eronen, "Improved Extensible Authentication Protocol Method for 3GPP Mobile Network Authentication and Key Agreement (EAP-AKA')", RFC 9048, DOI 10.17487/RFC9048, October 2021, <<https://www.rfc-editor.org/info/rfc9048>>.
- [FIPS186-4] "Digital Signature Standard (DSS)", July 2013.
- [SEC2] "SEC 2: Recommended Elliptic Curve Domain Parameters", September 2000.

9.2. Informative References

- [RFC4186] Haverinen, H., Ed. and J. Salowey, Ed., "Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM)", RFC 4186, DOI 10.17487/RFC4186, January 2006, <<https://www.rfc-editor.org/info/rfc4186>>.
- [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>>.
- [RFC5448] Arkko, J., Lehtovirta, V., and P. Eronen, "Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')", RFC 5448, DOI 10.17487/RFC5448, May 2009, <<https://www.rfc-editor.org/info/rfc5448>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [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>>.
- [RFC9190] Preuß Mattsson, J. and M. Sethi, "EAP-TLS 1.3: Using the Extensible Authentication Protocol with TLS 1.3", RFC 9190, DOI 10.17487/RFC9190, February 2022, <<https://www.rfc-editor.org/info/rfc9190>>.

[TrustCom2015]

Arkko, J., Norrman, K., Naslund, M., and B. Sahlin, "A USIM compatible 5G AKA protocol with perfect forward secrecy", 2015 in Proceedings of the TrustCom 2015, IEEE (August None),
<<http://ieeexplore.ieee.org/document/7345414/>>.

[Heist2015]

Scahill, J. and J. Begley, "The great SIM heist", 2015, in <https://firstlook.org/theintercept/2015/02/19/great-sim-heist/> (February None).

[DOW1992]

Diffie, W., vanOorschot, P., and M. Wiener, "Authentication and Authenticated Key Exchanges", 1992, in Designs, Codes and Cryptography 2 (2): pp. 107-125 (June None).

Appendix A. Change Log

The -06 version of the WG draft is a refresh and a reference update. However, the following should be noted:

- * The draft now uses "forward secrecy" terminology and references RFC 7624 per recommendations on mailing list discussion.
- * There's been mailing list discussion about the encoding of the public values; the current text requires confirmation from the working group that it is sufficient.

The -05 version of the WG draft takes into account feedback from the working group list, about the number of bytes needed to encode P-256 values.

The -04 version of the WG draft takes into account feedback from the May 2020 WG interim meeting, correcting the reference to the NIST P-256 specification.

The -03 version of the WG draft is first of all a refresh; there are no issues that we think need addressing, beyond the one for which there is a suggestion in -03: The specification now suggests an alternate group/curve as an optional one besides X25519. The specific choice of particular groups and algorithms is still up to the working group.

The -02 version of the WG draft took into account additional reviews, and changed the document to update RFC 5448 (or rather, its successor, [RFC9048]), changed the wording of the recommendation with regards to the use of this extension, clarified the references to the

definition of X25519 and Curve25519, clarified the distinction to ECDH methods that use partially static keys, and simplified the use of AKA and SIM card terminology. Some editorial changes were also made.

The -00 and -01 versions of the WG draft made no major changes, only updates to some references.

The -05 version is merely a refresh while the draft was waiting for WG adoption.

The -04 version of this draft made only editorial changes.

The -03 version of this draft changed the naming of various protocol components, values, and notation to match with the use of ECDH in ephemeral mode. The AT_KDF_FS negotiation process was clarified in that exactly one key is ever sent in AT_KDF_ECDHE. The option of checking for zero key values IN ECDHE was added. The format of the actual key in AT_PUB_ECDHE was specified. Denial-of-service considerations for the FS process have been updated. Bidding down attacks against this extension itself are discussed extensively. This version also addressed comments from reviewers, including the August review from Mohit Sethi, and comments made during IETF-102 discussion.

Appendix B. Acknowledgments

The authors would like to note that the technical solution in this document came out of the TrustCom paper [TrustCom2015], whose authors were J. Arkko, K. Norrman, M. Naslund, and B. Sahlin. This document uses also a lot of material from [RFC4187] by J. Arkko and H. Haverinen as well as [RFC5448] by J. Arkko, V. Lehtovirta, and P. Eronen.

The authors would also like to thank Tero Kivinen, John Mattsson, Mohit Sethi, Vesa Lehtovirta, Russ Housley, Sean Turner, Eliot Lear, Joseph Salowey, Kathleen Moriarty, Zhang Fu, Bengt Sahlin, Ben Campbell, Prajwol Kumar Nakarmi, Goran Rune, Tim Evans, Helena Vahidi Mazinani, Anand R. Prasad, Rene Struik, and many other people at the IETF, GSMA and 3GPP groups for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko
Ericsson
FI-02420 Jorvas
Finland

Email: jari.arkko@piuha.net

Karl Norrman
Ericsson
SE-16483 Stockholm
Sweden
Email: karl.norrman@ericsson.com

Vesa Torvinen
Ericsson
FI-02420 Jorvas
Finland
Email: vesa.torvinen@ericsson.com

Network Working Group
Internet-Draft
Updates: 5216 (if approved)
Intended status: Standards Track
Expires: March 23, 2020

J. Mattsson
M. Sethi
Ericsson
September 20, 2019

Using EAP-TLS with TLS 1.3
draft-ietf-emu-eap-tls13-07

Abstract

This document specifies the use of EAP-TLS with TLS 1.3 while remaining backwards compatible with existing implementations of EAP-TLS. TLS 1.3 provides significantly improved security, privacy, and reduced latency when compared to earlier versions of TLS. EAP-TLS with TLS 1.3 further improves security and privacy by mandating use of privacy and revocation checking. 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 March 23, 2020.

Copyright Notice

Copyright (c) 2019 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	2
1.1. Requirements and Terminology	3
2. Protocol Overview	4
2.1. Overview of the EAP-TLS Conversation	4
2.1.1. Mutual Authentication	4
2.1.2. Termination	5
2.1.3. No Peer Authentication	8
2.1.4. Hello Retry Request	9
2.1.5. Ticket Establishment	10
2.1.6. Resumption	11
2.1.7. Privacy	13
2.1.8. Fragmentation	13
2.2. Identity Verification	14
2.3. Key Hierarchy	14
2.4. Parameter Negotiation and Compliance Requirements	15
2.5. EAP State Machines	16
3. Detailed Description of the EAP-TLS Protocol	17
4. IANA considerations	17
5. Security Considerations	18
5.1. Security Claims	18
5.2. Peer and Server Identities	18
5.3. Certificate Validation	18
5.4. Certificate Revocation	19
5.5. Packet Modification Attacks	19
5.6. Authorization	19
5.7. Resumption	20
5.8. Privacy Considerations	21
5.9. Pervasive Monitoring	23
5.10. Discovered Vulnerabilities	23
6. References	23
6.1. Normative References	23
6.2. Informative references	24
Appendix A. Updated references	27
Acknowledgments	28
Contributors	28
Authors' Addresses	28

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 and key derivation utilizing the TLS handshake protocol for cryptographic algorithms and protocol version negotiation, mutual authentication, and establishment of shared secret keying material. EAP-TLS is widely supported for authentication in IEEE 802.11 [IEEE-802.11] networks (Wi-Fi) 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. EAP-TLS [RFC5216] references TLS 1.0 [RFC2246] and TLS 1.1 [RFC4346], but works perfectly also with TLS 1.2 [RFC5246]. TLS 1.0 and 1.1 are formally deprecated and prohibited to negotiate and use [I-D.ietf-tls-oldversions-deprecate].

Weaknesses found in TLS 1.2, as well as new requirements for security, privacy, and reduced latency has 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, different privacy protection, and 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 (or higher). Therefore, aspects such as resumption, privacy handling, and key derivation need to be appropriately addressed for EAP-TLS with TLS 1.3 (or higher).

This document defines how to use EAP-TLS with TLS 1.3 (or higher) and does not change how EAP-TLS is used with older versions of TLS. While this document updates EAP-TLS [RFC5216], it remains backwards compatible with it and existing implementations of EAP-TLS. This document only describes differences compared to [RFC5216].

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 is mandatory and achieved without any additional round-trips, revocation checking is mandatory and easy 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].

2. Protocol Overview

2.1. Overview of the EAP-TLS Conversation

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 (or higher).

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 or higher, the formatting and processing of the TLS handshake SHALL be done as specified in that version of TLS. This document only lists additional and different requirements, restrictions, and processing compared to [RFC8446] and [RFC5216].

2.1.1. Mutual Authentication

The EAP server MUST authenticate with a certificate and SHOULD require the EAP 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. SessionID is deprecated in TLS 1.3 and the EAP server SHALL ignore the legacy_session_id field if TLS 1.3 is negotiated. TLS 1.3 introduced early application data which is not used in EAP-TLS. A server which receives an "early_data" extension MUST ignore the extension or respond with a HelloRetryRequest as described in Section 4.2.10 of [RFC8446]. Resumption is handled as described in Section 2.1.6. After the TLS handshake has completed and all Post-Handshake messages have been sent, the EAP server sends EAP-Success.

In the case where EAP-TLS with mutual authentication is successful (and neither HelloRetryRequest nor Post-Handshake messages are sent) the conversation will appear as shown in Figure 1. The EAP server commits to not send any more handshake messages by sending a Commitment Message (a TLS record with the application data 0x00), see Section 2.5.

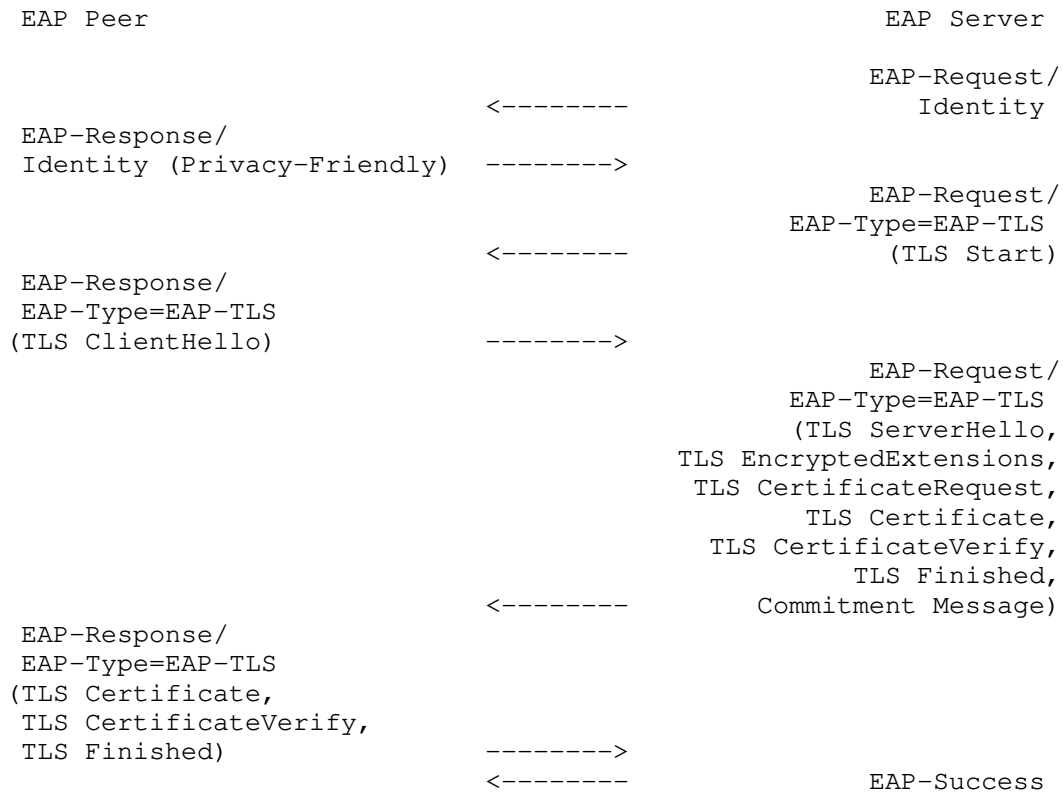


Figure 1: EAP-TLS mutual authentication

2.1.2. Termination

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 or higher. The two paragraphs below replaces the corresponding paragraphs in Section 2.1.3 of [RFC5216] when EAP-TLS is used with TLS 1.3 or higher. The other paragraphs in Section 2.1.3 of [RFC5216] still apply with the exception that SessionID is deprecated.

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

If the EAP peer authenticates successfully, the EAP 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 the EAP server sending an EAP-Success message.

Figures 2, 3, and 4 illustrate message flows in several cases where the EAP peer or EAP server sends a TLS fatal alert message. TLS warning alerts generally mean that the connection can continue normally and does not change the message flow. Note that the party receiving a TLS warning alert may choose to terminate the connection by sending a TLS fatal alert, which may add an extra round-trip, see [RFC8446].

In the case where the server rejects the ClientHello with a fatal error, the conversation will appear as shown in Figure 2. The server can also partly reject the ClientHello with a HelloRetryRequest, see Section 2.1.4.



Figure 2: EAP-TLS server rejection of ClientHello

In the case where server authentication is unsuccessful, the conversation will appear as shown in Figure 3.

```

EAP Peer                                EAP Server

                                EAP-Request/
                                Identity

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

                                EAP-Request/
                                EAP-Type=EAP-TLS
                                (TLS Start)

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

                                EAP-Request/
                                EAP-Type=EAP-TLS
                                (TLS ServerHello,
                                TLS EncryptedExtensions,
                                TLS CertificateRequest,
                                TLS Certificate,
                                TLS CertificateVerify,
                                TLS Finished,
                                Commitment Message)

EAP-Response/                          <-----
EAP-Type=EAP-TLS                      (Commitment Message)
(TLS Fatal Alert) ----->

                                <-----
                                EAP-Failure

```

Figure 3: EAP-TLS unsuccessful server authentication

In the case where the server authenticates to the peer successfully, but the peer fails to authenticate to the server, the conversation will appear as shown in Figure 4.

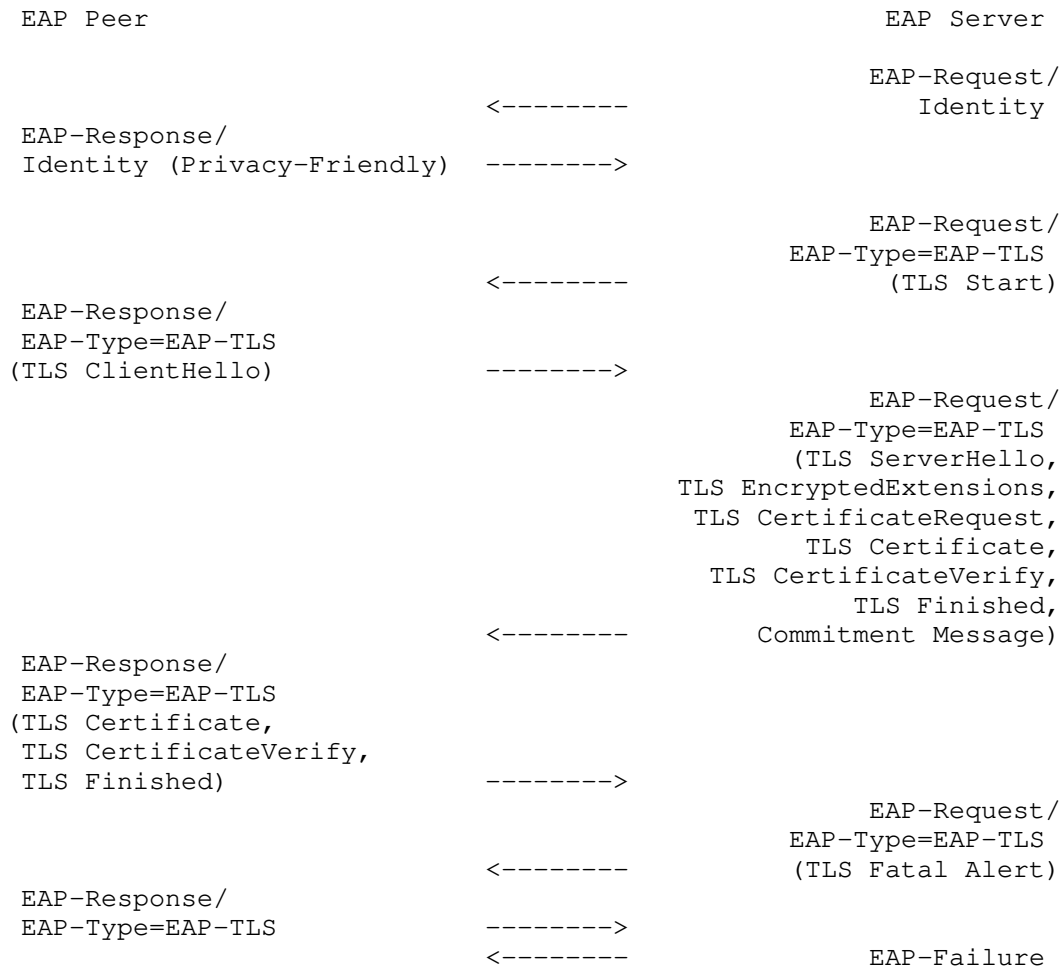


Figure 4: EAP-TLS unsuccessful client authentication

2.1.3. No Peer Authentication

In the case where EAP-TLS is used without peer authentication (e.g., emergency services, as described in [RFC7406]) the conversation will appear as shown in Figure 5.

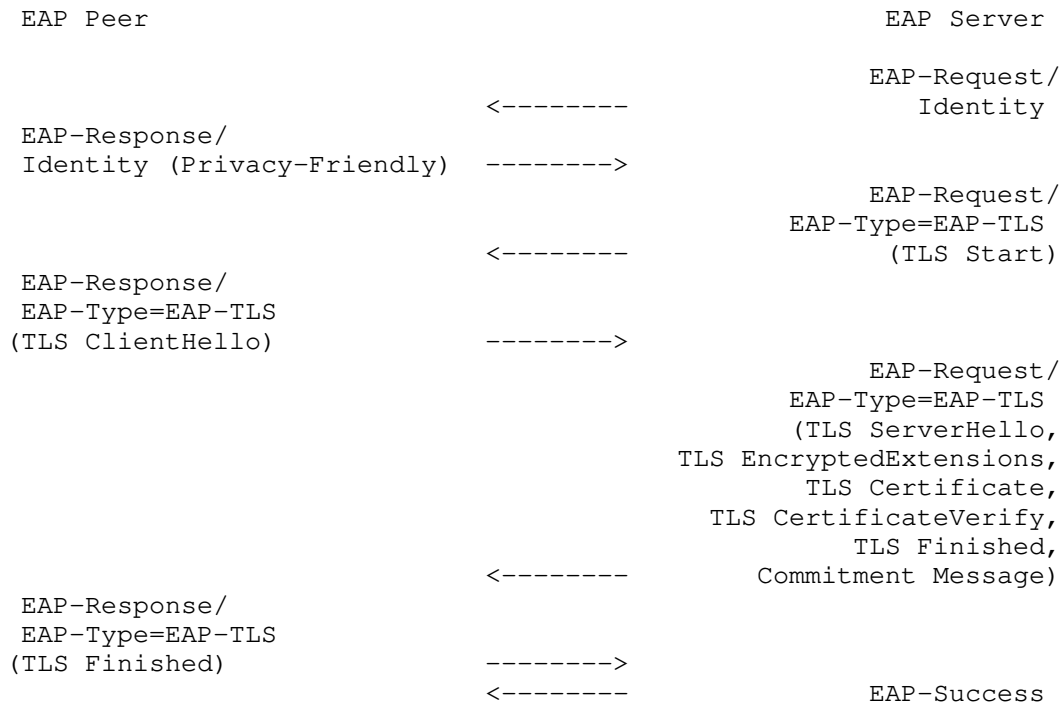


Figure 5: EAP-TLS without peer authentication

2.1.4. Hello Retry Request

TLS 1.3 [RFC8446] defines that TLS servers can send a HelloRetryRequest message in response to a ClientHello if the 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 server does not support the groups in the "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 server supports.

An EAP-TLS peer and server SHOULD support the use of HelloRetryRequest message. As noted in Section 4.1.4 of [RFC8446], the server MUST provide the supported_versions extensions and SHOULD contain the minimal set of extensions necessary for the client to generate a correct ClientHello pair. A HelloRetryRequest MUST NOT contain any extensions that were not first offered by the client in its ClientHello, with the exception of optionally the cookie extension.

The case of a successful EAP-TLS mutual authentication after the server has sent a HelloRetryRequest message is shown in Figure 6. Note the extra round-trip as a result of the HelloRetryRequest.

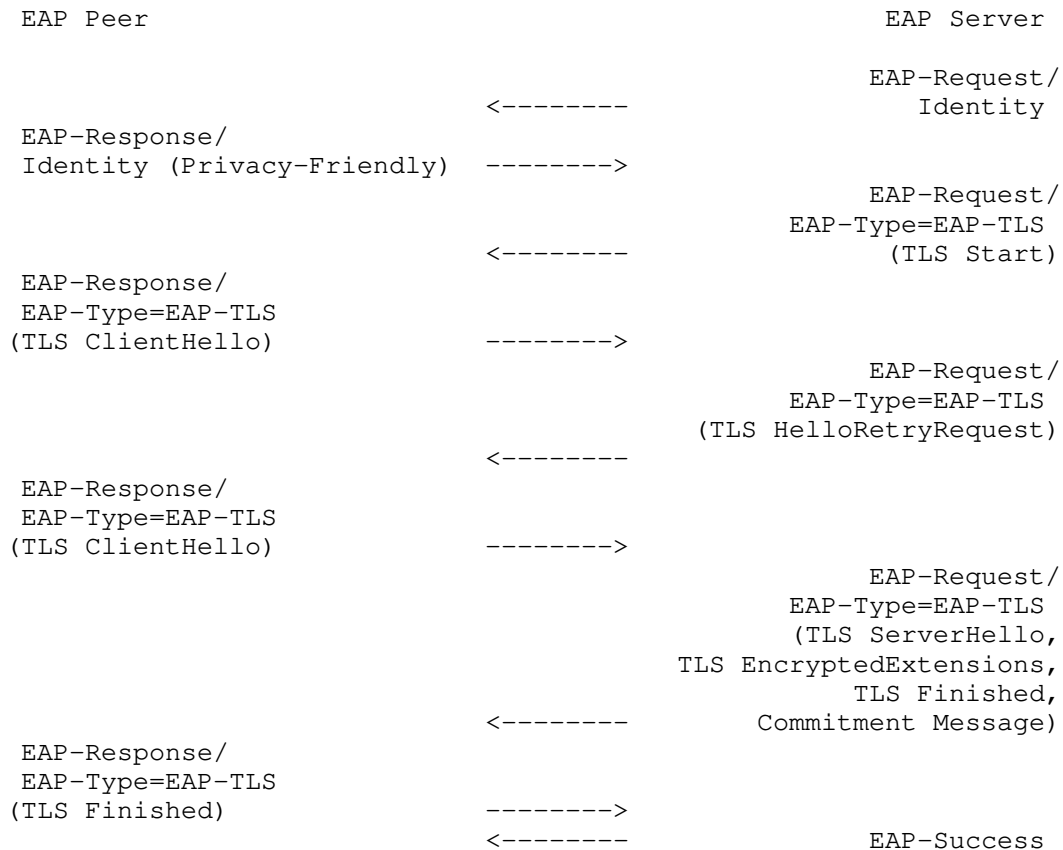


Figure 6: EAP-TLS with Hello Retry Request

2.1.5. Ticket Establishment

To enable resumption when using EAP-TLS with TLS 1.3, the EAP server MUST send a NewSessionTicket message (containing a PSK and other parameters) in the initial authentication. The NewSessionTicket is sent after the EAP server has received the Finished message in the initial authentication. The NewSessionTicket message MUST NOT include an "early_data" extension.

In the case where EAP-TLS with mutual authentication and ticket establishment is successful, the conversation will appear as shown in Figure 7.

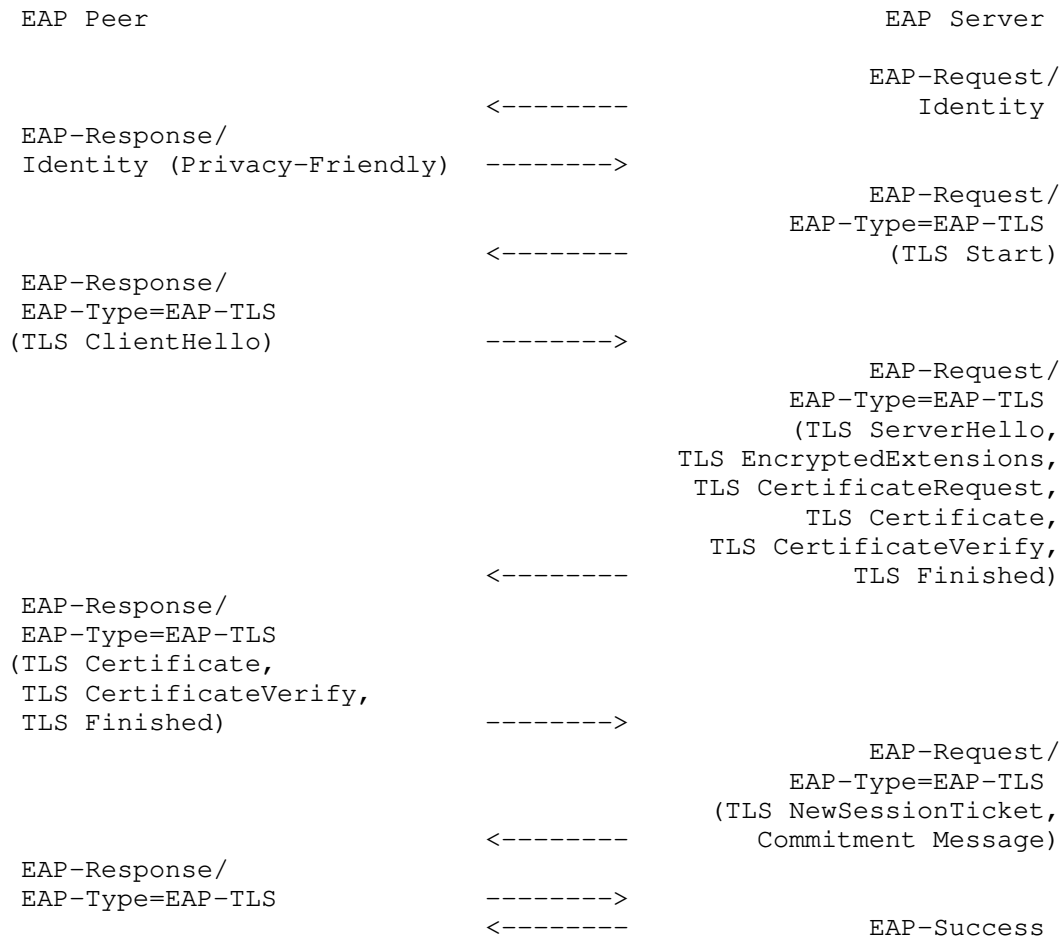


Figure 7: EAP-TLS ticket establishment

2.1.6. Resumption

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 or higher, EAP-TLS SHALL use a resumption mechanism compatible with that version 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 server, the client can use the PSK identity received in the ticket to negotiate the use of the associated PSK. If the server accepts it, then the security context of the new connection is tied to the original connection and the key derived from the initial handshake is

used to bootstrap the cryptographic state instead of a full handshake. It is left up to the EAP peer whether to use resumption, but it is RECOMMENDED that the EAP server accept resumption as long as the ticket is valid. However, the server MAY choose to require a full authentication. EAP peers and EAP servers SHOULD follow the client tracking preventions in Appendix C.4 of [RFC8446].

A subsequent authentication using resumption, where both sides authenticate successfully is shown in Figure 8.

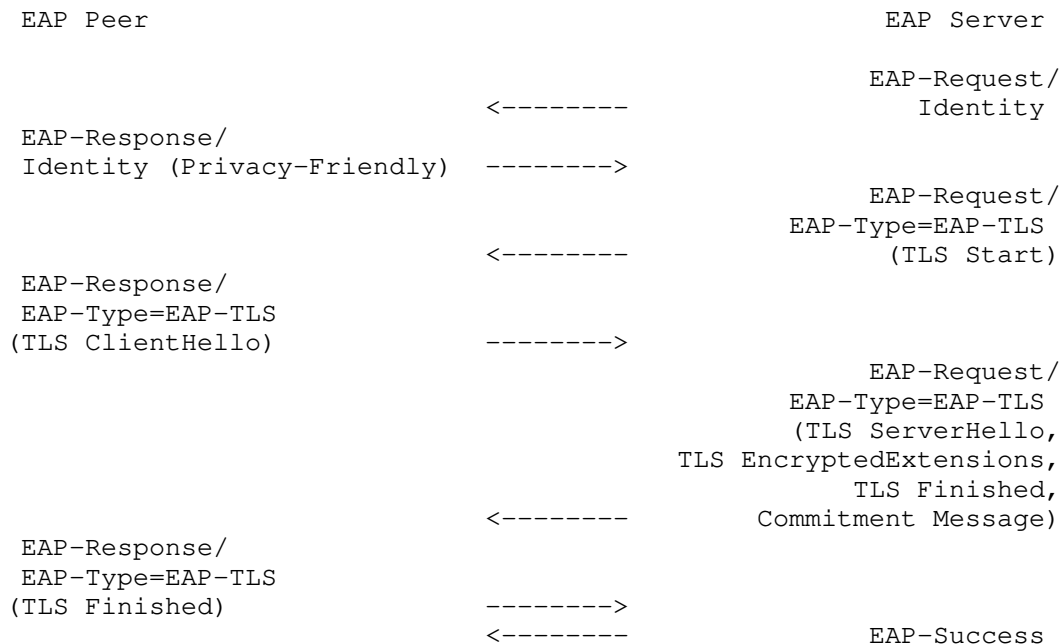


Figure 8: EAP-TLS resumption

As specified in Section 2.2 of [RFC8446], the EAP peer SHOULD supply a "key_share" extension when offering resumption, which allows the EAP server to decline resumption and continue the handshake as a full handshake. The message flow in case of mutual authentication (without the issuance of any resumption tickets) is given by Figure 1. If the EAP peer did not supply a "key_share" extension when offering resumption, the EAP server needs to reject the ClientHello and the EAP peer needs to restart a full handshake. The message flow in this case is given by Figure 2 followed by Figure 1.

Also during resumption, the server can respond with a Hello Retry Request (see Section 2.1.4) and issue a new ticket (see Section 2.1.5)

2.1.7. Privacy

TLS 1.3 significantly improves privacy when compared to earlier versions of TLS by forbidding cipher suites without confidentiality and encrypting large parts of the TLS handshake including the certificate messages.

EAP-TLS peer and server implementations supporting TLS 1.3 or higher MUST support anonymous NAIs (Network Access Identifiers) (Section 2.4 in [RFC7542]) and a client supporting TLS 1.3 MUST NOT send its username in cleartext in the Identity Response. It is RECOMMENDED to use anonymous NAIs, but other privacy-friendly identities (e.g. encrypted usernames) MAY be used.

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 or higher the EAP-TLS peer and EAP-TLS server SHALL follow the processing specified by the used version of TLS. For TLS 1.3 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.

2.1.8. Fragmentation

Including `ContentType` and `ProtocolVersion` a single TLS record may be up to 16387 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 `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 client, 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.

While Elliptic Curve Cryptography (ECC) was optional for earlier version of TLS, TLS 1.3 mandates support of ECC (see Section 9 of [RFC8446]). To avoid fragmentation, the use of ECC in certificates,

signature algorithms, and groups are RECOMMENDED when using EAP-TLS with TLS 1.3 or higher. At a 128-bit security level, this reduces public key sizes from 384 bytes (RSA and DHE) to 32-64 bytes (ECDHE) and signatures from 384 bytes (RSA) to 64 bytes (ECDSA and EdDSA). An EAP-TLS deployment MAY further reduce the certificate sizes by limiting the number of Subject Alternative Names.

Endpoints SHOULD reduce the sizes of Certificate messages by omitting certificates that the other endpoint is known to possess. When using TLS 1.3, all certificates that specifies a trust anchor may be omitted (see Section 4.4.2 of [RFC8446]). When using TLS 1.2, only the self-signed certificate that specifies the root certificate authority may be omitted (see Section 7.4.2 of [RFC5246]). EAP-TLS peers and servers SHOULD support and use the Cached Information Extension as specified in [RFC7924]. EAP-TLS peers and servers MAY use other extensions for reducing the sizes of Certificate messages, e.g. certificate compression [I-D.ietf-tls-certificate-compression].

For a detailed discussion on reducing message sizes to prevent fragmentation, see [I-D.ietf-emu-eaptls-cert].

2.2. Identity Verification

The identity provided in the EAP-Response/Identity is not authenticated by EAP-TLS. Unauthenticated information SHALL NOT be used for accounting purposes or to give authorization. The authenticator and the EAP server MAY examine the identity presented in EAP-Response/Identity for purposes such as routing and EAP method selection. They MAY reject conversations if the identity does not match their policy. Note that this also applies to resumption, see Sections 2.1.6, 5.6, and 5.7.

2.3. Key Hierarchy

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 or higher. 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 or higher the Key_Material, IV, and Method-Id SHALL be derived from the exporter_master_secret using the TLS exporter interface [RFC5705] (for TLS 1.3 this is defined in Section 7.5 of [RFC8446]).

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

All other parameters such as MSK and EMSK are derived 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)
Enc-RECV-Key    = MSK(0, 31)
Enc-SEND-Key    = MSK(32, 63)
RECV-IV         = IV(0, 31)
SEND-IV         = IV(32, 63)
```

The use of these keys is specific to the lower layer, as described [RFC5247].

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 without having to extract the Master Secret, ClientHello.random, and ServerHello.random in a non-standard way.

2.4. Parameter Negotiation and Compliance Requirements

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 or higher.

When EAP-TLS is used with TLS version 1.3 or higher, the EAP-TLS peers and servers MUST comply with the compliance requirements (mandatory-to-implement cipher suites, signature algorithms, key exchange algorithms, extensions, etc.) for the TLS version used. For

TLS 1.3 the compliance requirements are defined in Section 9 of [RFC8446].

While EAP-TLS does not protect any application data, 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

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. One such Post-Handshake message is NewSessionTicket. The NewSessionTicket can be used for resumption. After sending TLS Finished, the EAP server may send any number of Post-Handshake messages in separate EAP-Requests. To decrease the uncertainty for the EAP peer, the following procedure MUST be followed:

When an EAP server has sent its last handshake message (Finished or a Post-Handshake), it commits to not sending any more handshake messages by sending a Commitment Message. The Commitment Message is a TLS record with application data 0x00 (i.e. a TLS record with TLSPlaintext.type = application_data, TLSPlaintext.length = 1, and TLSPlaintext.fragment = 0x00). Note that the length of the plaintext is greater than the corresponding TLSPlaintext.length due to the inclusion of TLSInnerPlaintext.type and any padding supplied by the sender. EAP server implementations MUST set TLSPlaintext.fragment to 0x00, but EAP peer implementations MUST accept any application data as a Commitment Message from the EAP server to not send any more handshake messages. The Commitment Message may be sent in the same EAP-Request as the last handshake record or in a separate EAP-Request. Sending the Commitment Message in a separate EAP-Request adds an additional round-trip, but may be necessary in TLS implementations that only implement a subset of TLS 1.3. In the case where the server sends the Commitment Message in a separate EAP-Request, the conversation will appear as shown in Figure 9. After sending the Commitment Message, the EAP server may only send an EAP-Success, an EAP-Failure, or an EAP-Request with a TLS Alert Message.

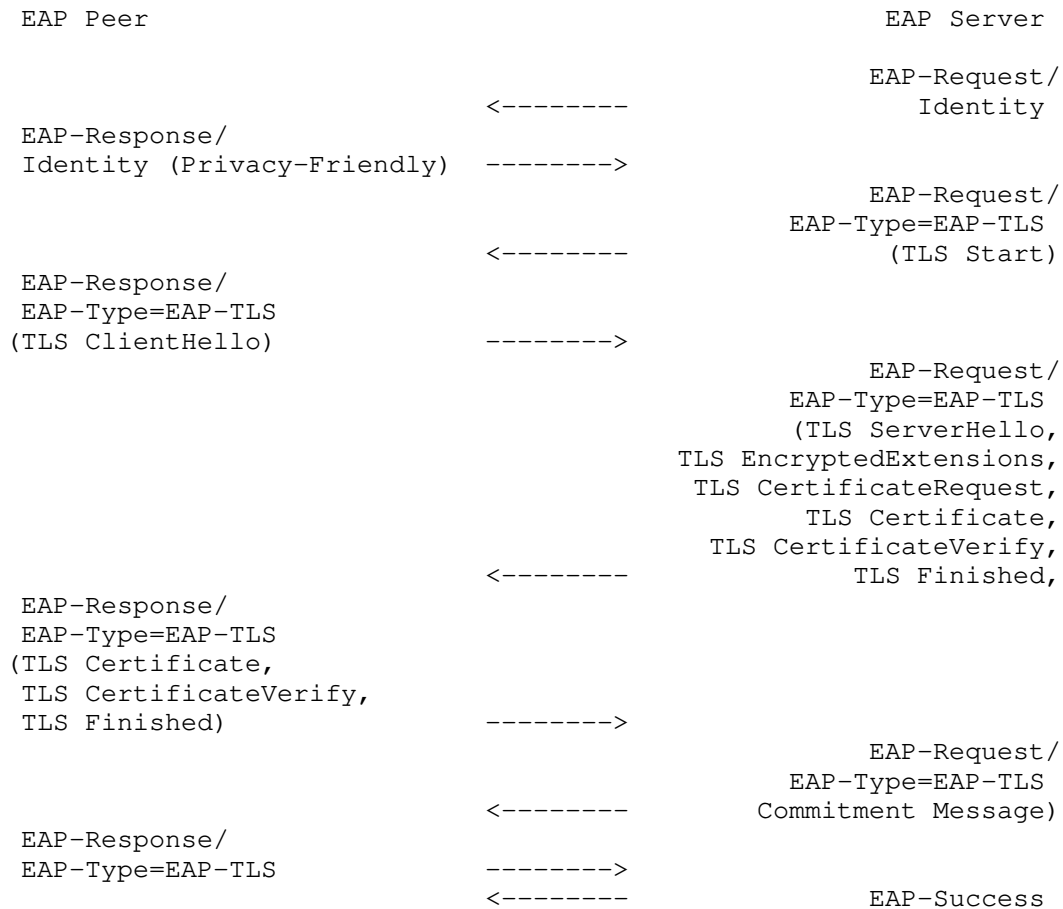


Figure 9: Commit in separate EAP-Request

3. Detailed Description of the EAP-TLS Protocol

No updates to [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 memo 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, IV and Method-Id as defined in Section 2.3:

- o "EXPORTER_EAP_TLS_Key_Material"
- o "EXPORTER_EAP_TLS_IV"
- o "EXPORTER_EAP_TLS_Method-Id"

5. Security Considerations

5.1. Security Claims

Using EAP-TLS with TLS 1.3 does not change the security claims for EAP-TLS as given in Section 4.1 of [RFC5216]. However, it strengthens several of the claims as described in the following updates to the notes given in Section 4.1 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 by mandating cipher suites with confidentiality and encrypting certificates and some of the extensions, see [RFC8446]. When using EAP-TLS with TLS 1.3, the use of privacy is mandatory and does not cause any additional round-trips.

[3] Key strength: TLS 1.3 forbids all algorithms with known weaknesses including 3DES, CBC mode, RC4, SHA-1, and MD5. TLS 1.3 only supports cryptographic algorithms offering at least 112-bit security, see [RFC8446].

[4] Cryptographic Negotiation: TLS 1.3 increases the number of cryptographic parameters that are negotiated in the handshake. 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 [RFC5216].

5.3. Certificate Validation

No updates to [RFC5216].

5.4. Certificate Revocation

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

EAP-TLS peers and servers supporting TLS 1.3 MUST support Certificate Status Requests (OCSP stapling) as specified in [RFC6066] and Section 4.4.2.1 of [RFC8446]. When EAP-TLS is used with TLS 1.3, the peer and server MUST use Certificate Status Requests [RFC6066] for the server's certificate chain and the EAP peer MUST treat a CertificateEntry (except the trust anchor) without a valid CertificateStatus extension as invalid and abort the handshake with an appropriate alert. When EAP-TLS is used with TLS 1.3, the server MUST check the revocation status of the certificates in the client's certificate chain.

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 [RFC4366]. This enables sending OCSP information for all certificates in the certificate chain.

5.5. Packet Modification Attacks

No updates to [RFC5216].

5.6. Authorization

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 server. This information can include, but is not limited to, information about the authenticator, information about the EAP peer, or information about the protocol layers above or below EAP (MAC addresses, IP addresses, port numbers, WiFi SSID, etc.). 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.

5.7. Resumption

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 all version of TLS.

When resumption occurs, it is based on cached information at the TLS layer. To perform resumption in a secure way, 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. We use the term "cached data" to describe such information. Authorization during resumption MUST be based on such cached data. The EAP peer and sever 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 information is not sufficiently current, the EAP Peer or EAP Server MAY force a full TLS handshake.

There are two ways to retrieve the cached information from the original full handshake. The first method is that the 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 TLS server encapsulates the information into a ticket and sends it to the client. The client can subsequently do resumption using the obtained ticket. Note that the client still needs to cache the information locally. The following requirements apply to both methods.

If the EAP server or EAP client do not apply any authorization policies, they MAY allow resumption where no cached data is available. In all other cases, they MUST cache data during the initial full authentication to enable resumption. The cached data MUST be sufficient to make authorization decisions during resumption.

If cached data cannot be retrieved in a secure way, resumption **MUST NOT** be done.

The above requirements also apply if the EAP 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.

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 leading to them obtaining access that they should not have.

If any authorization, accounting, or policy decisions were made with information that have 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 servers **MAY** reject resumption where the information supplied during resumption does not match the information supplied during the original authentication. Where a good decision is unclear, EAP servers **SHOULD** reject the resumption.

Section 4.2.11, 8.1, and 8.2 of [RFC8446] provides security consideration for resumption.

5.8. Privacy Considerations

[RFC6973] suggests that the privacy considerations of IETF protocols be documented.

TLS 1.3 offers much better privacy than earlier versions of TLS as discussed in Section 2.1.7. 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 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 always confidentiality protected (e.g. using Anonymous NAIs). However, 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 peers have the same domain, no additional information is leaked. If a domain name is used by a small subset of the EAP 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 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 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. 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 SHOULD NOT be formed by a fixed mapping that stays the same across multiple different authentications.

An EAP peer with a policy allowing communication with EAP servers supporting only TLS 1.2 without privacy and with a static RSA key exchange is vulnerable to disclosure of the peer username. An active attacker can in this case make the EAP peer believe that an EAP server supporting TLS 1.3 only supports TLS 1.2 without privacy. The attacker can simply impersonate the EAP server and negotiate TLS 1.2 with static RSA key exchange and send an TLS alert message when the EAP peer tries to use privacy by sending an empty certificate message. Since the attacker (impersonating the EAP 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 peer may inadvertently disclose its identity (username) to an attacker.

Therefore, it is RECOMMENDED for EAP peers to not use EAP-TLS with TLS 1.2 and static RSA based cipher suites without privacy.

5.9. Pervasive Monitoring

As required by [RFC7258], work on IETF protocols needs to consider the effects of pervasive monitoring and mitigate them when possible.

Pervasive Monitoring is widespread surveillance of users. By encrypting more information and by mandating the use of privacy, 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 wider 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

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 [RFC7525] provides recommendations 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 SHOULD mitigate known attacks and follow the recommendations in [RFC7525] and [I-D.ietf-tls-oldversions-deprecate]. The use of TLS 1.3 mitigates most of the known attacks.

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>>.
- [RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <<https://www.rfc-editor.org/info/rfc7924>>.
- [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>>.
- [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

- [I-D.ietf-emu-eaptlscert]
Sethi, M., Mattsson, J., and S. Turner, "Handling Large Certificates and Long Certificate Chains in TLS-based EAP Methods", draft-ietf-emu-eaptlscert-00 (work in progress), August 2019.
- [I-D.ietf-tls-certificate-compression]
Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", draft-ietf-tls-certificate-compression-05 (work in progress), April 2019.
- [I-D.ietf-tls-oldversions-deprecate]
Moriarty, K. and S. Farrell, "Deprecating TLSv1.0 and TLSv1.1", draft-ietf-tls-oldversions-deprecate-05 (work in progress), June 2019.
- [IEEE-802.11]
Institute of Electrical and Electronics Engineers, "IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012) , December 2016.
- [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-2010 , February 2010.
- [MultaFire]
MultaFire, "MultaFire Release 1.1 specification", 2019.
- [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>>.
- [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>>.
- [RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <<https://www.rfc-editor.org/info/rfc4366>>.
- [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>>.

- [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>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [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>>.
- [TS.33.501] 3GPP, "Security architecture and procedures for 5G System", 3GPP TS 33.501 15.5.0, June 2019.

Appendix A. Updated references

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

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

All references to [RFC3280] are updated with [RFC5280].

All references to [RFC4282] are updated with [RFC7542].

Acknowledgments

The authors want to thank Bernard Aboba, Jari Arkko, Alan DeKok, Ari Keraenen, Jouni Malinen, Oleg Pekar, Eric Rescorla, Jim Schaad, and Vesa Torvinen for comments and suggestions on the draft.

Contributors

Alan DeKok, FreeRADIUS

Authors' Addresses

John Preuss Mattsson
Ericsson
Stockholm 164 40
Sweden

Email: john.mattsson@ericsson.com

Mohit Sethi
Ericsson
Jorvas 02420
Finland

Email: mohit@piuha.net

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).

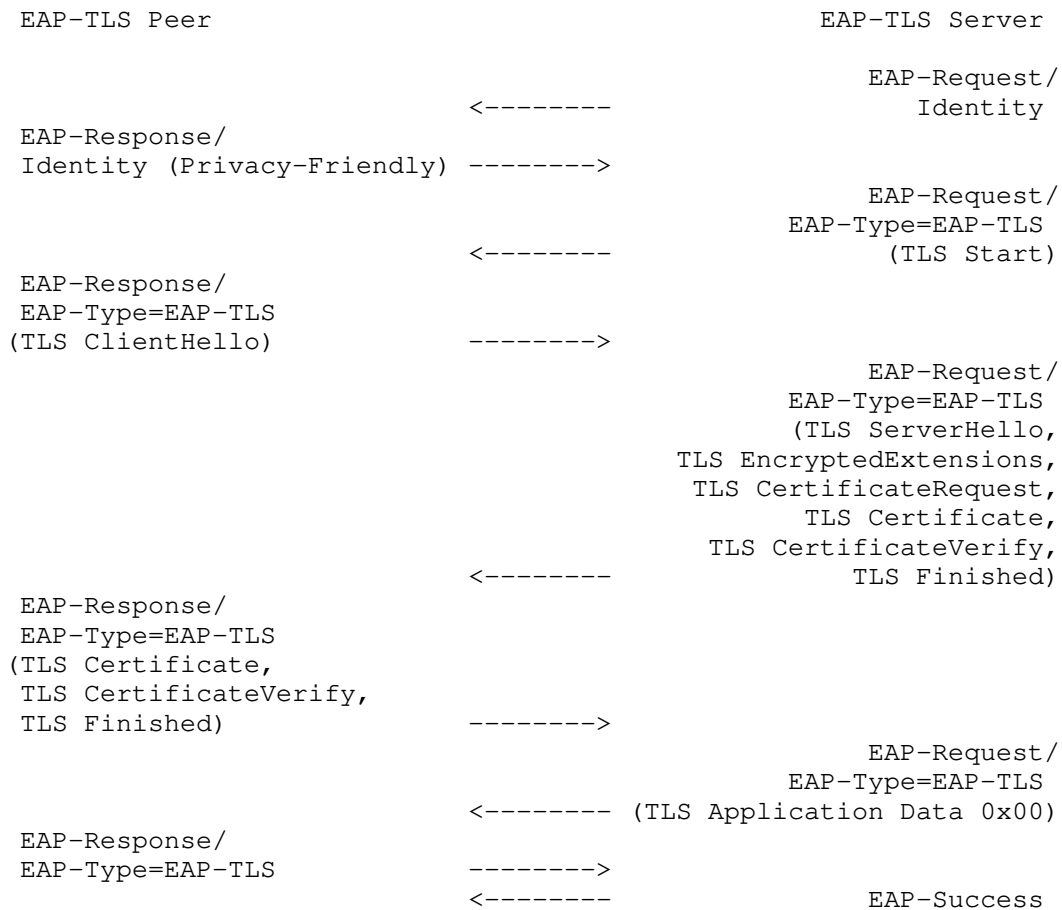


Figure 1: EAP-TLS mutual authentication

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.

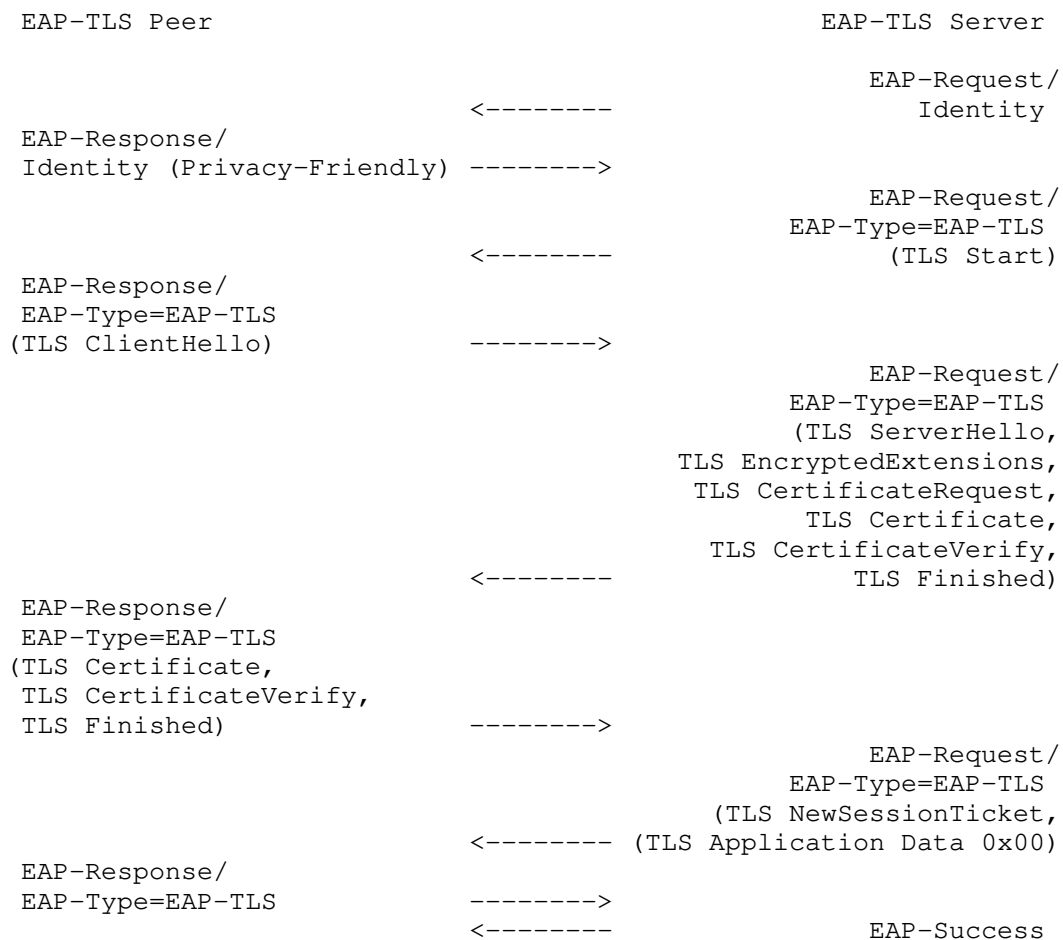


Figure 2: EAP-TLS ticket establishment

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.

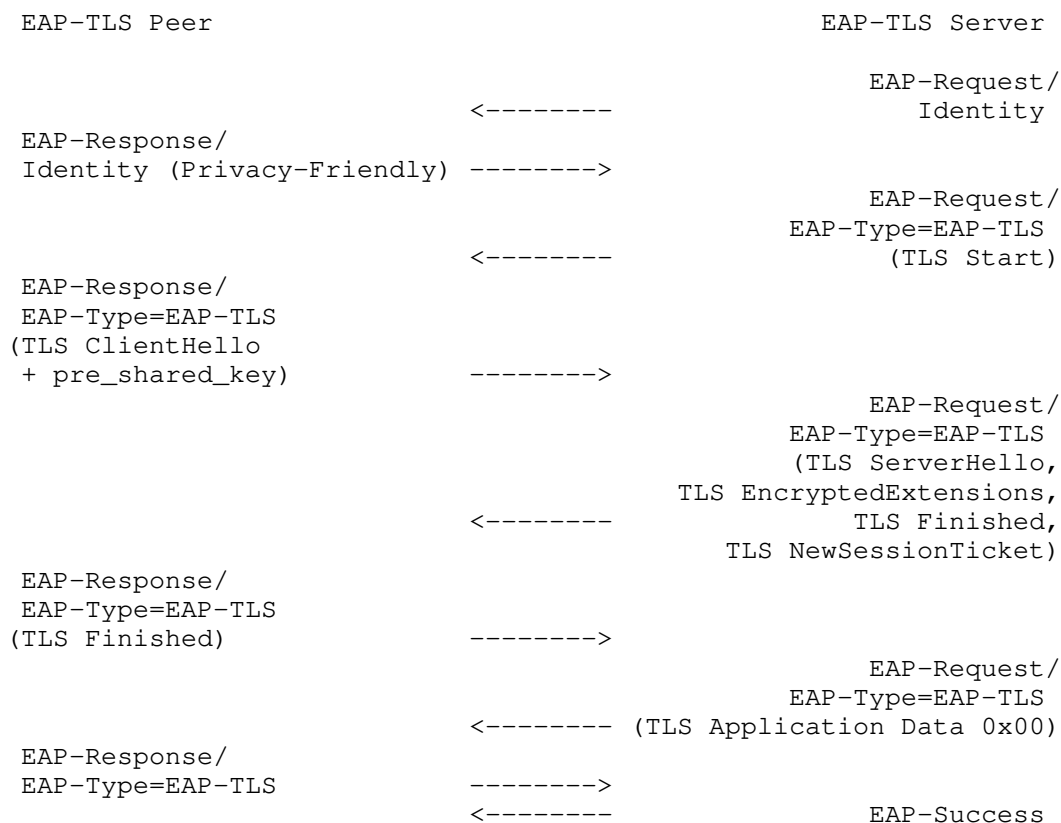


Figure 3: EAP-TLS resumption

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.

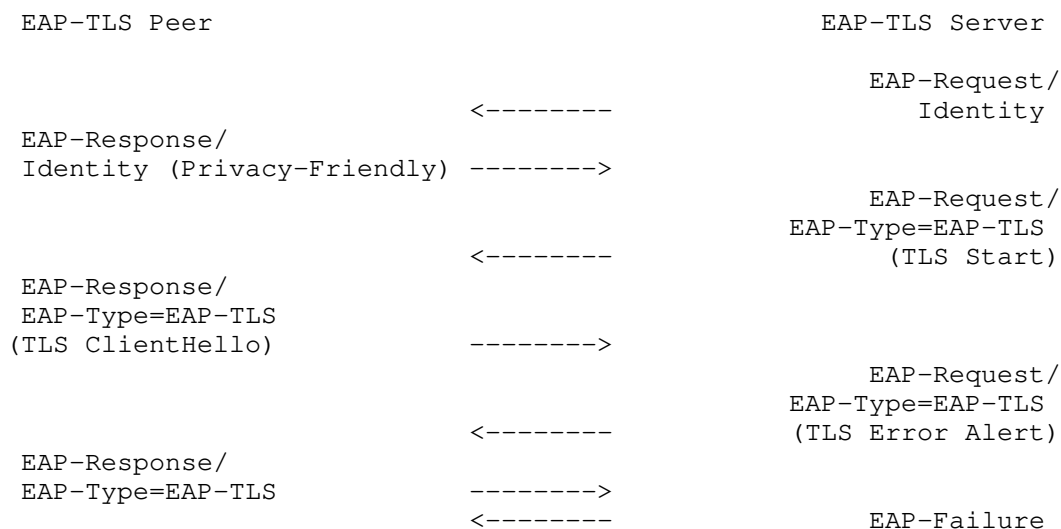


Figure 4: EAP-TLS server rejection of ClientHello

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

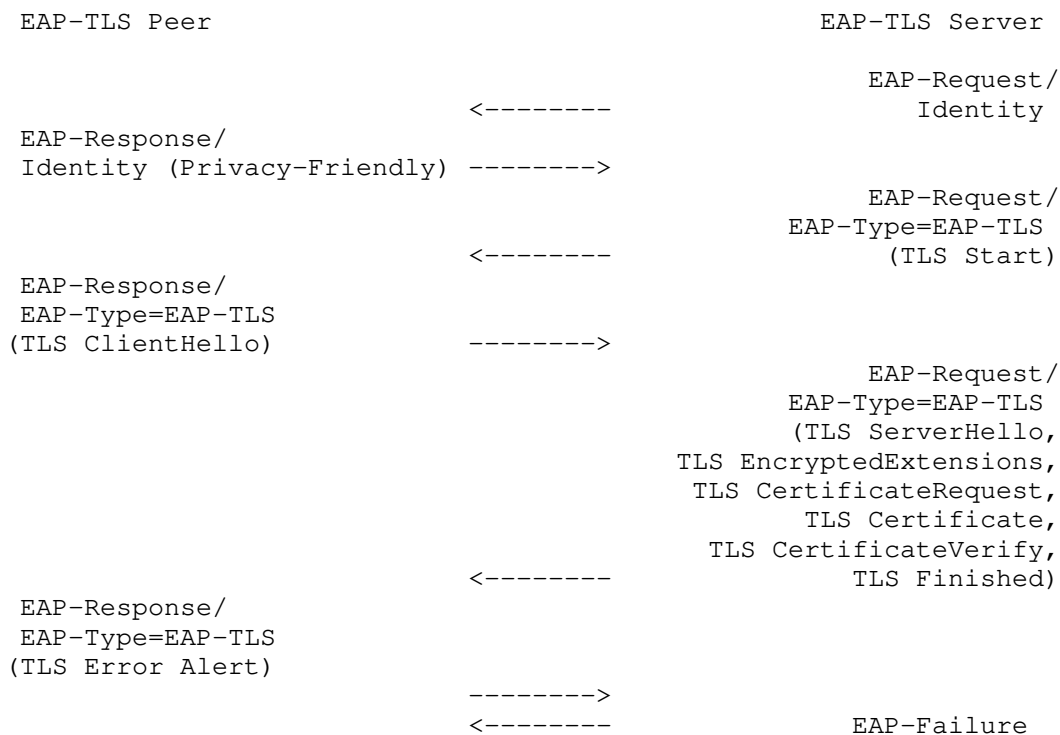


Figure 5: EAP-TLS unsuccessful EAP-TLS server authentication

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.

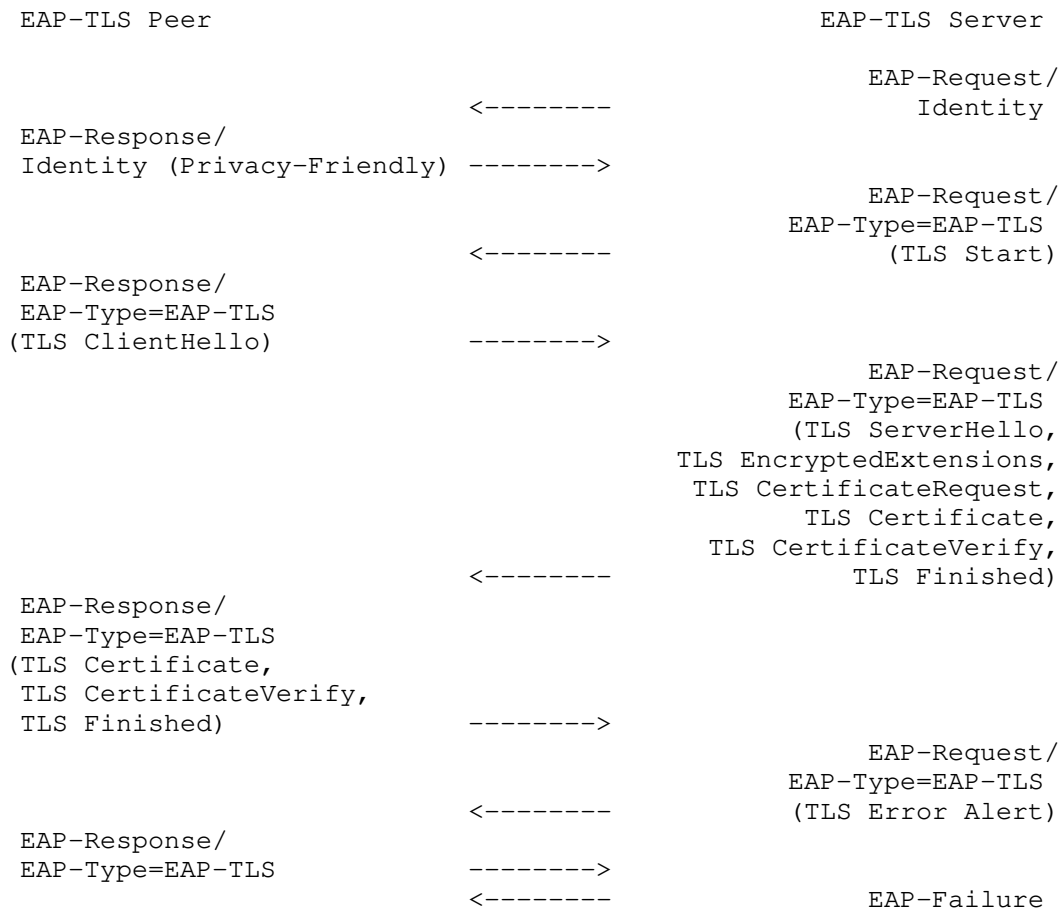


Figure 6: EAP-TLS unsuccessful client authentication

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]).

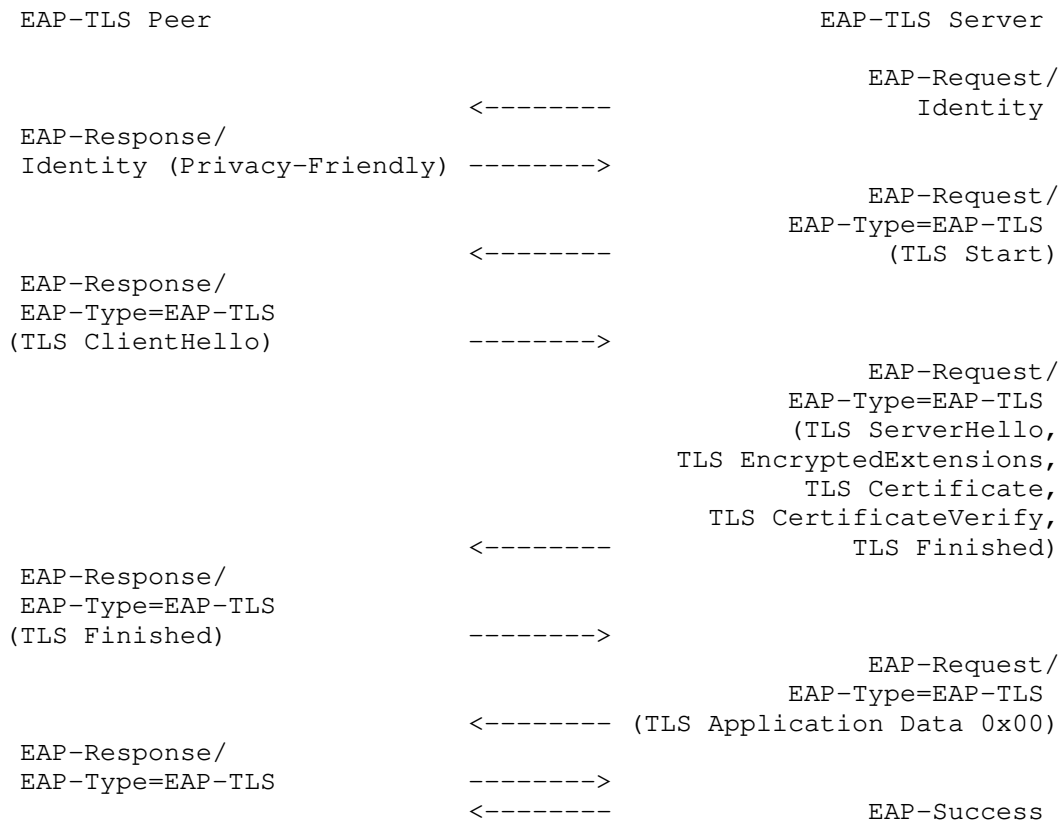


Figure 7: EAP-TLS without peer authentication

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 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
Intended status: Informational
Expires: May 24, 2021

M. Sethi
J. Mattsson
Ericsson
S. Turner
sn3rd
November 20, 2020

Handling Large Certificates and Long Certificate Chains
in TLS-based EAP Methods
draft-ietf-emu-eaptls-cert-08

Abstract

The Extensible Authentication Protocol (EAP), defined in RFC3748, provides a standard mechanism for support of multiple authentication methods. EAP-Transport Layer Security (EAP-TLS) and other TLS-based EAP methods are widely deployed and used for network access authentication. Large certificates and long certificate chains combined with authenticators that drop an EAP session after only 40 - 50 round-trips is a major deployment problem. This document looks at this problem in detail and describes the potential solutions available.

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 May 24, 2021.

Copyright Notice

Copyright (c) 2020 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	2
2. Terminology	3
3. Experience with Deployments	4
4. Handling of Large Certificates and Long Certificate Chains .	5
4.1. Updating Certificates and Certificate Chains	5
4.1.1. Guidelines for Certificates	6
4.1.2. Pre-distributing and Omitting CA certificates	7
4.1.3. Using Fewer Intermediate Certificates	7
4.2. Updating TLS and EAP-TLS Code	7
4.2.1. URLs for Client Certificates	7
4.2.2. Caching Certificates	8
4.2.3. Compressing Certificates	8
4.2.4. Compact TLS 1.3	9
4.2.5. Suppressing Intermediate Certificates	9
4.2.6. Raw Public Keys	9
4.2.7. New Certificate Types and Compression Algorithms . .	10
4.3. Updating Authenticators	10
5. IANA Considerations	11
6. Security Considerations	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Acknowledgements	14
Authors' Addresses	14

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] [I-D.ietf-emu-eap-tls13] relies on TLS [RFC8446] to provide strong mutual authentication with certificates [RFC5280] and is widely deployed and often used for network access authentication. There are also many other TLS-based EAP methods, such as Flexible Authentication via Secure Tunneling (EAP-FAST) [RFC4851], Tunneled Transport Layer Security (EAP-TTLS) [RFC5281], Tunnel Extensible Authentication Protocol (EAP-TEAP) [RFC7170], and possibly many vendor-specific EAP methods.

Certificates in EAP deployments can be relatively large, and the certificate chains can be long. Unlike the use of TLS on the web, where typically only the TLS server is authenticated; EAP-TLS deployments typically authenticate both the EAP peer and the EAP server. Also, from deployment experience, EAP peers typically have longer certificate chains than servers. This is because EAP peers often follow organizational hierarchies and tend to have many intermediate certificates. Thus, EAP-TLS authentication usually involves exchange of significantly more octets than when TLS is used as part of HTTPS.

Section 3.1 of [RFC3748] states that EAP implementations can assume a Maximum Transmission Unit (MTU) of at least 1020 octets from lower layers. The EAP fragment size in typical deployments is just 1020 - 1500 octets (since the maximum Ethernet frame size is ~ 1500 bytes). Thus, EAP-TLS authentication needs to be fragmented into many smaller packets for transportation over the lower layers. Such fragmentation not only can negatively affect the latency, but also results in other challenges. For example, some EAP authenticator (access point) implementations will drop an EAP session if it has not finished after 40 - 50 round-trips. This is a major problem and means that in many situations, the EAP peer cannot perform network access authentication even though both the sides have valid credentials for successful authentication and key derivation.

Not all EAP deployments are constrained by the MTU of the lower layer. For example, some implementations support EAP over Ethernet "Jumbo" frames that can easily allow very large EAP packets. Larger packets will naturally help lower the number of round trips required for successful EAP-TLS authentication. However, deployment experience has shown that these jumbo frames are not always implemented correctly. Additionally, EAP fragment size is also restricted by protocols such as RADIUS [RFC2865] which are responsible for transporting EAP messages between an authenticator and an EAP server. RADIUS can generally transport only about 4000 octets of EAP in a single message (the maximum length of RADIUS packet is restricted to 4096 octets in [RFC2865]).

This document looks at related work and potential tools available for overcoming the deployment challenges induced by large certificates and long certificate chains. It then discusses the solutions available to overcome these challenges.

2. 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 [RFC3748], EAP-TLS [RFC5216], and TLS [RFC8446]. In particular, this document frequently uses the following terms as they have been defined in [RFC5216]:

Authenticator The entity initiating EAP authentication. Typically implemented as part of a network switch or a wireless access point.

EAP peer The entity that responds to the authenticator. In [IEEE-802.1X], this entity is known as the supplicant. In EAP-TLS, the EAP peer implements the TLS client role.

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. In EAP-TLS, the EAP server implements the TLS server role.

The document additionally uses the terms "trust anchor" and "certification path" defined in [RFC5280].

3. Experience with Deployments

As stated earlier, the EAP fragment size in typical deployments is just 1020 - 1500 octets. A certificate can, however, be large for a number of reasons:

- o It can have a long Subject Alternative Name field.
- o It can have long Public Key and Signature fields.
- o It can contain multiple object identifiers (OID) that indicate the permitted uses of the certificate as noted in Section 5.3 of [RFC5216]. Most implementations verify the presence of these OIDs for successful authentication.
- o It can contain multiple organization fields to reflect the multiple group memberships of a user (in a client certificate).

A certificate chain (called a certification path in [RFC5280]) in EAP-TLS can commonly have 2 - 6 intermediate certificates between the end-entity certificate and the trust anchor.

The size of certificates (and certificate chains) may also increase many-fold in the future with the introduction of quantum-safe cryptography. For example, lattice-based cryptography would have public keys of approximately 1000 bytes and signatures of approximately 2000 bytes.

Many access point implementations drop EAP sessions that do not complete within 40 - 50 round-trips. This means that if the chain is larger than ~ 60 kbytes, EAP-TLS authentication cannot complete successfully in most deployments.

4. Handling of Large Certificates and Long Certificate Chains

This section discusses some possible alternatives for overcoming the challenge of large certificates and long certificate chains in EAP-TLS authentication. Section 4.1 considers recommendations that require an update of the certificates or certificate chains used for EAP-TLS authentication without requiring changes to the existing EAP-TLS code base. It also provides some guidelines that should be followed when issuing certificates for use with EAP-TLS. Section 4.2 considers recommendations that rely on updates to the EAP-TLS implementations and can be deployed with existing certificates. Finally, Section 4.3 briefly discusses what could be done to update or reconfigure authenticators when it is infeasible to replace deployed components giving a solution which can be deployed without changes to existing certificates or code.

4.1. Updating Certificates and Certificate Chains

Many IETF protocols now use elliptic curve cryptography (ECC) [RFC6090] for the underlying cryptographic operations. The use of ECC can reduce the size of certificates and signatures. For example, at a 128-bit security level, the size of a public key with traditional RSA is about 384 bytes, while the size of a public key with ECC is only 32-64 bytes. Similarly, the size of a digital signature with traditional RSA is 384 bytes, while the size is only 64 bytes with elliptic curve digital signature algorithm (ECDSA) and Edwards-curve digital signature algorithm (EdDSA) [RFC8032]. Using certificates that use ECC can reduce the number of messages in EAP-TLS authentication, which can alleviate the problem of authenticators dropping an EAP session because of too many round-trips. In the absence of a standard application profile specifying otherwise, TLS 1.3 [RFC8446] requires implementations to support ECC. New cipher suites that use ECC are also specified for TLS 1.2 [RFC8422]. Using ECC-based cipher suites with existing code can significantly reduce the number of messages in a single EAP session.

4.1.1. Guidelines for Certificates

The general guideline of keeping the certificate size small by not populating fields with excessive information can help avert the problems of failed EAP-TLS authentication. More specific recommendations for certificates used with EAP-TLS are as follows:

- o Object Identifier (OID) is an ASN.1 data type that defines unique identifiers for objects. The OID's ASN.1 value, which is a string of integers, is then used to name objects to which they relate. The Distinguished Encoding Rules (DER) specify that the first two integers always occupy one octet and subsequent integers are base 128-encoded in the fewest possible octets. OIDs are used lavishly in X.509 certificates [RFC5280] and while not all can be avoided, e.g., OIDs for extensions or algorithms and their associate parameters, some are well within the certificate issuer's control:
 - * Each naming attribute in a DN (Directory Name) has one. DNs are used in the issuer and subject fields as well as numerous extensions. A shallower naming will be smaller, e.g., C=FI, O=Example, SN=B0A123499EFC as against C=FI, O=Example, OU=Division 1, SOPN=Southern Finland, CN=Coolest IoT Gadget Ever, SN=B0A123499EFC.
 - * Every certificate policy (and qualifier) and any mappings to another policy uses identifiers. Consider carefully what policies apply.
- o DirectoryString and GeneralName types are used extensively to name things, e.g., the DN naming attribute O= (the organizational naming attribute) DirectoryString includes "Example" for the Example organization and uniformResourceIdentifier can be used to indicate the location of the CRL, e.g., "http://crl.example.com/sfig2sl-128.crl", in the CRL Distribution Point extension. For these particular examples, each character is a byte. For some non-ASCII character strings in the DN, characters can be multi-byte. Obviously, the names need to be unique, but there is more than one way to accomplish this without long strings. This is especially true if the names are not meant to be meaningful to users.
- o Extensions are necessary to comply with [RFC5280], but the vast majority are optional. Include only those that are necessary to operate.
- o As stated earlier, certificate chains of the EAP peer often follow organizational hierarchies. In such cases, information in intermediate certificates (such as postal addresses) do not

provide any additional value and they can be shortened (for example: only including the department name instead of the full postal address).

4.1.2. Pre-distributing and Omitting CA certificates

The TLS Certificate message conveys the sending endpoint's certificate chain. TLS allows endpoints to reduce the size of the Certificate message by omitting certificates that the other endpoint is known to possess. When using TLS 1.3, all certificates that specify a trust anchor known by the other endpoint may be omitted (see Section 4.4.2 of [RFC8446]). When using TLS 1.2 or earlier, only the self-signed certificate that specifies the root certificate authority may be omitted (see Section 7.4.2 of [RFC5246]). Therefore, updating TLS implementations to version 1.3 can help to significantly reduce the number of messages exchanged for EAP-TLS authentication. The omitted certificates need to be pre-distributed independently of TLS and the TLS implementations need to be configured to omit these pre-distributed certificates.

4.1.3. Using Fewer Intermediate Certificates

The EAP peer certificate chain does not have to mirror the organizational hierarchy. For successful EAP-TLS authentication, certificate chains SHOULD NOT contain more than 4 intermediate certificates.

Administrators responsible for deployments using TLS-based EAP methods can examine the certificate chains and make rough calculations about the number of round trips required for successful authentication. For example, dividing the total size of all the certificates in the peer and server certificate chain (in bytes) by 1020 bytes will indicate the minimum number of round trips required. If this number exceeds 50, then, administrators can expect failures with many common authenticator implementations.

4.2. Updating TLS and EAP-TLS Code

This section discusses how the fragmentation problem can be avoided by updating the underlying TLS or EAP-TLS implementation. Note that in some cases the new feature may already be implemented in the underlying library and simply needs to be taken into use.

4.2.1. URLs for Client Certificates

[RFC6066] defines the "client_certificate_url" extension which allows TLS clients to send a sequence of Uniform Resource Locators (URLs) instead of the client certificate. URLs can refer to a single

certificate or a certificate chain. Using this extension can curtail the amount of fragmentation in EAP deployments thereby allowing EAP sessions to successfully complete.

4.2.2. Caching Certificates

The TLS Cached Information Extension [RFC7924] specifies an extension where a server can exclude transmission of certificate information cached in an earlier TLS handshake. The client and the server would first execute the full TLS handshake. The client would then cache the certificate provided by the server. When the TLS client later connects to the same TLS server without using session resumption, it can attach the "cached_info" extension to the ClientHello message. This would allow the client to indicate that it has cached the certificate. The client would also include a fingerprint of the server certificate chain. If the server's certificate has not changed, then the server does not need to send its certificate and the corresponding certificate chain again. In case information has changed, which can be seen from the fingerprint provided by the client, the certificate payload is transmitted to the client to allow the client to update the cache. The extension however necessitates a successful full handshake before any caching. This extension can be useful when, for example, a successful authentication between an EAP peer and EAP server has occurred in the home network. If authenticators in a roaming network are stricter at dropping long EAP sessions, an EAP peer can use the Cached Information Extension to reduce the total number of messages.

However, if all authenticators drop the EAP session for a given EAP peer and EAP server combination, a successful full handshake is not possible. An option in such a scenario would be to cache validated certificate chains even if the EAP-TLS exchange fails, but such caching is currently not specified in [RFC7924].

4.2.3. Compressing Certificates

The TLS working group is also working on an extension for TLS 1.3 [I-D.ietf-tls-certificate-compression] that allows compression of certificates and certificate chains during full handshakes. The client can indicate support for compressed server certificates by including this extension in the ClientHello message. Similarly, the server can indicate support for compression of client certificates by including this extension in the CertificateRequest message. While such an extension can alleviate the problem of excessive fragmentation in EAP-TLS, it can only be used with TLS version 1.3 and higher. Deployments that rely on older versions of TLS cannot benefit from this extension.

4.2.4. Compact TLS 1.3

[I-D.ietf-tls-ctls] defines a "compact" version of TLS 1.3 and reduces the message size of the protocol by removing obsolete material and using more efficient encoding. It also defines a compression profile with which either side can define a dictionary of "known certificates". Thus, cTLS could provide another mechanism for EAP-TLS deployments to reduce the size of messages and avoid excessive fragmentation.

4.2.5. Suppressing Intermediate Certificates

For a client that has all intermediate certificates in the certificate chain, having the server send intermediates in the TLS handshake increases the size of the handshake unnecessarily. [I-D.thomson-tls-sic] proposes an extension for TLS 1.3 that allows a TLS client that has access to the complete set of published intermediate certificates to inform servers of this fact so that the server can avoid sending intermediates, reducing the size of the TLS handshake. The mechanism is intended to be complementary with certificate compression.

The Authority Information Access (AIA) extension specified in [RFC5280] can be used with end-entity and CA certificates to access information about the issuer of the certificate in which the extension appears. For example, it can be used to provide the address of the OCSP responder from where revocation status of the certificate (in which the extension appears) can be checked. It can also be used to obtain the issuer certificate. Thus, the AIA extension can reduce the size of the certificate chain by only including a pointer to the issuer certificate instead of including the entire issuer certificate. However, it requires the side receiving the certificate containing the extension to have network connectivity (unless the information is already cached locally). Naturally, such indirection cannot be used for the server certificate (since EAP peers in most deployments do not have network connectivity before authentication and typically do not maintain an up-to-date local cache of issuer certificates).

4.2.6. Raw Public Keys

[RFC7250] defines a new certificate type and TLS extensions to enable the use of raw public keys for authentication. Raw public keys use only a subset of information found in typical certificates and are therefore much smaller in size. However, raw public keys require an out-of-band mechanism to bind the public key with the entity presenting the key. Using raw public keys will obviously avoid the fragmentation problems resulting from large certificates and long

certificate chains. Deployments can consider their use as long as an appropriate out-of-band mechanism for binding public keys with identifiers is in place. Naturally, deployments will also need to consider the challenges of revocation and key rotation with the use of raw public keys.

4.2.7. New Certificate Types and Compression Algorithms

There is ongoing work to specify new certificate types which are smaller than traditional X.509 certificates. For example, [I-D.mattsson-cose-cbor-cert-compress] defines a Concise Binary Object Representation (CBOR) [RFC7049] encoding of X.509 Certificates. The CBOR encoding can be used to compress existing X.509 certificate or for natively signed CBOR certificates. [I-D.tschofenig-tls-cwt] registers a new TLS Certificate type which would enable TLS implementations to use CBOR Web Tokens (CWTs) [RFC8392] as certificates. While these are early initiatives, future EAP-TLS deployments can consider the use of these new certificate types and compression algorithms to avoid large message sizes.

4.3. Updating Authenticators

There are several legitimate reasons that authenticators may want to limit the number of round-trips/packets/octetets that can be sent. The main reason has been to work around issues where the EAP peer and EAP server end up in an infinite loop ACKing their messages. Another reason is that unlimited communication from an unauthenticated device using EAP could provide a channel for inappropriate bulk data transfer. A third reason is to prevent denial-of-service attacks.

Updating the millions of already deployed access points and switches is in many cases not realistic. Vendors may be out of business or no longer supporting the products and administrators may have lost the login information to the devices. For practical purposes the EAP infrastructure is ossified for the time being.

Vendors making new authenticators should consider increasing the number of round-trips allowed to 100 before denying the EAP authentication to complete. Based on the size of the certificates and certificate chains currently deployed, such an increase would likely ensure that peers and servers can complete EAP-TLS authentication. At the same time, administrators responsible for EAP deployments should ensure that this 100 roundtrip limit is not exceeded in practice.

5. IANA Considerations

This document includes no request to IANA.

6. Security Considerations

Updating implementations to TLS version 1.3 allows omitting all certificates with a trust anchor known by the other endpoint. TLS 1.3 additionally provides improved security, privacy, and reduced latency for EAP-TLS [I-D.ietf-emu-eap-tls13].

Security considerations when compressing certificates are specified in [I-D.ietf-tls-certificate-compression].

Specific security considerations of the referenced documents apply when they are taken into use.

7. References

7.1. Normative References

- [I-D.ietf-emu-eap-tls13]
Mattsson, J. and M. Sethi, "Using EAP-TLS with TLS 1.3", draft-ietf-emu-eap-tls13-12 (work in progress), November 2020.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

7.2. Informative References

- [I-D.ietf-tls-certificate-compression]
Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", draft-ietf-tls-certificate-compression-10 (work in progress), January 2020.
- [I-D.ietf-tls-ctls]
Rescorla, E., Barnes, R., and H. Tschofenig, "Compact TLS 1.3", draft-ietf-tls-ctls-01 (work in progress), November 2020.
- [I-D.mattsson-cose-cbor-cert-compress]
Raza, S., Hoglund, J., Selander, G., Mattsson, J., and M. Furuheid, "CBOR Encoding of X.509 Certificates (CBOR Certificates)", draft-mattsson-cose-cbor-cert-compress-03 (work in progress), November 2020.
- [I-D.thomson-tls-sic]
Thomson, M., "Suppressing Intermediate Certificates in TLS", draft-thomson-tls-sic-00 (work in progress), March 2019.

- [I-D.tschofenig-tls-cwt]
Tschofenig, H. and M. Brossard, "Using CBOR Web Tokens (CWTs) in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", draft-tschofenig-tls-cwt-02 (work in progress), July 2020.
- [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-2010 , February 2010.
- [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>>.
- [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>>.
- [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>>.
- [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <<https://www.rfc-editor.org/info/rfc6090>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <<https://www.rfc-editor.org/info/rfc7924>>.

- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.

Acknowledgements

This draft is a result of several useful discussions with Alan DeKok, Bernard Aboba, Jari Arkko, Jouni Malinen, Darshak Thakore, and Hannes Tschofenig.

Authors' Addresses

Mohit Sethi
Ericsson
Jorvas 02420
Finland

Email: mohit@piuha.net

John Mattsson
Ericsson
Kista
Sweden

Email: john.mattsson@ericsson.com

Sean Turner
sn3rd

Email: sean@sn3rd.com

Network Working Group
Internet-Draft
Updates: RFC7170 (if approved)
Intended status: Standards Track
Expires: May 4, 2020

E. Lear
O. Friel
N. Cam-Winget
Cisco Systems
D. Harkins
HP Enterprise
November 01, 2019

TEAP Update and Extensions for Bootstrapping
draft-lear-eap-teap-brski-05

Abstract

In certain environments, in order for a device to establish any layer three communications, it is necessary for that device to be properly credentialed. This is a relatively easy problem to solve when a device is associated with a human being and has both input and display functions. It is less easy when the human, input, and display functions are not present. To address this case, this memo specifies extensions to the Tunnel Extensible Authentication Protocol (TEAP) method that leverages Bootstrapping Remote Secure Key Infrastructures (BRSKI) in order to provide a credential to a device at layer two. The basis of this work is that a manufacturer will introduce the device and the local deployment through cryptographic means. In this sense the same trust model as BRSKI is used.

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 May 4, 2020.

Copyright Notice

Copyright (c) 2019 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. Terminology	3
2. TEAP BRSKI Architecture	4
3. BRSKI Bootstrap and Enroll Operation	4
3.1. Discovery of Trusted MASA	5
3.2. Executing BRSKI in a TEAP Tunnel	5
4. PKI Certificate Considerations	9
4.1. TEAP Tunnel Establishment	9
4.2. BRSKI Trust Establishment	11
4.3. Certificate Expiration Times	12
4.4. LDevID Subject and Subject Alternative Names	12
4.5. PKCS#10 Retry Handling	13
5. Peer Identity	13
6. Channel and Crypto Binding	14
7. Protocol Flows	14
7.1. TEAP Server Grants Access	14
7.2. TEAP Server Instructs Client to Perform BRSKI Flow	16
7.3. TEAP Server Instructs Client to Reenroll	20
7.4. Out of Band Reenroll	22
8. TEAP TLV Formats	22
8.1. New TLVs	22
8.1.1. BRSKI-RequestVoucher TLV	23
8.1.2. BRSKI-Voucher TLV	23
8.1.3. CSR-Attributes TLV	24
8.1.4. Retry-After TLV	25
8.1.5. NAI TLV	25
8.2. Existing TEAP TLV Specifications	26
8.2.1. PKCS#10 TLV	26
8.3. TLV Rules	26
9. Fragmentation	27
10. IANA Considerations	27

11. Security Considerations	27
11.1. Issues with Provisionally Authenticated TEAP	28
11.2. Attack Against Discovery	28
11.3. TEAP Server as Registration Authority	29
11.4. Trust of Registrar	29
12. Acknowledgments	29
13. References	29
13.1. Normative References	29
13.2. Informative References	30
Appendix A. Changes from Earlier Versions	30
Authors' Addresses	31

1. Introduction

[I-D.ietf-anima-bootstrapping-keyinfra] (BRSKI) specifies a means to provision credentials to be used as credentials to operationally access networks. It was designed as a standalone means where some limited access to an IP network is already available. This is not always the case. For example, IEEE 802.11 networks generally require authentication prior to any form of address assignment. While it is possible to assign an IP address to a device on some form of an open network, or to accept some sort of default credential to establish initial IP connectivity, the steps that would then follow might well require that the device is placed on a new network, requiring resetting all layer three parameters.

A more natural approach in such cases is to more tightly bind the provisioning of credentials with the authentication mechanism. One such way to do this is to make use of the Extensible Authentication Protocol (EAP) [RFC3748] and the Tunnel Extensible Authentication Protocol (TEAP) method [RFC7170]. Thus we define new TEAP Type-Length-Value (TLV) objects that can be used to transport the BRSKI protocol messages within the context of a TEAP TLS tunnel.

[RFC7170] discusses the notion of provisioning peers. Several different mechanisms are available. Section 3.8 of that document acknowledges the concept of not initially authenticating the outer TLS session so that provisioning may occur. In addition, exchange of multiple TLV messages between client and EAP server permits multiple provisioning steps.

1.1. Terminology

The reader is presumed to be familiar with EAP terminology as stated in [RFC3748]. In addition, the following terms are commonly used in this document.

- o BRSKI: Bootstrapping Remote Secure Key Infrastructures, as defined in [I-D.ietf-anima-bootstrapping-keyinfra]. The term is also used to refer to the flow described in that document.
- o EST: Enrollment over Secure Transport, as defined in [RFC7030].
- o Voucher: a signed JSON object as defined in [RFC8366].

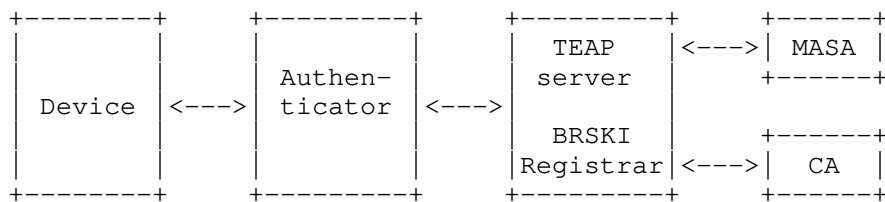
2. TEAP BRSKI Architecture

The TEAP BRSKI architecture is illustrated in Section 3. The device talks to the TEAP server via the Authenticator using any compliant transport such as [IEEE8021X]. The architecture illustrated shows an Authenticator distinct from the TEAP server. This is a deployment optimization and when so deployed the communication between Authenticator and TEAP server is a AAA protocol such as RADIUS or DIAMETER.

The architecture illustrated shows a co-located TEAP server and BRSKI registrar. Not only are these two functions co-located, they MUST be the same entity. This ensures that the entity identified in the device's voucher request (the TEAP server) is the same entity that signs the voucher request (the registrar).

The registrar communicates with the BRSKI MASA service for the purposes of getting signed vouchers.

The registrar also communicates with a Certificate Authority in order to issue LDevIDs. The architecture shows the registrar and CA as being two logically separate entities, however the CA may be integrated into the registrar. The device is not explicitly aware of whether the CA and registrar functions are integrated.



3. BRSKI Bootstrap and Enroll Operation

This section summarises the current BRSKI operation. The BRSKI flow assumes the device has an IDevID and has a manufacturer installed trust anchor that can be used to validate the BRSKI voucher. The

BRSKI flow comprises several main steps from the perspective of the device:

- o Step 1: Device discovers the registrar
- o Step 2: Device establishes provisional TLS connection to registrar
- o Step 3: Device sends voucher request message and receives signed voucher response
- o Step 4: Device validates voucher and validates provisional TLS connection to registrar
- o Step 5: Device downloads additional local domain CA information
- o Step 6: Device downloads Certificate Signing Request (CSR) attributes
- o Step 7: Device does a certificate enroll to obtain an LDevID
- o Step 8: Device periodically reenrolls via EST to refresh its LDevID

Most of the operational steps require the device, and thus its internal state machine, to automatically complete the next step without being explicitly instructed to do so by the registrar. For example, the registrar does not explicitly tell the device to download additional local domain CA information, or to do an EST enroll to obtain an LDevID.

3.1. Discovery of Trusted MASA

BRSKI section 2.8 outlines how the Registrar discovers the correct MASA to connect with. BRSKI section 5.3 outlines how the Registrar can make policy decisions about which devices to trust.

Similar approaches are applicable for TEAP servers executing BRSKI. For example, the TEAP server may be configured with a list of trusted manufacturing CAs. During device bootstrap, only devices with an IDevID signed by a trusted manufacturing CA may be allowed to establish a TLS connection with the TEAP server, and the TEAP server could then extract the MASA URI from the device's IDevID.

3.2. Executing BRSKI in a TEAP Tunnel

This section outlines how the main BRSKI steps outlined above map to TEAP, and how BRSKI and enrollment can be accomplished inside a TEAP TLS tunnel. The following new TEAP TLVs are introduced:

- o BRSKI-VoucherRequest
- o BRSKI-Voucher
- o CSR-Attributes

The following steps outline how the above BRSKI flow maps to TEAP.

- o Step 1: Device discovers the registrar

When BRSKI is executed in a TEAP tunnel, the device exchanges BRSKI TLVs with the TEAP server. The discovery process for devices is therefore the standard wired or wireless LAN EAP server discovery process. The discovery processes outlined in section 4 of [I-D.ietf-anima-bootstrapping-keyinfra] are not required for initial discovery of the registrar.

- o Step 2: Device establishes provisional TLS connection to registrar

The device establishes an outer TEAP tunnel with the TEAP server and does not validate the server certificate. The device presents its LDevID as its identity certificate if it has a valid LDevID, otherwise it presents its IDevID. The TEAP server validates the device's certificate using its implicit or explicit trust anchor database. If the device presents an IDevID it is verified against a database of trusted manufacturer certificates. Server policy may also be used to control which certificate the device is allowed present, as described in section {pki-certificate-authority-considerations}.

If the presented credential is sufficient to grant access, the TEAP server can return a TEAP Result TLV indicating success immediately. The device may still send a Request-Action TLV including a BRSKI-VoucherRequest TLV in response to the TEAP Result TLV if it does not have, but requires, provisioning of trust anchors for validating the TEAP server certificate. Note that no inner EAP method is required for this, only an exchange of TEAP TLVs.

[todo] Question: as the device wants the server to reply with a BRSKI-Voucher TLV, does it really send a Request-Action TLV containing a BRSKI-VoucherRequest TLV, or does it send a Request-Action TLV containing a BRSKI-Voucher TLV?? The TEAP draft is a bit ambiguous here. Normally, if one end sends a Request-Action including XXX-TLV, it means it wants the far end to send an XXX-TLV...

[todo] Question: general TEAP protocol question: does the device have to send a Request-Action w/BRSKI-VoucherRequest or can it send a BRSKI-VoucherRequest on its own? I'm not clear on this.

If the TEAP server requires that the device execute a BRSKI flow, the server sends a Request-Action TLV that includes a BRSKI-VoucherRequest TLV. For example, if the device presented its IDevID but the TEAP server requires an LDevID.

[todo] Question: to nit pick, the server should send a Request-Action TLV including a PKCS#10 TLV to tell the client to enroll. How does the server really know that the client has the correct trust established (as previously received by a BRSKI-Voucher)? If the client sends an IDevID, does server always send a Request-Action including both BRSKI-VoucherRequest and PKCS#10 TLVs? Whats the client behaviour? I assume client can spontaneously send BRSKI-VoucherRequest and/or PCSK#10 without being explicitly instructed to. Just need to get the language correct here.

The TEAP server may also require the device to reenroll, for example, if the device presented a valid LDevID that is very closed to expiration. The server may instruct a device to reenroll by sending a Request-Action TLV that includes a zero byte length PKCS#10 TLV.

- o Step 3: Device sends voucher request message and receives signed voucher response

The device sends a BRSKI-RequestVoucher TLV to the TEAP server. The TEAP server forwards the RequestVoucher message to the MASA server, and the MASA server replies with a signed voucher. The TEAP server sends a BRSKI-Voucher TLV to the device.

If the MASA server does not issue a signed voucher, the TEAP server sends an EAP-Error TLV with a suitable error code to the device.

For wireless devices in particular, it is important that the MASA server only return a voucher for devices known to be associated with a particular registrar. In this sense, success indicates that the device is on the correct network, while failure indicates the device should try to provision itself within wireless networks (e.g, go to the next SSID).

- o Step 4: Device validates voucher and validates provisional TLS connection to registrar

The device validates the signed voucher using its manufacturer installed trust anchor, and uses the CA information in the voucher to validate the TLS connection to the TEAP server.

If the device fails to validate the voucher, then it sends a TEAP-Error TLV indicating failiure to the TEAP server.

Similarly, if the device validates the voucher, but fails to validate the provisional TLS connection, then it sends a TEAP-Error TLV indicating failure to the TEAP server. Note that the outer TLS tunnel has already been established, thus allowing the client to send a TEAP-Error TLV to the server inside that tunnel to indicate that it failed to verify the provisionally accepted outer TLS tunnel server identity.

- o Step 5: Device downloads additional local domain CA information

On completion of the BRSKI flow, the device SHOULD send a Trusted-Server-Root TLV to the TEAP server in order to discover additional local domain CAs. This is equivalent to section [todo] from [I-D.ietf-anima-bootstrapping-keyinfra].

- o Step 6: Device downloads CSR attributes

No later than the completion of step 5, server MUST send a CSR-Attributes TLV to peer server in order to discover the correct fields to include when it enrolls to get an LDevID.

- o Step 7: Device does a certificate enroll to obtain an LDevID

When executing the BRSKI flow inside a TEAP tunnel, the device does not directly leverage EST when doing its initial enroll. Instead, the device uses the existing TEAP PKCS#10 and PCKS#7 TEAP mechanisms.

Once the BRSKI flow is complete, the device can now send a PKCS#10 TLV to enroll and request an LDevID. If the TEAP server instructed the device to start the BRSKI flow via a Request-Action TLV that includes a BRSKI-RequestVoucher TLV, then the device MUST send a PKCS#10 in order to start the enroll process. The TEAP server will handle the PKCS#10 and ultimately return a PKCS#7 including an LDevID to the device.

If the TEAP server granted the device access on completion of the outer TEAP TLS tunnel in step 2 without sending a Request-Action TLV, the device does not have to send a PKCS#10 to enroll.

At this point, the device is said to be provisioned for local network access, and may authenticate in the future via 802.1X with its newly acquired credentials.

- o Step 8: Device periodically reenrolls to refresh its LDevID

When a device's LDevID is close to expiration, there are two options for re-enrollment in order to obtain a fresh LDevID. As outlined in Step 2 above, the TEAP server may instruct the device to reenroll by sending a Request-Action TLV including a PKCS#10 TLV. If the TEAP server explicitly instructs the device to reenroll via these TLV exchange, then the device MUST send a PKCS#10 to reenroll and request a fresh LDevID.

However, the device SHOULD reenroll if it determines that its LDevID is close to expiration without waiting for explicit instruction from the TEAP server. There are two options to do this.

Option 1: The device reenrolls for a new LDevID directly with the EST CA outside the context of the 802.1X TEAP flow. The device uses the registrar discovery mechanisms outlined in [I-D.ietf-anima-bootstrapping-keyinfra] to discover the registrar and the device sends the EST reenroll messages to the discovered registrar endpoint. No new TEAP TLVs are defined to facilitate discover of the registrar or EST endpoints inside the context of the TEAP tunnel.

Option 2: When the device is performing a periodic 802.1X authentication using its current LDevID, it reenrolls for a new LDevID by sending a PKCS#10 TLV inside the TEAP TLS tunnel.

4. PKI Certificate Considerations

There are multiple noteworthy PKI certificate handling considerations. These include:

- o PKI CA handling when establishing the TEAP tunnel
- o PKI CA handling establishing trust using BRSKI
- o IDevID and LDevID expiration times
- o Specifying LDevID Subject and Subject Alternative Names
- o PKCS#10 retry handling

These are described in more detail here.

4.1. TEAP Tunnel Establishment

Because this method establishes a client identity, if the peer has not been previously bootstrapped, or otherwise cannot successfully authenticate, it will use a generic identity string of teap-bootstrap@TBD1 as its network access identifier (NAI).

BRSKI section 5.3 outlines the policy decisions a Registrar may make when deciding whether to accept connections from clients. Similarly, the TEAP server operator may configure a set of trusted CAs for validating incoming TLS connections from clients. The operator may want to 'allow any device from a specific vendor', or from a set of vendors, to access the network. Network operators may do this by restricting network access to clients that have a certificate signed by one of a small set of trusted manufacturer/supplier CAs.

When the client sends its ClientHello to initiate TLS tunnel establishment, it is possible for the TEAP server to restrict the certificates that the client can use for tunnel establishment by including a list of CA distinguished names in the `certificate_authorities` field in the CertificateRequest message. The client should only continue with the handshake if it has a certificate signed by one of the indicated CAs.

In practice, network operators will likely want to onboard devices from a large number of device manufacturers, with each manufacturer using a different root CA when issuing IDevIDs. If the number of different manufacturer root CAs is large, this could result in very large TLS handshake messages. Therefore, the TEAP server may send a CertificateRequest message and not specify any `certificate_authorities`, thus allowing the client present a certificate signed by any authority in its Certificate message.

If the client has both an IDevID and an LDevID, the client should present the LDevID in preference to its IDevID, if allowed by server policy.

Once the client has sent its TLS Finished message, the TEAP server can make a policy decision, based on the CA used to sign the client's certificate, on whether to establish the outer TLS tunnel or not.

The TEAP server may delegate policy decisions to the MASA or CA function. For example, the TEAP server may declare EAP success and grant network access if the client presents a valid LDevID signed by a trusted domain CA. However, if the client presents an IDevID signed by a trusted manufacturer CA, the TEAP server may establish the TLS tunnel but not declare EAP success and grant network access until the client successfully completes a BRSKI Voucher exchange and PKCS#10/PKCS#7 exchange inside that tunnel.

It is recommended that the client validate the certificate presented by the server in the server's Certificate message, but this may not be possible for clients that have not yet provisioned appropriate trust anchors. If the client is in the provisioning phase and has not yet completed a BRSKI flow, it will not have trust anchors

installed yet, and thus will not be able to validate the server's certificate. The client must however note the certificate presented by the server for (i) inclusion in the BRSKI-RequestVoucher TLV and for (ii) validation once the client has discovered the local domain trust anchors.

If the client does not present a suitable certificate to the server, the server MUST terminate the connection and fail the EAP request. If the TEAP server is unable to validate the client's certificate using its implicit or explicit trust anchor database it MUST fail the EAP request.

On establishment of the outer TLS tunnel, the TEAP server will make a policy decision on next steps. Possible policy decisions include:

- o Option 1: Server grants client full network access and returns EAP-Success. This will typically happen when the client presents a valid LDevID. Network policy may grant client network access based on IDevID without requiring the device to enroll to obtain an LDevID.
- o Option 2: Server requires that client perform a full BRSKI flow, and then enroll to get an LDevID. This will typically happen when the client presents a valid IDevID and network policy requires all clients to have LDevIDs. The server sends a Request-Action TLV that includes a BRSKI-RequestVoucher TLV to the client to instruct it to start the BRSKI flow.
- o Option 3: Server requires that the client reenroll to obtain a new LDevID. This could happen when the client presents a valid LDevID that is very close to expiration time, or the server's policy requires an LDevID update. The server sends an Action-Request TLV including a PKCS#10 TLV to the client to instruct it to reenroll.

4.2. BRSKI Trust Establishment

If the server requires that client perform a full BRSKI flow, it sends a Request-Action TLV that includes a zero byte length BRSKI-RequestVoucher TLV to the client. The client sends a new BRSKI-RequestVoucher TLV to the server, which contains all data specified in [I-D.ietf-anima-bootstrapping-keyinfra] section 5.2. The client includes the server certificate it received in the server's Certificate message during outer TLS tunnel establishment in the proximity-registrar-cert field. The client signs the request using its IDevID.

The server includes all additional information as required by [I-D.ietf-anima-bootstrapping-keyinfra] section 5.4 and signs the request prior to forwarding to the MASA.

The MASA responds as per [I-D.ietf-anima-bootstrapping-keyinfra] section 5.5. The response may indicate failure and the server should react accordingly to failures by sending a failure response to the client, and failing the TEAP method.

If the MASA replies with a signed voucher and a successful result, the server then forwards this response to the client in a BRSKI-Voucher TLV.

When the client receives the signed voucher, it validates the signature using its built in trust anchor list, and extracts the pinned-domain-cert field. The client must use the CA included in the pinned-domain-cert to validate the certificate that was presented by the server when establishing the outer TLS tunnel. If this certificate validation fails, the client must fail the TEAP request and not connect to the network.

[TBD- based on client responses, the registrar sends a status update to the MASA]

4.3. Certificate Expiration Times

[IEEE8021AR] section 7.2.7.2 states:

notAfter: The latest time a DevID is expected to be used. Devices possessing an IDevID are expected to operate indefinitely into the future and should use the value 99991231235959Z. Solutions verifying an IDevID are expected to accept this value indefinitely.

TEAP servers SHOULD follow the 802.1AR standard when validating IDevIDs.

TEAP servers SHOULD reject LDevIDs with expired certificates and SHOULD NOT allow clients to connect with recently expired LDevIDs. If a client presents a recently expired LDevID it SHOULD be forced to authenticate using its IDevID and then reenroll to obtain a valid LDevID.

4.4. LDevID Subject and Subject Alternative Names

BRSKI section 5.9.2 specifies that the pledge MUST send a CSR Attributes request to the registrar. The registrar MAY use this mechanism to instruct the pledge about the identities it should

include in the CSR request it sends as part of enrollment. The registrar may use this mechanism to tell the pledge what Subject or Subject Alternative Name identity information to include in its CSR request. This can be useful if the Subject must have a specific value in order to complete enrollment with the CA.

4.5. PKCS#10 Retry Handling

They will be scenarios where the TEAP server is willing to handle a PKCS#10 request from a client and issue a certificate via a PKCS#7 response, however, the TEAP server is unable to immediately completely the request and needs to instruct the client to retry later after a specified time interval.

A new Retry-After TLV is defined that the TEAP server uses to specify a retry interval in seconds. New error codes are defined to handle these two alternate retry scenarios.

- o The TEAP tunnel remains up: The client is instructed to resend the PKCS#10 request after a retry interval but inside the same TEAP tunnel. The TEAP server returns a Retry-After TLV to the client, and returns an Error TLV with a new code in the 1000-1999 range.
- o The TEAP tunnel is torn down: The client is instructed to establish a new TEAP connection and TEAP tunnel after a retry interval, and resend the PKCS#10 request inside the new tunnel. The TEAP server returns a Retry-After TLV to the client, and returns an Error TLV with a new code in the 2000-2999 range.

5. Peer Identity

EAP [RFC3748] recommends that "the Identity Response be used primarily for routing purposes and selecting which EAP method to use". NAI [RFC7542] recommends omitting the username part of an NAI in order to support username privacy, where appropriate.

A device that has not been bootstrapped at all SHOULD send an identity of teap-bootstrap@TBD1. Otherwise, a device SHOULD send its configured NAI.

The TEAP server may specify an NAI that it wishes the device to use. For example, the server may want a bootstrapped device to use an NAI of "abc123@example.com", or simply an NAI of "@example.com". This could be desirable in order to facilitate roaming scenarios. The server can do this by sending the device an NAI TLV inside the TEAP tunnel.

If the server specifies an NAI TLV, and the device handles the TLV, the device MAY use the specified NAI in all subsequent EAP authentication flows. If the device is not willing to handle the NAI TLV, it MUST reply with an Error TLV.

Authentication servers implementing this specification MAY reply with an Error TLV to any unrecognized NAI, or MAY attempt to bootstrap the device, regardless of the NAI. A device receiving an Error from the server MAY attempt a new session without the NAI in order to bootstrap.

6. Channel and Crypto Binding

As the TEAP BRSKI flow does not define or require an inner EAP method, there is no explicit need for exchange of Channel-Binding TLVs between the device and the TEAP server.

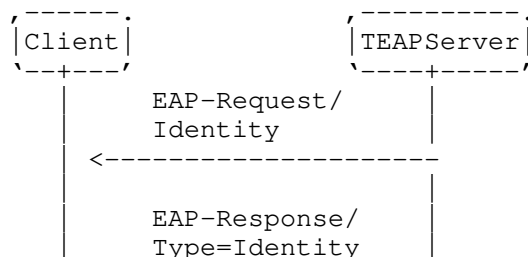
The TEAP BRSKI TLVs are expected to occur at the beginning of the TEAP Phase 2 and MUST occur before the final Crypto-Binding TLV. This draft does not exclude the possibility of having other EAP methods occur following the TEAP BRSKI TLVs and as such, the Crypto-Binding TLV process rules as defined in [RFC7170] apply.

7. Protocol Flows

This section outlines protocol flows that map to the three server policy options described in section Section 4.1. The protocol flows illustrate a TLS1.2 exchange. Pertinent notes are outlined in the protocol flows.

7.1. TEAP Server Grants Access

In this flow, the server grants access as server policy allows the client to access the network based on the identity certificate that the client presented. This means that either (i) the client has previously completed BRSKI and has presented a valid LDevID or (ii) the client presents an IDevID and network policy allows access based purely on IDevID.



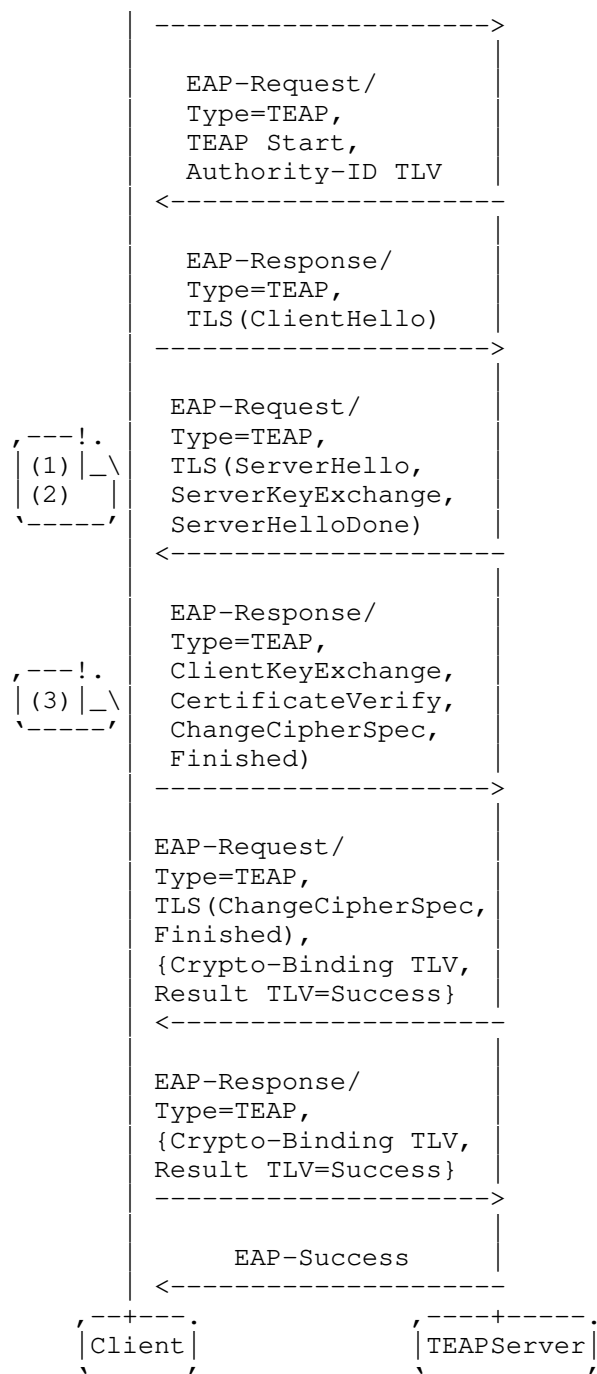


Figure 1: TEAP Server Grants Access

Notes:

(1) If the client has completed the BRSKI flow and has locally significant trust anchors, it must validate the Certificate received from the server. If the client has not yet completed the BRSKI flow, then it provisionally accepts the server Certificate and must validate it later once BRSKI is complete.

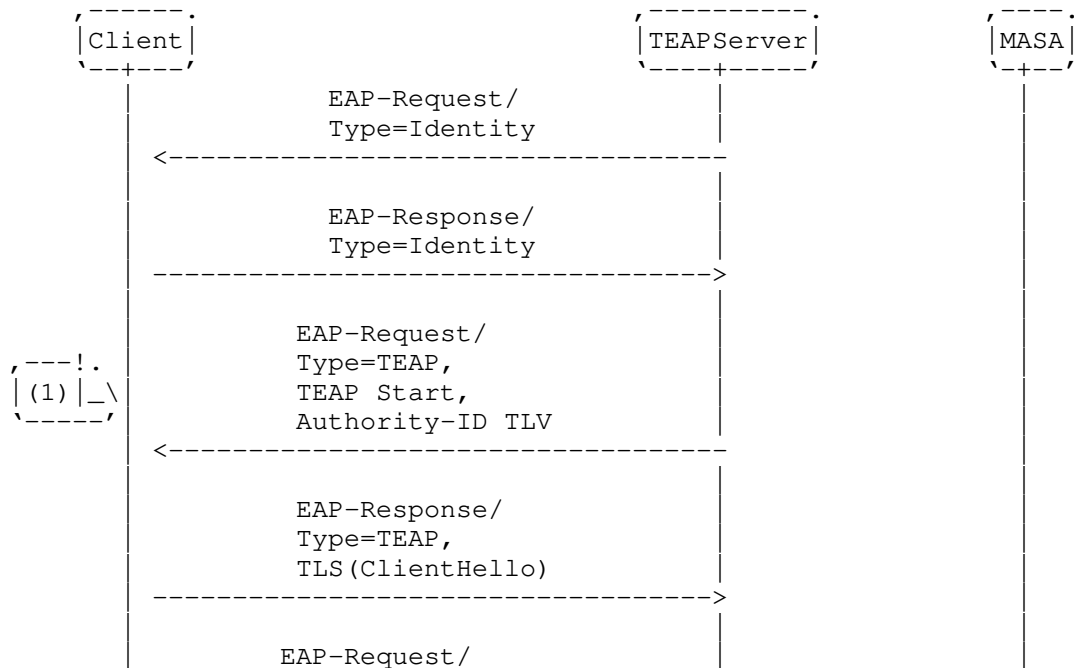
(2) The server may include certificate_authorities field in the CertificateRequest message in order to restrict the identity certificates that the device is allowed present.

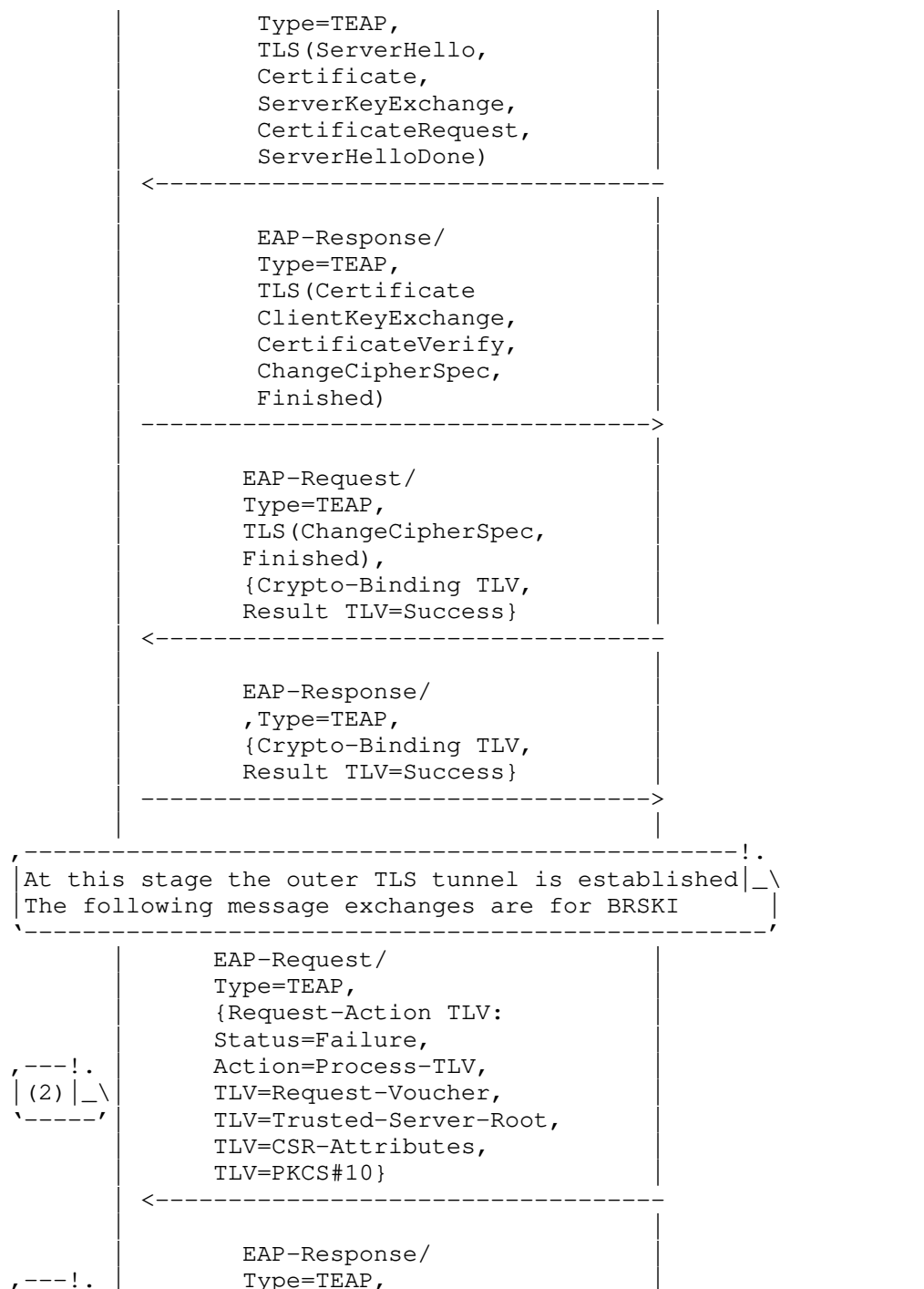
(3) The device will present its LDevID, if it has one, in preference to its IDevID, if allowed by server policy.

7.2. TEAP Server Instructs Client to Perform BRSKI Flow

In this two part flow, the server instructs the client to perform a BRSKI flow by exchanging TLVs once the outer TLS tunnel is established. After that, enrollment takes place.

In the first part of the flow, the MASA is depicted on the right.





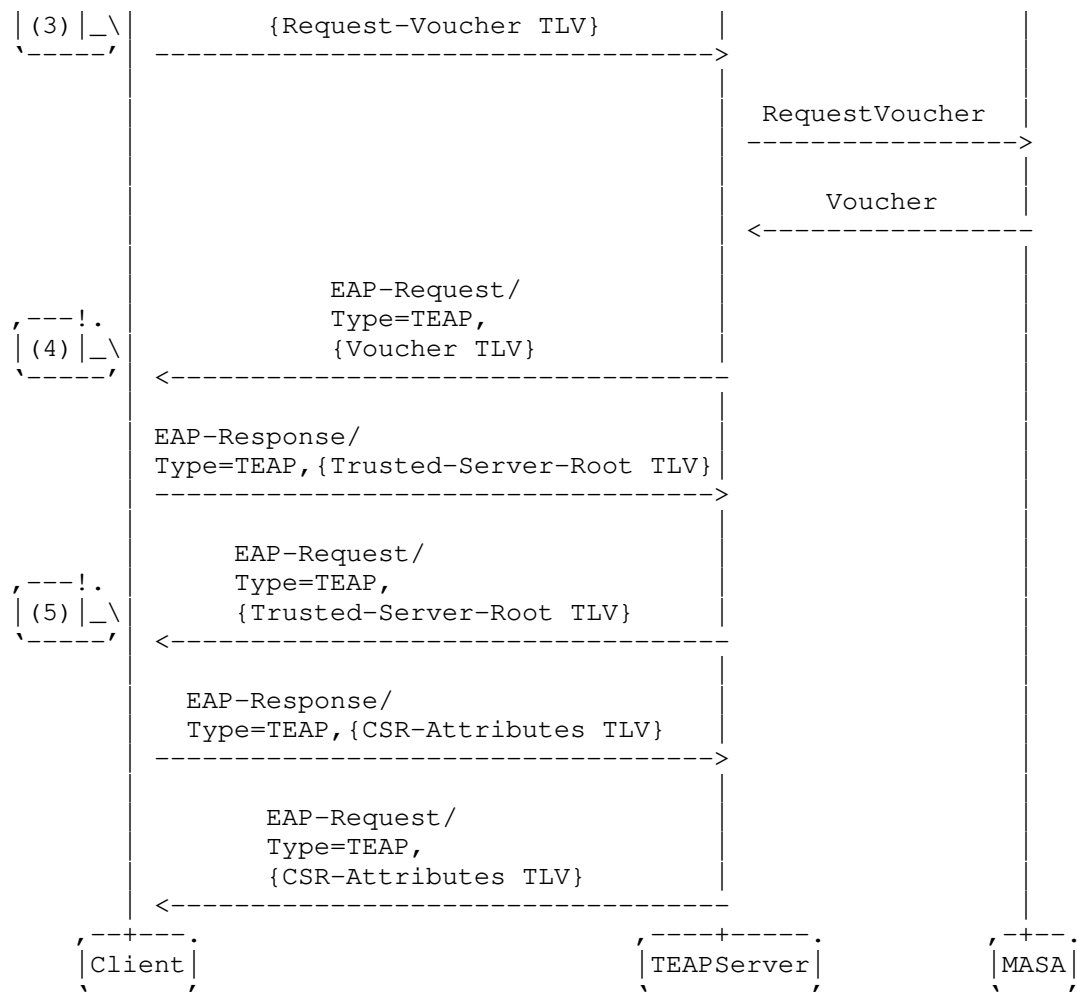


Figure 2: TEAP Server Instructs Client to Perform BRSKI Flow

The second part of the flow depicts the CA on the right.

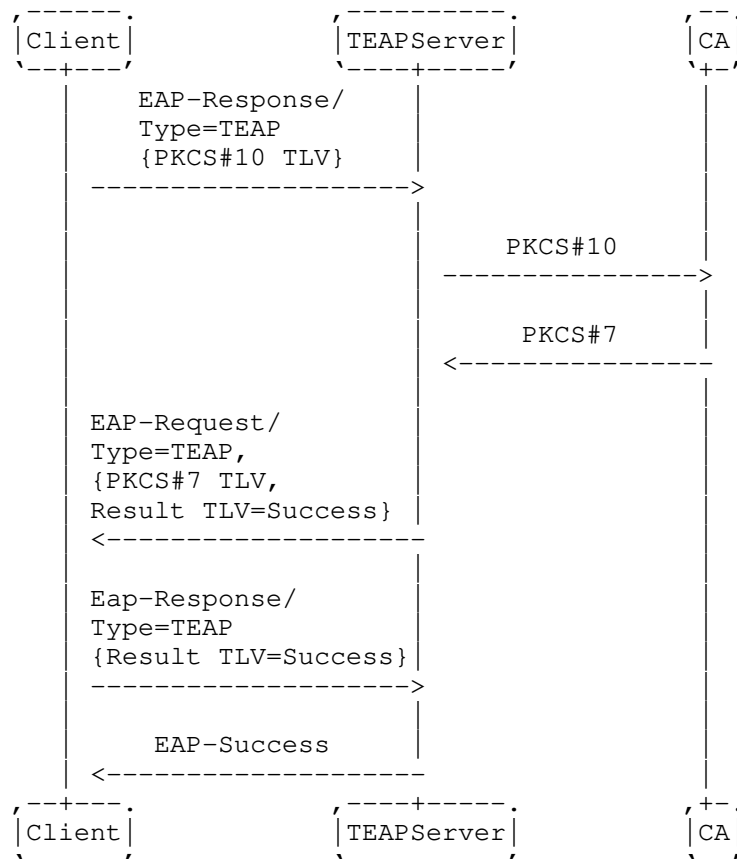


Figure 3: Enrollment after BRSKI Flow

Notes:

(1) If the client has not yet completed the BRSKI flow, then it provisionally accepts the server certificate and must validate it later once BRSKI is complete. The server validates the client certificate using its trust anchor database.

(2) The server instructs the client to start the BRSKI flow by sending a Request-Action TLV that includes a BRSKI-RequestVoucher TLV. The server also instructs the client to request trust anchors, to request CSR Attributes, and to initiate a PKCS certificate enrolment. As outlined in [RFC7170], the Request-Action TLV is sent after the Crypto-Binding TLV and Result TLV exchange.

(3) The client includes the certificate it received from the server in the RequestVoucher message.

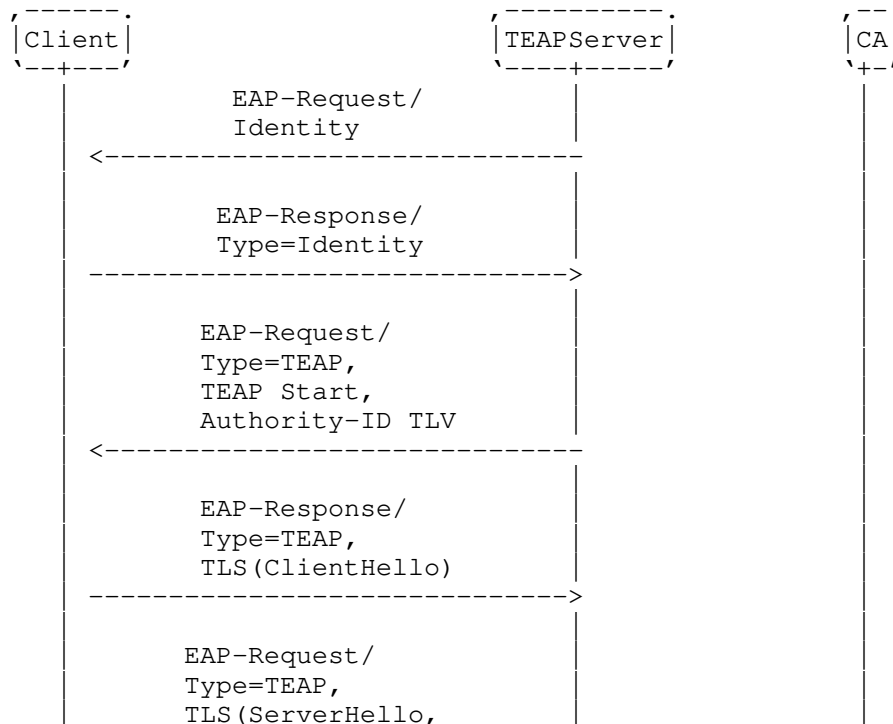
(4) Once the client receives and validates the voucher signed by the MASA, it must verify the certificate it previously received from the server.

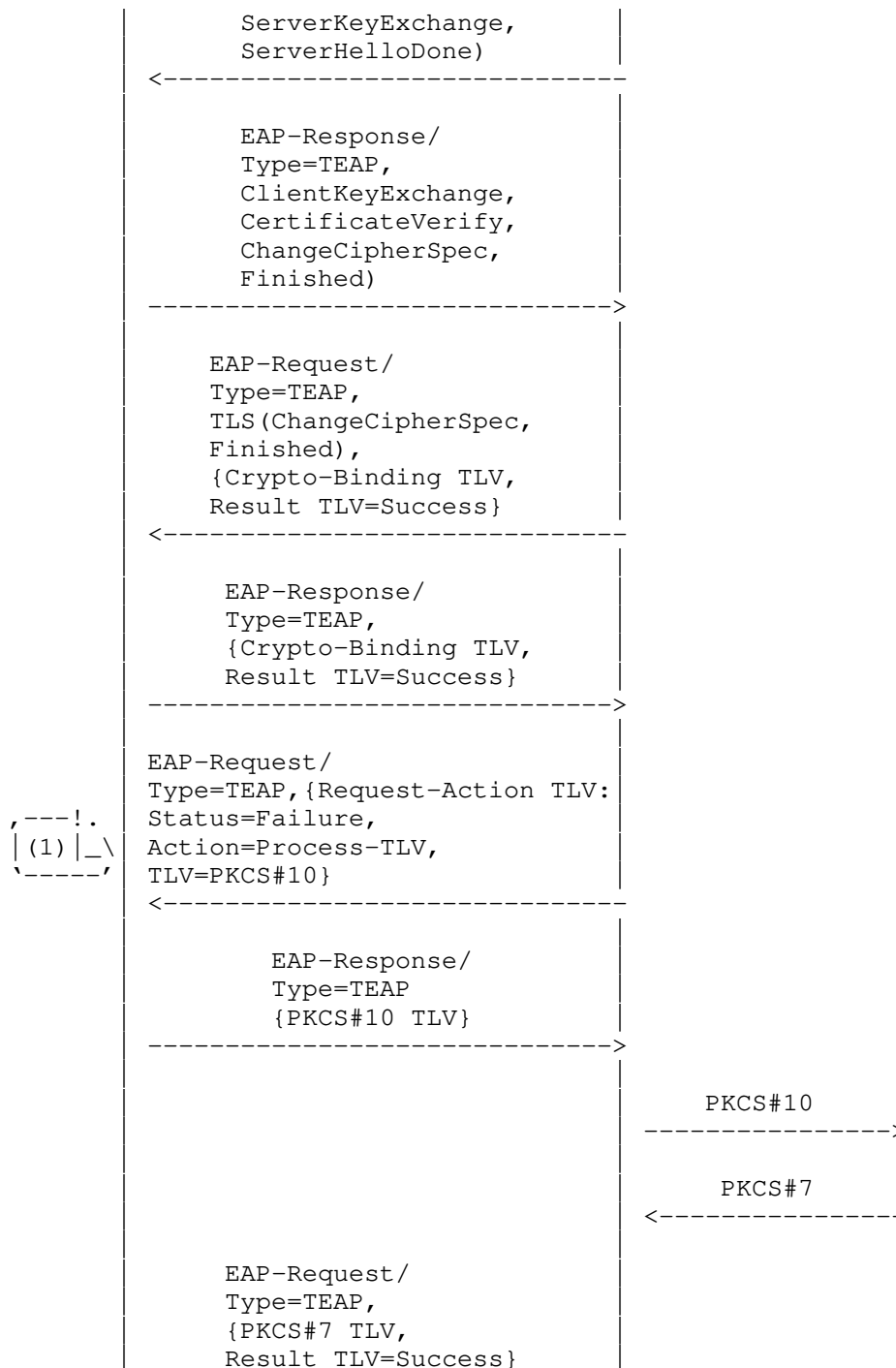
(5) As outlined in [RFC7170], the Trusted-Server-Root TLV is exchanged after the Crypto-Binding TLV exchange, and after the client has used the Voucher to authenticate the TEAP server identity. This is equivalent to section [todo] from [I-D.ietf-anima-bootstrapping-keyinfra].

(6) There is no need for an additional Crypto-Binding TLV exchange as there is no inner EAP method. All BRSKI exchanges are simply TLVs exchanged inside the outer TLS tunnel.

7.3. TEAP Server Instructs Client to Reenroll

In this flow, the server instructs the client to reenroll and get a new LDevID by exchanging TLVs once the outer TLS tunnel is established.





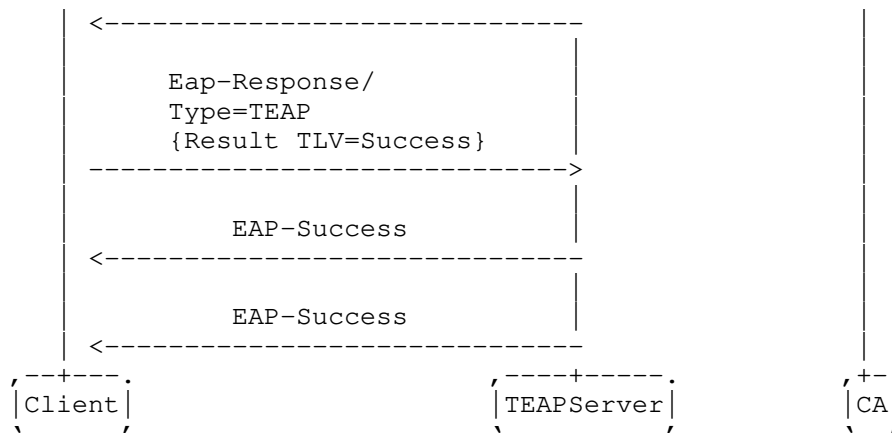


Figure 4: TEAP Server Instructs Client to Reenroll

(1) The server instructs the client to reenroll by sending a Request-Action TLV that includes a PKCS#10 TLV.

7.4. Out of Band Reenroll

This section shows how the device does a reenroll to refresh its LDevID directly against the registrar outside the context of the TEAP tunnel.

8. TEAP TLV Formats

8.1. New TLVs

This document defines 5 new TEAP TLVs. The following table indicates whether the TLVs can be included in Request messages from TEAP server to device, or Response messages from device to TEAP server.

TLV	Message
BRSKI-VoucherRequest	Response
BRSKI-Voucher	Request
CSR-Attributes	Response
Retry-After	Response
NAI-Identity	Request

These new TLVs are detailed in this section.

8.1.1.1. BRSKI-RequestVoucher TLV

This TLV is used by the server as part of a Request-Action TLV to request from the peer that it initiate a voucher request. When used in this fashion, the length of this TLV will be set to zero. The Status field of the Request-Action TLV MUST be set to Failure.

It is also used by the peer to initiate the voucher request. When used in this fashion, the length of the TLV will be set to that of the voucher request, as encoded and described in Section 3.3 in [I-D.ietf-anima-bootstrapping-keyinfra].

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD1-VoucherRequest | Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The M and R bits are always expected to be set to 0.

The server is expected to forward the voucher request to the MASA, and then return a voucher in a BRSKI-Voucher TLV as described below. If it is unable to do so, it returns an TEAP Error TLV with one of the defined errors or the following:

```

TBD2-MASA-Notavailable  MASA unavailable
TBD3-MASA-Refused      MASA refuses to sign the voucher

```

The peer terminates the TEAP connection, but may retry at some later point. The backoff mechanism for such retries should be appropriate for the device. Retries MUST occur no more frequently than once every two (TBD) minutes.

8.1.1.2. BRSKI-Voucher TLV

This TLV is transmitted from the server to the peer. It contains a signed voucher, as describe in [RFC8366].

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD4-Voucher | Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Upon receiving this TLV the peer will validate the signature of the voucher, using its pre-installed manufacturer trust anchor (LDevID). It MUST also validate the certificate used by the server to establish the TLS connection.

If successful, it installs the new trust anchor contained in the voucher.

Otherwise, the peer transmits an TEAP error TLV with one of the following error messages:

```

TBD5-Invalid-Signature  The signature on the voucher is invalid
TBD6-Invalid-Voucher    The form or content of the voucher is invalid
TBD7-Invalid-TLS-Signer The certificate used for the TLS connection
                        could not be validated.

```

8.1.3. CSR-Attributes TLV

The server SHALL transmit this TLV to the peer, either along with the BRSKI-Voucher TLV or at any time earlier in a communication. The peer shall include attributes required by the server in any following CSR. The value of this TLV is the base64 encoding described in Section 4.5.2 of [RFC7030].

The TEAP server MAY use this TLV to specify the subject identity information to include in Subject or Subject Alternate Name fields in any following CSR.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD8-CSR-Attributes | length |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Again, the M and R values are set to 0. In the case where the client is unable to provide the requested attributes, an TEAP-Error is returned as follows:

TBD9-CSR-Attribute-Fail Unable to supply the requested attributes.

8.1.4. Retry-After TLV

The server MUST transmit this TLV to the peer when replying to a PKCS#10 TLV request from the peer where the server is willing to fulfill the request and issue a certificate via a PKCS#7 response, but is unable to fulfill the request immediately. This TLV is used to tell the peer the minimum length of time it MUST wait before resending the PKCS#10 request. The value of this TLV is the time in seconds that the peer MUST wait before resending the PKCS#10 request. The peer MUST resend the exact same PKCS#10 request.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD10-Retry-After | length |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Again, the M and R values are set to 0.

8.1.5. NAI TLV

The server may use this TLV to provision a realm-specific NAI on the device.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD10-NAI | length |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

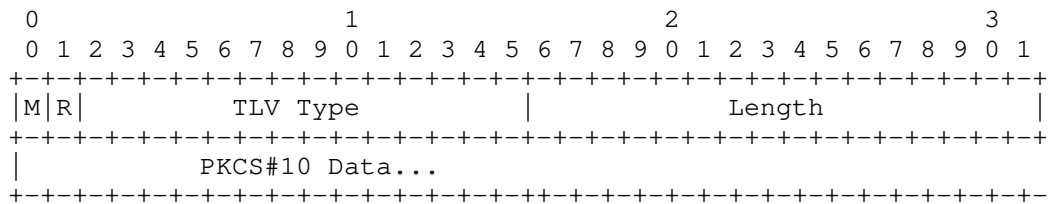
Again, the M and R values are set to 0.

8.2. Existing TEAP TLV Specifications

This section documents allowed usage of existing TEAP TLVs. The definition of the TLV is not changed, however clarifications on allowed values for the TLV fields is documented.

8.2.1. PKCS#10 TLV

[RFC7170] defines the PKCS#10 TLV as follows:



[RFC7170] does not explicitly allow a Length value of zero.

A Length value of zero is allowed for this TLV when the TEAP server sends a Request-Action TLV with a child PKCS#10 TLV to the client. In this scenario, there is no PKCS#10 Data included in the TLV. Clients MUST NOT send a zero length PKCS#10 TLV to the server.

8.3. TLV Rules

BRSKI TLVs can only be transported inside the TLS tunnel. The following table provides a guide to which TLVs may be encapsulated in which kind of packets, and in what quantity. The messages are as follows: Request is a TEAP Request, Response is a TEAP Response, Success is a message containing a successful Result TLV, and Failure is a message containing a failed Result TLV.

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

0 This TLV MUST NOT be present in the message.

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

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

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

Request	Response	Success	Failure	TLVs	0	0-1	0	0	BRSKI-VoucherRequest
0-1	0	0	0	BRSKI-Voucher	0	0-1	0	0	CSR-Attributes

9. Fragmentation

TEAP is expected to provide fragmentation support. Thus EAP-TEAP-BRSKI does not specifically provide any, as it is only expected to be used as an inner method to TEAP.

10. IANA Considerations

The IANA is requested to add entries into the following tables:

The following new TEAP TLVs are defined:

TBD1-VoucherRequest	Described in this document.
TBD4-Voucher	Described in this document.
TBD8-CSR-Attributes	Described in this document.
TBD10-Retry-After	Described in this document.

The following TEAP Error Codes are defined, with their meanings listed here and in previous sections:

TBD2-MASA-Notavailable	MASA unavailable
TBD3-MASA-Refused	MASA refuses to sign the voucher
TBD5-Invalid-Signature	The signature on the voucher is invalid
TBD6-Invalid-Voucher	The form or content of the voucher is invalid
TBD7-Invalid-TLS-Signer	The certificate used for the TLS connection could not be validated.
TBD9-CSR-Attribute-Fail	Unable to supply the requested attributes.
TBD11-Retry-PKCS#10	Retry PKCS#10 Request (1000 range code)
TBD12-Retry-PKCS#10	Retry PKCS#10 Request (2000 range code)
TBD13-NAI-Rejected	The device will not use the indicated NAI (1000 range code)

[[TODO: is there a registry of NAIs that map to TEAP methods? e.g. @eap-teap.net is reserved to indicate the peer wants to use TEAP method]]

11. Security Considerations

BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] provides a zero touch way for devices to enroll in a certification authority (CA). It assumes the device has IP connectivity. For networks that will not grant IP connectivity before authenticating (with a local credential) this poses a Catch-22- can't get on the network without a credential and can't get a credential without getting on the network.

This protocol provides a way for BRSKI to be in an EAP method which allows the BRSKI conversation to happen as part of EAP authentication and prior to obtaining IP connectivity.

The security considerations of [I-D.ietf-anima-bootstrapping-keyinfra] apply to this protocol. Running BRSKI through EAP introduces some additional areas of concern though.

11.1. Issues with Provisionally Authenticated TEAP

This protocol establishes an unauthenticated TLS connection and passes data through it. Provided that the only messages passed in this state are self-protected BRSKI messages this does not present a problem. Passing any other messages or TLVs prior to authentication of the provisional TLS connection could potentially introduce security issues.

While the TLS connection is unauthenticated, it must still be validated to the fullest extent possible. It is critical that the device and the TEAP server perform all steps in TLS- checking the validity of the presented certificate, validating the signature using the public key of the certificate, etc- except ensuring the trust of the presented certificate.

11.2. Attack Against Discovery

The device discovery technique specified in this protocol is the standard EAP server discovery process. Since it is trivial to set up an 802.11 wireless access point and advertise any network, an attacker can impersonate a legitimate wireless network and attract unprovisioned pledges. Given that an unprovisioned device will not know the legitimate network to connect to, it will probably attempt the first network it finds, making the attack that much easier. This allows for a "rogue registrar" to provision and take control of the device.

If the MASA verifies ownership prior to issuance of a voucher, this attack can be thwarted. But if the MASA is in reduced security mode and does not verify ownership this attack cannot be prevented. Registrars SHOULD use the audit log of a MASA when deploying newly purchased equipment in order to mitigate this attack.

Another way to mitigate this attack is through normal "rogue AP" detection and prevention.

11.3. TEAP Server as Registration Authority

If the TEAP server is logically separate from the Certification Authority (CA) (see Section 2) it will be acting as a Registration Authority (RA) when it obtains the PKCS#10 TLV and replies with a PKCS#7 TLV (see [RFC7170], Sections 4.2.16 and 4.2.17, respectively). The assurance a RA makes to a CA is that the public key in the presented CSR is bound to an authenticated identity in way that will assure non-repudiation.

To make such an assurance, the TEAP server MUST authenticate the provisional TLS connection with the device by validating the voucher response received from the MASA. In addition, it is RECOMMENDED that the TEAP server indicate that proof-of-possession (see [RFC7170], Section 3.8.2) is required by including the challengePassword OID in the CSR Attributes TLV.

11.4. Trust of Registrar

The device accepts a trusted server (CA) certificate and installs it in its trust anchor database during step 5 (see Section 3.2). This can happen only after the provisional TLS connection has been authenticated using the voucher and the Crypto-Binding TLV has been validated.

12. Acknowledgments

The authors would like to thank Brian Weis for his assistance, and Alan Dakok for improving language consistency. In addition, with ruthlessly "borrowed" the concept around NAI handling from Tuomas Aura and Mohit Sethi.

13. References

13.1. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M.,
and K. Watsen, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-29 (work in progress), October 2019.

[IEEE8021AR]
Institute for Electrical and Electronics Engineers,
"Secure Device Identity", 1998.

- [IEEE8021X] Institute for Electrical and Electronics Engineers, "IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control", 2010.
- [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>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.

13.2. Informative References

- [RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <<https://www.rfc-editor.org/info/rfc7542>>.

Appendix A. Changes from Earlier Versions

Draft -03: * Merge EAP server and Registrar * Security considerations
* References improvements * Add Dan Harkins as co-author

Draft -02: * Flow corrections

Draft -01: * Add packet descriptions, IANA considerations, smooth out language.

Draft -00:

- o Initial revision

Authors' Addresses

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Owen Friel
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
United States

Email: ofriel@cisco.com

Nancy Cam-Winget
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
United States

Email: ncamwing@cisco.com

Dan Harkins
HP Enterprise
3333 Scott Boulevard
Santa Clara, CA 95054
United States

Email: dharkins@arubanetworks.com

Network Working Group
Internet-Draft
Updates: RFC7170 (if approved)
Intended status: Standards Track
Expires: 26 February 2022

E. Lear
O. Friel
N. Cam-Winget
Cisco Systems
D. Harkins
HP Enterprise
25 August 2021

TEAP Update and Extensions for Bootstrapping
draft-lear-eap-teap-brski-06

Abstract

In certain environments, in order for a device to establish any layer three communications, it is necessary for that device to be properly credentialed. This is a relatively easy problem to solve when a device is associated with a human being and has both input and display functions. It is less easy when the human, input, and display functions are not present. To address this case, this memo specifies extensions to the Tunnel Extensible Authentication Protocol (TEAP) method that leverages Bootstrapping Remote Secure Key Infrastructures (BRSKI) in order to provide a credential to a device at layer two. The basis of this work is that a manufacturer will introduce the device and the local deployment through cryptographic means. In this sense the same trust model as BRSKI is used.

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 26 February 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. Terminology	3
2. TEAP BRSKI Architecture	4
3. BRSKI Bootstrap and Enroll Operation	4
3.1. Discovery of Trusted MASA	5
3.2. Executing BRSKI in a TEAP Tunnel	5
4. PKI Certificate Considerations	9
4.1. TEAP Tunnel Establishment	9
4.2. BRSKI Trust Establishment	11
4.3. Certificate Expiration Times	12
4.4. LDevID Subject and Subject Alternative Names	12
4.5. PKCS#10 Retry Handling	13
5. Peer Identity	13
6. Channel and Crypto Binding	14
7. Protocol Flows	14
7.1. TEAP Server Grants Access	14
7.2. TEAP Server Instructs Client to Perform BRSKI Flow	16
7.3. TEAP Server Instructs Client to Reenroll	20
7.4. Out of Band Reenroll	22
8. TEAP TLV Formats	22
8.1. New TLVs	22
8.1.1. BRSKI-RequestVoucher TLV	23
8.1.2. BRSKI-Voucher TLV	23
8.1.3. CSR-Attributes TLV	24
8.1.4. Retry-After TLV	25
8.1.5. NAI TLV	25
8.2. Existing TEAP TLV Specifications	25
8.2.1. PKCS#10 TLV	25
8.3. TLV Rules	26
9. Fragmentation	26
10. IANA Considerations	26
11. Security Considerations	27
11.1. Issues with Provisionally Authenticated TEAP	28
11.2. Attack Against Discovery	28
11.3. TEAP Server as Registration Authority	28
11.4. Trust of Registrar	29
12. Acknowledgments	29

13. References	29
13.1. Normative References	29
13.2. Informative References	30
Appendix A. Changes from Earlier Versions	30
Authors' Addresses	30

1. Introduction

[I-D.ietf-anima-bootstrapping-keyinfra] (BRSKI) specifies a means to provision credentials to be used as credentials to operationally access networks. It was designed as a standalone means where some limited access to an IP network is already available. This is not always the case. For example, IEEE 802.11 networks generally require authentication prior to any form of address assignment. While it is possible to assign an IP address to a device on some form of an open network, or to accept some sort of default credential to establish initial IP connectivity, the steps that would then follow might well require that the device is placed on a new network, requiring resetting all layer three parameters.

A more natural approach in such cases is to more tightly bind the provisioning of credentials with the authentication mechanism. One such way to do this is to make use of the Extensible Authentication Protocol (EAP) [RFC3748] and the Tunnel Extensible Authentication Protocol (TEAP) method [RFC7170]. Thus we define new TEAP Type-Length-Value (TLV) objects that can be used to transport the BRSKI protocol messages within the context of a TEAP TLS tunnel.

[RFC7170] discusses the notion of provisioning peers. Several different mechanisms are available. Section 3.8 of that document acknowledges the concept of not initially authenticating the outer TLS session so that provisioning may occur. In addition, exchange of multiple TLV messages between client and EAP server permits multiple provisioning steps.

1.1. Terminology

The reader is presumed to be familiar with EAP terminology as stated in [RFC3748]. In addition, the following terms are commonly used in this document.

- * BRSKI: Bootstrapping Remote Secure Key Infrastructures, as defined in [I-D.ietf-anima-bootstrapping-keyinfra]. The term is also used to refer to the flow described in that document.
- * EST: Enrollment over Secure Transport, as defined in [RFC7030].
- * Voucher: a signed JSON object as defined in [RFC8366].

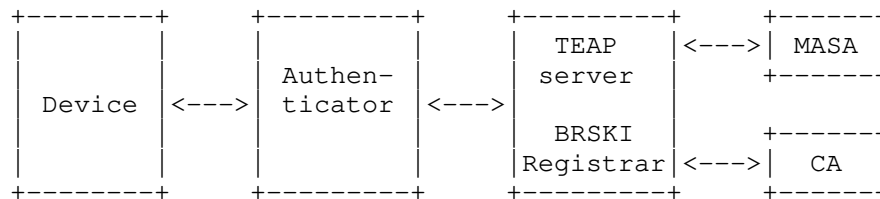
2. TEAP BRSKI Architecture

The TEAP BRSKI architecture is illustrated in Section 3. The device talks to the TEAP server via the Authenticator using any compliant transport such as [IEEE8021X]. The architecture illustrated shows an Authenticator distinct from the TEAP server. This is a deployment optimization and when so deployed the communication between Authenticator and TEAP server is a AAA protocol such as RADIUS or DIAMETER.

The architecture illustrated shows a co-located TEAP server and BRSKI registrar. Not only are these two functions co-located, they MUST be the same entity. This ensures that the entity identified in the device's voucher request (the TEAP server) is the same entity that signs the voucher request (the registrar).

The registrar communicates with the BRSKI MASA service for the purposes of getting signed vouchers.

The registrar also communicates with a Certificate Authority in order to issue LDevIDs. The architecture shows the registrar and CA as being two logically separate entities, however the CA may be integrated into the registrar. The device is not explicitly aware of whether the CA and registrar functions are integrated.



3. BRSKI Bootstrap and Enroll Operation

This section summarises the current BRSKI operation. The BRSKI flow assumes the device has an IDevID and has a manufacturer installed trust anchor that can be used to validate the BRSKI voucher. The BRSKI flow comprises several main steps from the perspective of the device:

- * Step 1: Device discovers the registrar
- * Step 2: Device establishes provisional TLS connection to registrar
- * Step 3: Device sends voucher request message and receives signed voucher response

- * Step 4: Device validates voucher and validates provisional TLS connection to registrar
- * Step 5: Device downloads additional local domain CA information
- * Step 6: Device downloads Certificate Signing Request (CSR) attributes
- * Step 7: Device does a certificate enroll to obtain an LDevID
- * Step 8: Device periodically reenrolls via EST to refresh its LDevID

Most of the operational steps require the device, and thus its internal state machine, to automatically complete the next step without being explicitly instructed to do so by the registrar. For example, the registrar does not explicitly tell the device to download additional local domain CA information, or to do an EST enroll to obtain an LDevID.

3.1. Discovery of Trusted MASA

BRSKI section 2.8 outlines how the Registrar discovers the correct MASA to connect with. BRSKI section 5.3 outlines how the Registrar can make policy decisions about which devices to trust.

Similar approaches are applicable for TEAP servers executing BRSKI. For example, the TEAP server may be configured with a list of trusted manufacturing CAs. During device bootstrap, only devices with an IDevID signed by a trusted manufacturing CA may be allowed to establish a TLS connection with the TEAP server, and the TEAP server could then extract the MASA URI from the device's IDevID.

3.2. Executing BRSKI in a TEAP Tunnel

This section outlines how the main BRSKI steps outlined above map to TEAP, and how BRSKI and enrollment can be accomplished inside a TEAP TLS tunnel. The following new TEAP TLVs are introduced:

- * BRSKI-VoucherRequest
- * BRSKI-Voucher
- * CSR-Attributes

The following steps outline how the above BRSKI flow maps to TEAP.

- * Step 1: Device discovers the registrar

When BRSKI is executed in a TEAP tunnel, the device exchanges BRSKI TLVs with the TEAP server. The discovery process for devices is therefore the standard wired or wireless LAN EAP server discovery process. The discovery processes outlined in section 4 of [I-D.ietf-anima-bootstrapping-keyinfra] are not required for initial discovery of the registrar.

* Step 2: Device establishes provisional TLS connection to registrar

The device establishes an outer TEAP tunnel with the TEAP server and does not validate the server certificate. The device presents its LDevID as its identity certificate if it has a valid LDevID, otherwise it presents its IDevID. The TEAP server validates the device's certificate using its implicit or explicit trust anchor database. If the device presents an IDevID it is verified against a database of trusted manufacturer certificates. Server policy may also be used to control which certificate the device is allowed present, as described in section {pki-certificate-authority-considerations}.

If the presented credential is sufficient to grant access, the TEAP server can return a TEAP Result TLV indicating success immediately. The device may still send a Request-Action TLV including a BRSKI-VoucherRequest TLV in response to the TEAP Result TLV if it does not have, but requires, provisioning of trust anchors for validating the TEAP server certificate. Note that no inner EAP method is required for this, only an exchange of TEAP TLVs.

[todo] Question: as the device wants the server to reply with a BRSKI-Voucher TLV, does it really send a Request-Action TLV containing a BRSKI-VoucherRequest TLV, or does it send a Request-Action TLV containing a BRSKI-Voucher TLV?? The TEAP draft is a bit ambiguous here. Normally, if one end sends a Request-Action including XXX-TLV, it means it wants the far end to send an XXX-TLV...

[todo] Question: general TEAP protocol question: does the device have to send a Request-Action w/BRSKI-VoucherRequest or can it send a BRSKI-VoucherRequest on its own? I'm not clear on this.

If the TEAP server requires that the device execute a BRSKI flow, the server sends a Request-Action TLV that includes a BRSKI-VoucherRequest TLV. For example, if the device presented its IDevID but the TEAP server requires an LDevID.

[todo] Question: to nit pick, the server should send a Request-Action TLV including a PKCS#10 TLV to tell the client to enroll. How does the server really know that the client has the correct trust

established (as previously received by a BRSKI-Voucher)? If the client sends an IDevID, does server always send a Request-Action including both BRSKI-VoucherRequest and PKCS#10 TLVs? Whats the client behaviour? I assume client can spontaneously send BRSKI-VoucherRequest and/or PCSK#10 without being explicitly instructed to. Just need to get the language correct here.

The TEAP server may also require the device to reenroll, for example, if the device presented a valid LDevID that is very closed to expiration. The server may instruct a device to reenroll by sending a Request-Action TLV that includes a zero byte length PKCS#10 TLV.

- * Step 3: Device sends voucher request message and receives signed voucher response

The device sends a BRSKI-RequestVoucher TLV to the TEAP server. The TEAP server forwards the RequestVoucher message to the MASA server, and the MASA server replies with a signed voucher. The TEAP server sends a BRSKI-Voucher TLV to the device.

If the MASA server does not issue a signed voucher, the TEAP server sends an EAP-Error TLV with a suitable error code to the device.

For wireless devices in particular, it is important that the MASA server only return a voucher for devices known to be associated with a particular registrar. In this sense, success indicates that the device is on the correct network, while failure indicates the device should try to provision itself within wireless networks (e.g, go to the next SSID).

- * Step 4: Device validates voucher and validates provisional TLS connection to registrar

The device validates the signed voucher using its manufacturer installed trust anchor, and uses the CA information in the voucher to validate the TLS connection to the TEAP server.

If the device fails to validate the voucher, then it sends a TEAP-Error TLV indicating failiure to the TEAP server.

Similarly, if the device validates the voucher, but fails to validate the provisional TLS connection, then it sends a TEAP-Error TLV indicating failure to the TEAP server. Note that the outer TLS tunnel has already been established, thus allowing the client to send a TEAP-Error TLV to the server inside that tunnel to indicate that it failed to verify the provisionally accepted outer TLS tunnel server identity.

- * Step 5: Device downloads additional local domain CA information

On completion of the BRSKI flow, the device SHOULD send a Trusted-Server-Root TLV to the TEAP server in order to discover additional local domain CAs. This is equivalent to section [todo] from [I-D.ietf-anima-bootstrapping-keyinfra].

- * Step 6: Device downloads CSR attributes

No later than the completion of step 5, server MUST send a CSR-Attributes TLV to peer server in order to discover the correct fields to include when it enrolls to get an LDevID.

- * Step 7: Device does a certificate enroll to obtain an LDevID

When executing the BRSKI flow inside a TEAP tunnel, the device does not directly leverage EST when doing its initial enroll. Instead, the device uses the existing TEAP PKCS#10 and PKCS#7 TEAP mechanisms.

Once the BRSKI flow is complete, the device can now send a PKCS#10 TLV to enroll and request an LDevID. If the TEAP server instructed the device to start the BRSKI flow via a Request-Action TLV that includes a BRSKI-RequestVoucher TLV, then the device MUST send a PKCS#10 in order to start the enroll process. The TEAP server will handle the PKCS#10 and ultimately return a PKCS#7 including an LDevID to the device.

If the TEAP server granted the device access on completion of the outer TEAP TLS tunnel in step 2 without sending a Request-Action TLV, the device does not have to send a PKCS#10 to enroll.

At this point, the device is said to be provisioned for local network access, and may authenticate in the future via 802.1X with its newly acquired credentials.

- * Step 8: Device periodically reenrolls to refresh its LDevID

When a device's LDevID is close to expiration, there are two options for re-enrollment in order to obtain a fresh LDevID. As outlined in Step 2 above, the TEAP server may instruct the device to reenroll by sending a Request-Action TLV including a PKCS#10 TLV. If the TEAP server explicitly instructs the device to reenroll via these TLV exchange, then the device MUST send a PKCS#10 to reenroll and request a fresh LDevID.

However, the device SHOULD reenroll if it determines that its LDevID is close to expiration without waiting for explicit instruction from the TEAP server. There are two options to do this.

Option 1: The device reenrolls for a new LDevID directly with the EST CA outside the context of the 802.1X TEAP flow. The device uses the registrar discovery mechanisms outlined in [I-D.ietf-anima-bootstrapping-keyinfra] to discover the registrar and the device sends the EST reenroll messages to the discovered registrar endpoint. No new TEAP TLVs are defined to facilitate discover of the registrar or EST endpoints inside the context of the TEAP tunnel.

Option 2: When the device is performing a periodic 802.1X authentication using its current LDevID, it reenrolls for a new LDevID by sending a PKCS#10 TLV inside the TEAP TLS tunnel.

4. PKI Certificate Considerations

There are multiple noteworthy PKI certificate handling considerations. These include:

- * PKI CA handling when establishing the TEAP tunnel
- * PKI CA handling establishing trust using BRSKI
- * IDevID and LDevID expiration times
- * Specifying LDevID Subject and Subject Alternative Names
- * PKCS#10 retry handling

These are described in more detail here.

4.1. TEAP Tunnel Establishment

Because this method establishes a client identity, if the peer has not been previously bootstrapped, or otherwise cannot successfully authenticate, it will use a generic identity string of teap-bootstrap@TBD1 as its network access identifier (NAI).

BRSKI section 5.3 outlines the policy decisions a Registrar may make when deciding whether to accept connections from clients. Similarly, the TEAP server operator may configure a set of trusted CAs for validating incoming TLS connections from clients. The operator may want to 'allow any device from a specific vendor', or from a set of vendors, to access the network. Network operators may do this by restricting network access to clients that have a certificate signed by one of a small set of trusted manufacturer/supplier CAs.

When the client sends its ClientHello to initiate TLS tunnel establishment, it is possible for the TEAP server to restrict the certificates that the client can use for tunnel establishment by including a list of CA distinguished names in the certificate_authorities field in the CertificateRequest message. The client should only continue with the handshake if it has a certificate signed by one of the indicated CAs.

In practice, network operators will likely want to onboard devices from a large number of device manufacturers, with each manufacturer using a different root CA when issuing IDevIDs. If the number of different manufacturer root CAs is large, this could result in very large TLS handshake messages. Therefore, the TEAP server may send a CertificateRequest message and not specify any certificate_authorities, thus allowing the client present a certificate signed by any authority in its Certificate message.

If the client has both an IDevID and an LDevID, the client should present the LDevID in preference to its IDevID, if allowed by server policy.

Once the client has sent its TLS Finished message, the TEAP server can make a policy decision, based on the CA used to sign the client's certificate, on whether to establish the outer TLS tunnel or not.

The TEAP server may delegate policy decisions to the MASA or CA function. For example, the TEAP server may declare EAP success and grant network access if the client presents a valid LDevID signed by a trusted domain CA. However, if the client presents an IDevID signed by a trusted manufacturer CA, the TEAP server may establish the TLS tunnel but not declare EAP success and grant network access until the client successfully completes a BRSKI Voucher exchange and PKCS#10/PKCS#7 exchange inside that tunnel.

It is recommended that the client validate the certificate presented by the server in the server's Certificate message, but this may not be possible for clients that have not yet provisioned appropriate trust anchors. If the client is in the provisioning phase and has not yet completed a BRSKI flow, it will not have trust anchors installed yet, and thus will not be able to validate the server's certificate. The client must however note the certificate presented by the server for (i) inclusion in the BRSKI-RequestVoucher TLV and for (ii) validation once the client has discovered the local domain trust anchors.

If the client does not present a suitable certificate to the server, the server MUST terminate the connection and fail the EAP request. If the TEAP server is unable to validate the client's certificate using its implicit or explicit trust anchor database it MUST fail the EAP request.

On establishment of the outer TLS tunnel, the TEAP server will make a policy decision on next steps. Possible policy decisions include:

- * Option 1: Server grants client full network access and returns EAP-Success. This will typically happen when the client presents a valid LDevID. Network policy may grant client network access based on LDevID without requiring the device to enroll to obtain an LDevID.
- * Option 2: Server requires that client perform a full BRSKI flow, and then enroll to get an LDevID. This will typically happen when the client presents a valid IDevID and network policy requires all clients to have LDevIDs. The server sends a Request-Action TLV that includes a BRSKI-RequestVoucher TLV to the client to instruct it to start the BRSKI flow.
- * Option 3: Server requires that the client reenroll to obtain a new LDevID. This could happen when the client presents a valid LDevID that is very close to expiration time, or the server's policy requires an LDevID update. The server sends an Action-Request TLV including a PKCS#10 TLV to the client to instruct it to reenroll.

4.2. BRSKI Trust Establishment

If the server requires that client perform a full BRSKI flow, it sends a Request-Action TLV that includes a zero byte length BRSKI-RequestVoucher TLV to the client. The client sends a new BRSKI-RequestVoucher TLV to the server, which contains all data specified in [I-D.ietf-anima-bootstrapping-keyinfra] section 5.2. The client includes the server certificate it received in the server's Certificate message during outer TLS tunnel establishment in the proximity-registrar-cert field. The client signs the request using its IDevID.

The server includes all additional information as required by [I-D.ietf-anima-bootstrapping-keyinfra] section 5.4 and signs the request prior to forwarding to the MASA.

The MASA responds as per [I-D.ietf-anima-bootstrapping-keyinfra] section 5.5. The response may indicate failure and the server should react accordingly to failures by sending a failure response to the client, and failing the TEAP method.

If the MASA replies with a signed voucher and a successful result, the server then forwards this response to the client in a BRSKI-Voucher TLV.

When the client receives the signed voucher, it validates the signature using its built in trust anchor list, and extracts the pinned-domain-cert field. The client must use the CA included in the pinned-domain-cert to validate the certificate that was presented by the server when establishing the outer TLS tunnel. If this certificate validation fails, the client must fail the TEAP request and not connect to the network.

[TBD- based on client responses, the registrar sends a status update to the MASA]

4.3. Certificate Expiration Times

[IEEE802.1AR] section 7.2.7.2 states:

notAfter: The latest time a DevID is expected to be used. Devices possessing an IDevID are expected to operate indefinitely into the future and should use the value 99991231235959Z. Solutions verifying an IDevID are expected to accept this value indefinitely.

TEAP servers SHOULD follow the 802.1AR standard when validating IDevIDs.

TEAP servers SHOULD reject LDevIDs with expired certificates and SHOULD NOT allow clients to connect with recently expired LDevIDs. If a client presents a recently expired LDevID it SHOULD be forced to authenticate using its IDevID and then reenroll to obtain a valid LDevID.

4.4. LDevID Subject and Subject Alternative Names

BRSKI section 5.9.2 specifies that the pledge MUST send a CSR Attributes request to the registrar. The registrar MAY use this mechanism to instruct the pledge about the identities it should include in the CSR request it sends as part of enrollment. The registrar may use this mechanism to tell the pledge what Subject or Subject Alternative Name identity information to include in its CSR request. This can be useful if the Subject must have a specific value in order to complete enrollment with the CA.

4.5. PKCS#10 Retry Handling

They will be scenarios where the TEAP server is willing to handle a PKCS#10 request from a client and issue a certificate via a PKCS#7 response, however, the TEAP server is unable to immediately completely the request and needs to instruct the client to retry later after a specified time interval.

A new Retry-After TLV is defined that the TEAP server uses to specify a retry interval in seconds. New error codes are defined to handle these two alternate retry scenarios.

- * The TEAP tunnel remains up: The client is instructed to resend the PKCS#10 request after a retry interval but inside the same TEAP tunnel. The TEAP server returns a Retry-After TLV to the client, and returns an Error TLV with a new code in the 1000-1999 range.
- * The TEAP tunnel is torn down: The client is instructed to establish a new TEAP connection and TEAP tunnel after a retry interval, and resend the PKCS#10 request inside the new tunnel. The TEAP server returns a Retry-After TLV to the client, and returns an Error TLV with a new code in the 2000-2999 range.

5. Peer Identity

EAP [RFC3748] recommends that "the Identity Response be used primarily for routing purposes and selecting which EAP method to use". NAI [RFC7542] recommends omitting the username part of an NAI in order to support username privacy, where appropriate.

A device that has not been bootstrapped at all SHOULD send an identity of teap-bootstrap@TBD1. Otherwise, a device SHOULD send its configured NAI.

The TEAP server may specify an NAI that it wishes the device to use. For example, the server may want a bootstrapped device to use an NAI of "abc123@example.com", or simply an NAI of "@example.com". This could be desirable in order to facilitate roaming scenarios. The server can do this by sending the device an NAI TLV inside the TEAP tunnel.

If the server specifies an NAI TLV, and the device handles the TLV, the device MAY use the specified NAI in all subsequent EAP authentication flows. If the device is not willing to handle the NAI TLV, it MUST reply with an Error TLV.

Authentication servers implementing this specification MAY reply with an Error TLV to any unrecognized NAI, or MAY attempt to bootstrap the device, regardless of the NAI. A device receiving an Error from the server MAY attempt a new session without the NAI in order to bootstrap.

6. Channel and Crypto Binding

As the TEAP BRSKI flow does not define or require an inner EAP method, there is no explicit need for exchange of Channel-Binding TLVs between the device and the TEAP server.

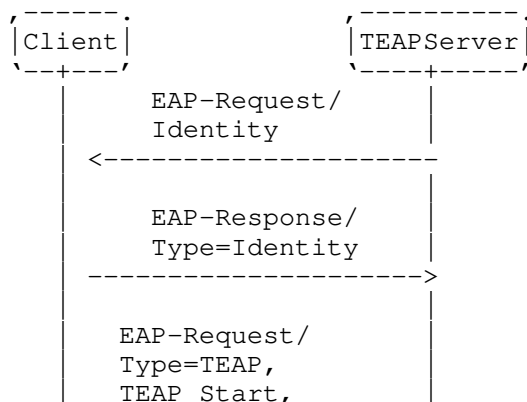
The TEAP BRSKI TLVs are expected to occur at the beginning of the TEAP Phase 2 and MUST occur before the final Crypto-Binding TLV. This draft does not exclude the possibility of having other EAP methods occur following the TEAP BRSKI TLVs and as such, the Crypto-Binding TLV process rules as defined in [RFC7170] apply.

7. Protocol Flows

This section outlines protocol flows that map to the three server policy options described in section Section 4.1. The protocol flows illustrate a TLS1.2 exchange. Pertinent notes are outlined in the protocol flows.

7.1. TEAP Server Grants Access

In this flow, the server grants access as server policy allows the client to access the network based on the identity certificate that the client presented. This means that either (i) the client has previously completed BRSKI and has presented a valid LDevID or (ii) the client presents an IDevID and network policy allows access based purely on IDevID.



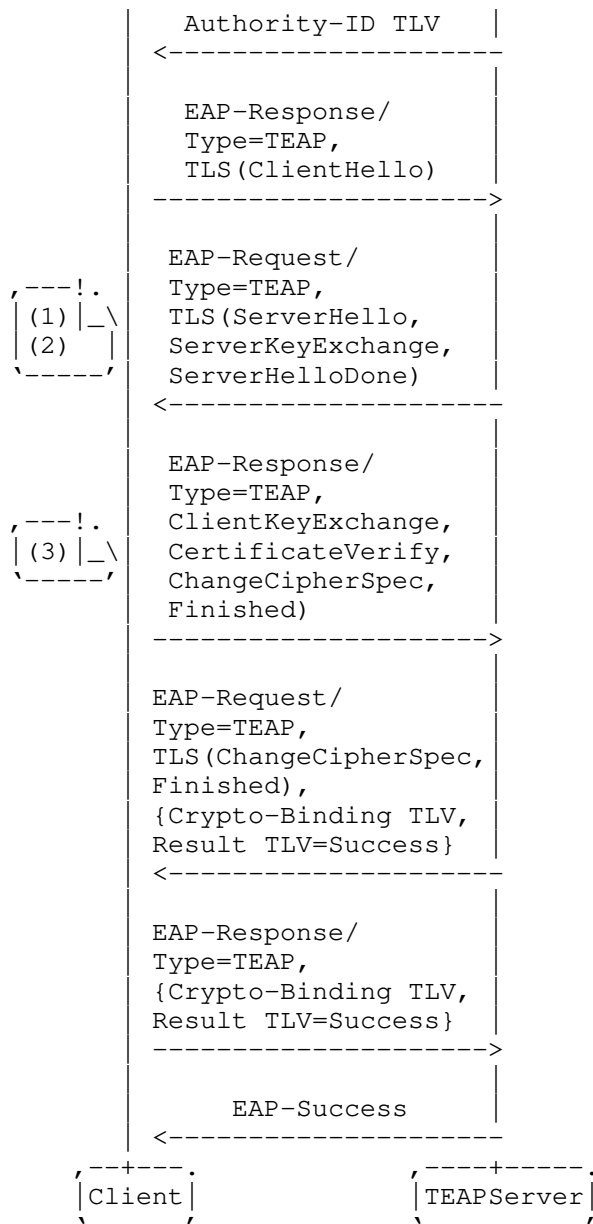


Figure 1: TEAP Server Grants Access

Notes:

(1) If the client has completed the BRSKI flow and has locally significant trust anchors, it must validate the Certificate received from the server. If the client has not yet completed the BRSKI flow, then it provisionally accepts the server Certificate and must validate it later once BRSKI is complete.

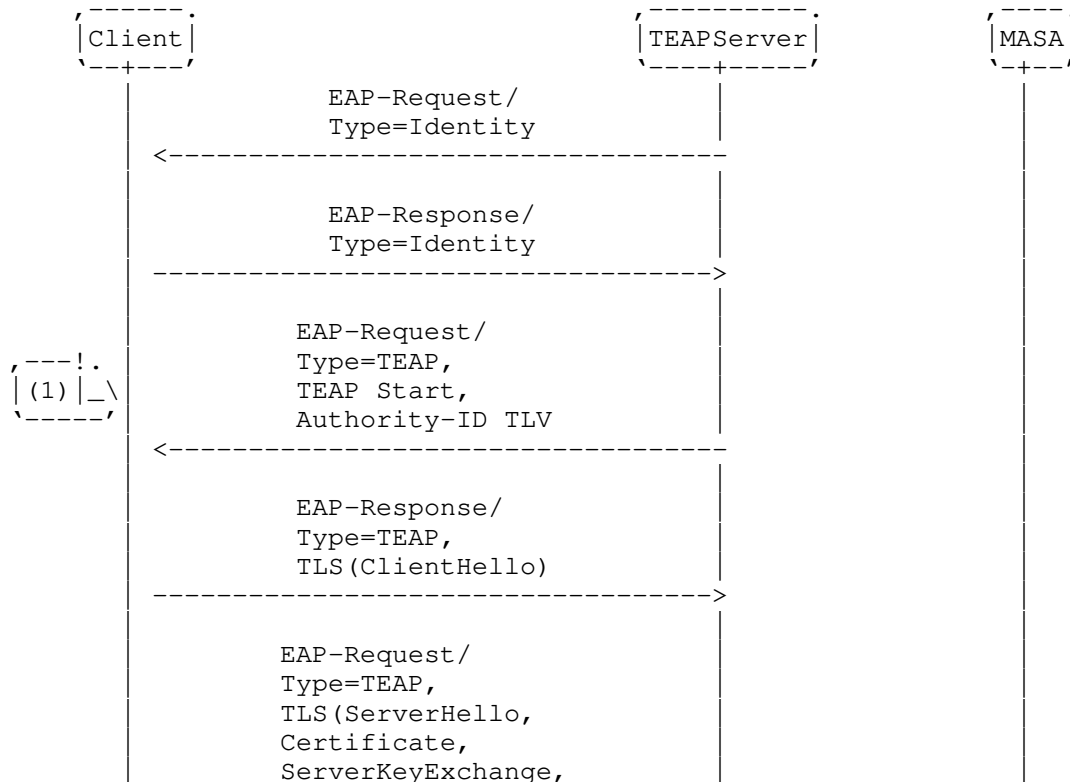
(2) The server may include `certificate_authorities` field in the `CertificateRequest` message in order to restrict the identity certificates that the device is allowed present.

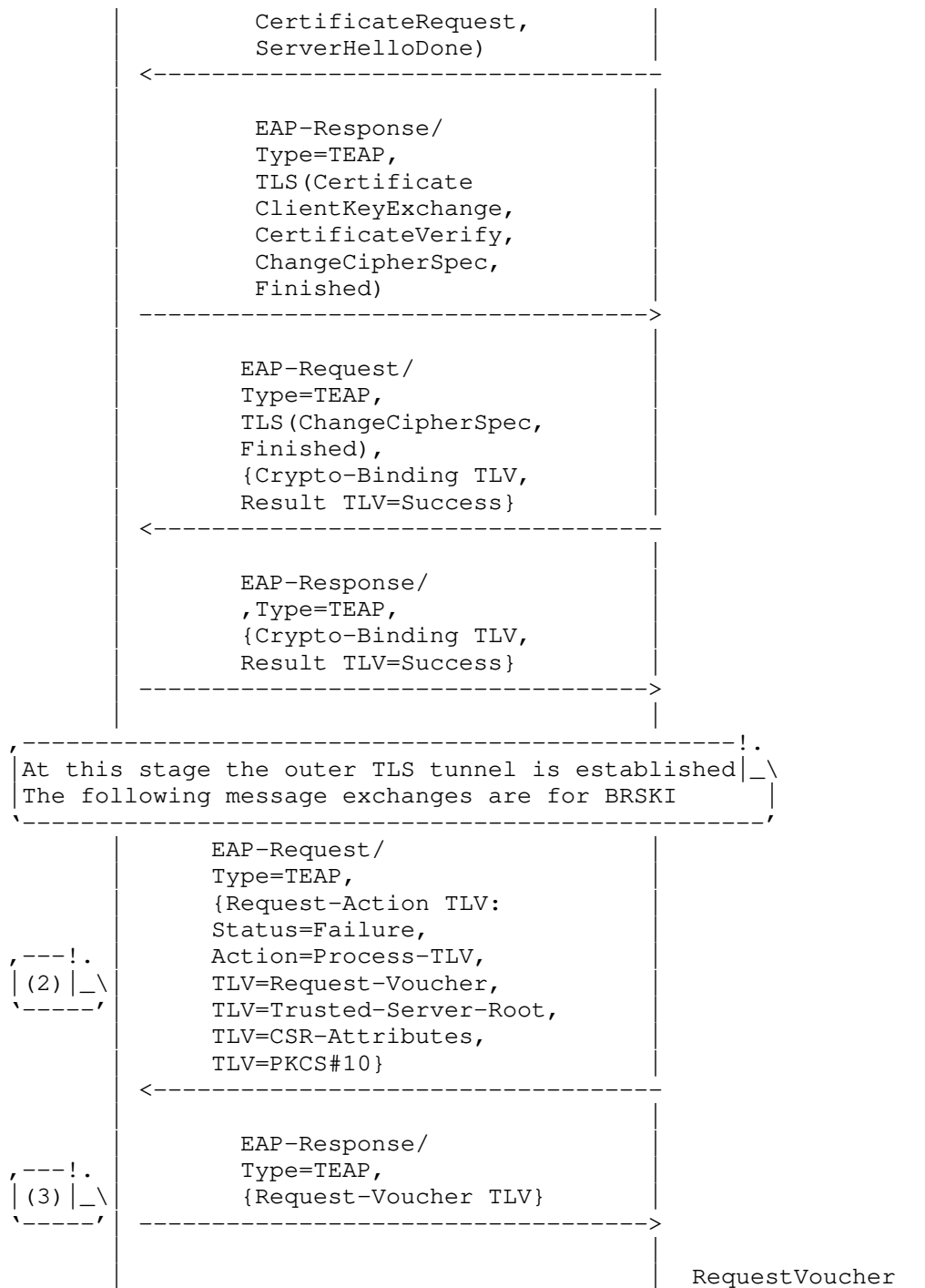
(3) The device will present its `LDevID`, if it has one, in preference to its `IDevID`, if allowed by server policy.

7.2. TEAP Server Instructs Client to Perform BRSKI Flow

In this two part flow, the server instructs the client to perform a BRSKI flow by exchanging TLVs once the outer TLS tunnel is established. After that, enrollment takes place.

In the first part of the flow, the MASA is depicted on the right.





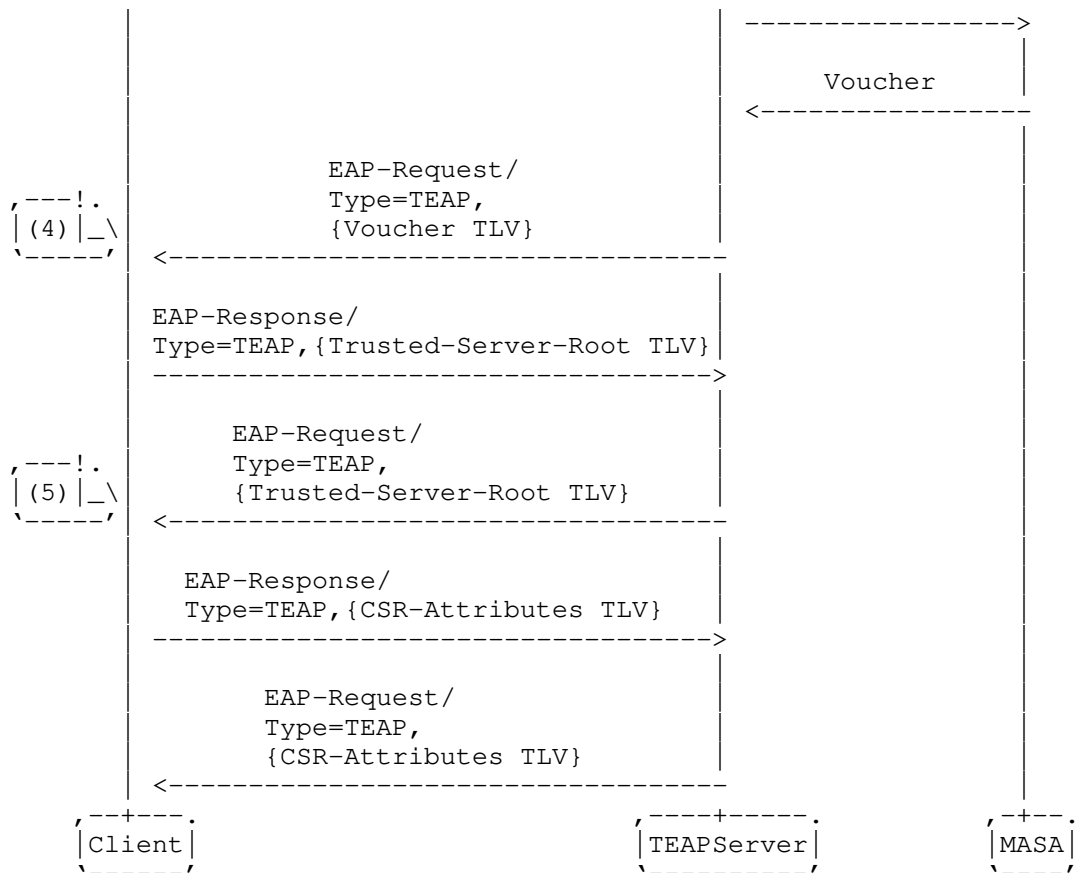


Figure 2: TEAP Server Instructs Client to Perform BRISKI Flow

The second part of the flow depicts the CA on the right.

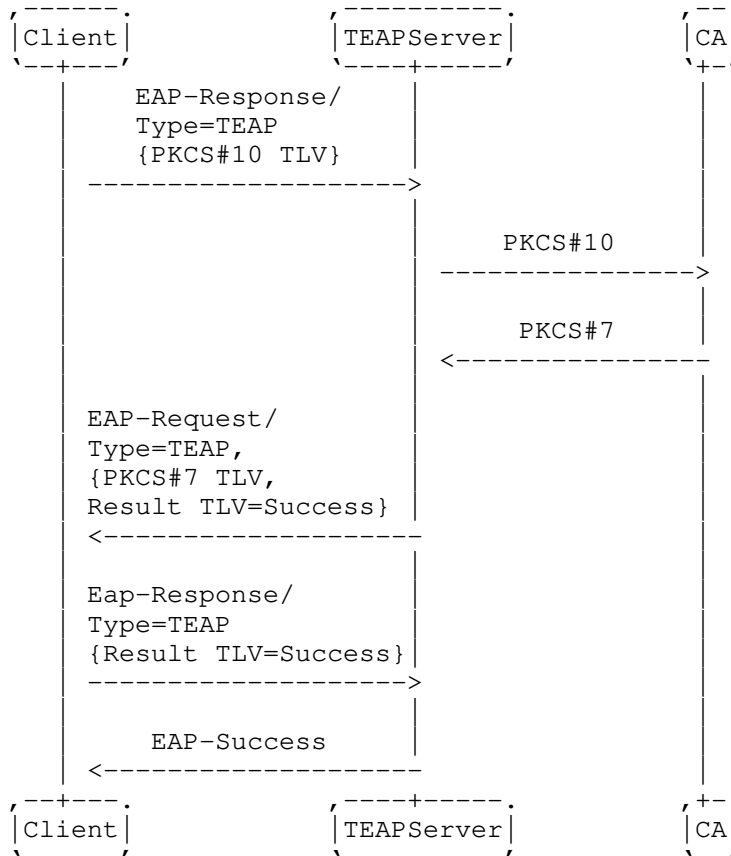


Figure 3: Enrollment after BRSKI Flow

Notes:

(1) If the client has not yet completed the BRSKI flow, then it provisionally accepts the server certificate and must validate it later once BRSKI is complete. The server validates the client certificate using its trust anchor database.

(2) The server instructs the client to start the BRSKI flow by sending a Request-Action TLV that includes a BRSKI-RequestVoucher TLV. The server also instructs the client to request trust anchors, to request CSR Attributes, and to initiate a PKCS certificate enrolment. As outlined in [RFC7170], the Request-Action TLV is sent after the Crypto-Binding TLV and Result TLV exchange.

(3) The client includes the certificate it received from the server in the RequestVoucher message.

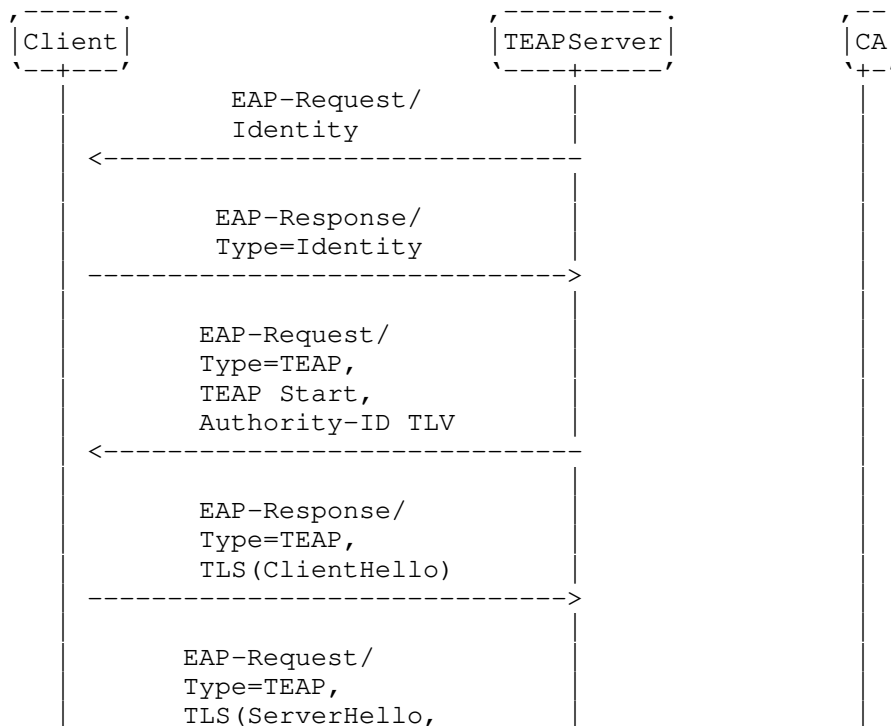
(4) Once the client receives and validates the voucher signed by the MASA, it must verify the certificate it previously received from the server.

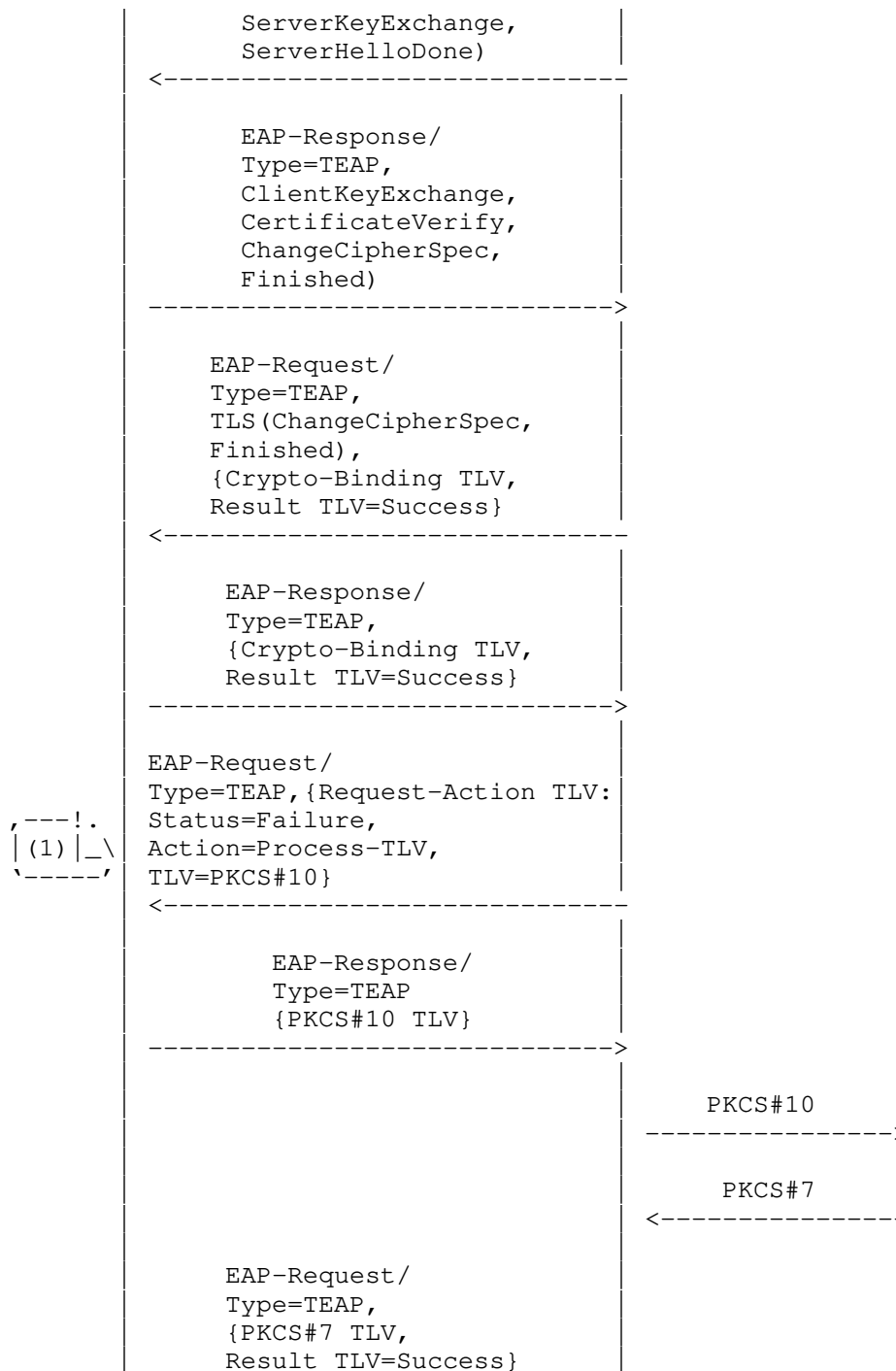
(5) As outlined in [RFC7170], the Trusted-Server-Root TLV is exchanged after the Crypto-Binding TLV exchange, and after the client has used the Voucher to authenticate the TEAP server identity. This is equivalent to section [todo] from [I-D.ietf-anima-bootstrapping-keyinfra].

(6) There is no need for an additional Crypto-Binding TLV exchange as there is no inner EAP method. All BRSKI exchanges are simply TLVs exchanged inside the outer TLS tunnel.

7.3. TEAP Server Instructs Client to Reenroll

In this flow, the server instructs the client to reenroll and get a new LDevID by exchanging TLVs once the outer TLS tunnel is established.





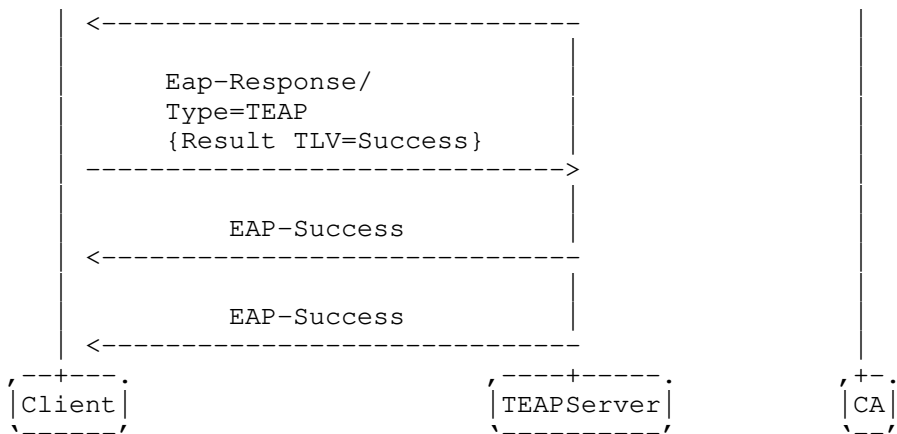


Figure 4: TEAP Server Instructs Client to Reenroll

(1) The server instructs the client to reenroll by sending a Request-Action TLV that includes a PKCS#10 TLV.

7.4. Out of Band Reenroll

This section shows how the device does a reenroll to refresh its LDEVID directly against the registrar outside the context of the TEAP tunnel.

8. TEAP TLV Formats

8.1. New TLVs

This document defines 5 new TEAP TLVs. The following table indicates whether the TLVs can be included in Request messages from TEAP server to device, or Response messages from device to TEAP server.

TLV	Message
BRSKI-VoucherRequest	Response
BRSKI-Voucher	Request
CSR-Attributes	Response
Retry-After	Response
NAI-Identity	Request

These new TLVs are detailed in this section.

8.1.1. BRSKI-RequestVoucher TLV

This TLV is used by the server as part of a Request-Action TLV to request from the peer that it initiate a voucher request. When used in this fashion, the length of this TLV will be set to zero. The Status field of the Request-Action TLV MUST be set to Failure.

It is also used by the peer to initiate the voucher request. When used in this fashion, the length of the TLV will be set to that of the voucher request, as encoded and described in Section 3.3 in [I-D.ietf-anima-bootstrapping-keyinfra].

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD1-VoucherRequest | Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The M and R bits are always expected to be set to 0.

The server is expected to forward the voucher request to the MASA, and then return a voucher in a BRSKI-Voucher TLV as described below. If it is unable to do so, it returns an TEAP Error TLV with one of the defined errors or the following:

```

TBD2-MASA-Notavailable  MASA unavailable
TBD3-MASA-Refused      MASA refuses to sign the voucher

```

The peer terminates the TEAP connection, but may retry at some later point. The backoff mechanism for such retries should be appropriate for the device. Retries MUST occur no more frequently than once every two (TBD) minutes.

8.1.2. BRSKI-Voucher TLV

This TLV is transmitted from the server to the peer. It contains a signed voucher, as describe in [RFC8366].

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R| TLV=TBD4-Voucher | Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```


Upon receiving this TLV the peer will validate the signature of the voucher, using its pre-installed manufacturer trust anchor (LDevID). It MUST also validate the certificate used by the server to establish the TLS connection.

If successful, it installs the new trust anchor contained in the voucher.

Otherwise, the peer transmits an TEAP error TLV with one of the following error messages:

TBD5-Invalid-Signature The signature on the voucher is invalid
 TBD6-Invalid-Voucher The form or content of the voucher is invalid
 TBD7-Invalid-TLS-Signer The certificate used for the TLS connection could not be validated.

8.1.3. CSR-Attributes TLV

The server SHALL transmit this TLV to the peer, either along with the BRSKI-Voucher TLV or at any time earlier in a communication. The peer shall include attributes required by the server in any following CSR. The value of this TLV is the base64 encoding described in Section 4.5.2 of [RFC7030].

The TEAP server MAY use this TLV to specify the subject identity information to include in Subject or Subject Alternate Name fields in any following CSR.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-----+-----+-----+-----+-----+-----+-----+-----+
      |M|R| TLV=TBD8-CSR-Attributes |          length          |
      +-----+-----+-----+-----+-----+-----+-----+-----+
      | Value...
      +-----+-----+-----+-----+-----+-----+-----+-----+
  
```

Again, the M and R values are set to 0. In the case where the client is unable to provide the requested attributes, an TEAP-Error is returned as follows:

TBD9-CSR-Attribute-Fail Unable to supply the requested attributes.

8.1.4. Retry-After TLV

The server MUST transmit this TLV to the peer when replying to a PKCS#10 TLV request from the peer where the server is willing to fulfill the request and issue a certificate via a PKCS#7 response, but is unable to fulfill the request immediately. This TLV is used to tell the peer the minimum length of time it MUST wait before resending the PKCS#10 request. The value of this TLV is the time in seconds that the peer MUST wait before resending the PKCS#10 request. The peer MUST resend the exact same PKCS#10 request.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R|  TLV=TBD10-Retry-After  |          length          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Again, the M and R values are set to 0.

8.1.5. NAI TLV

The server may use this TLV to provision a realm-specific NAI on the device.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R|  TLV=TBD10-NAI          |          length          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value...
+-----+-----+-----+-----+-----+-----+-----+-----+

```

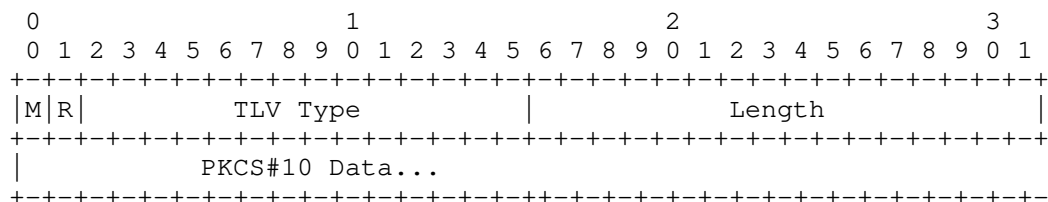
Again, the M and R values are set to 0.

8.2. Existing TEAP TLV Specifications

This section documents allowed usage of existing TEAP TLVs. The definition of the TLV is not changed, however clarifications on allowed values for the TLV fields is documented.

8.2.1. PKCS#10 TLV

[RFC7170] defines the PKCS#10 TLV as follows:



[RFC7170] does not explicitly allow a Length value of zero.

A Length value of zero is allowed for this TLV when the TEAP server sends a Request-Action TLV with a child PKCS#10 TLV to the client. In this scenario, there is no PKCS#10 Data included in the TLV. Clients MUST NOT send a zero length PKCS#10 TLV to the server.

8.3. TLV Rules

BRSKI TLVs can only be transported inside the TLS tunnel. The following table provides a guide to which TLVs may be encapsulated in which kind of packets, and in what quantity. The messages are as follows: Request is a TEAP Request, Response is a TEAP Response, Success is a message containing a successful Result TLV, and Failure is a message containing a failed Result TLV.

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

0 This TLV MUST NOT be present in the message.

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

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

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

Request	Response	Success	Failure	TLVs	0	0-1	0	0	BRSKI-VoucherRequest
0-1	0	0	0	BRSKI-Voucher	0	0-1	0	0	CSR-Attributes

9. Fragmentation

TEAP is expected to provide fragmentation support. Thus EAP-TEAP-BRSKI does not specifically provide any, as it is only expected to be used as an inner method to TEAP.

10. IANA Considerations

The IANA is requested to add entries into the following tables:

The following new TEAP TLVs are defined:

TBD1-VoucherRequest	Described in this document.
TBD4-Voucher	Described in this document.
TBD8-CSR-Attributes	Described in this document.
TBD10-Retry-After	Described in this document.

The following TEAP Error Codes are defined, with their meanings listed here and in previous sections:

TBD2-MASA-Notavailable	MASA unavailable
TBD3-MASA-Refused	MASA refuses to sign the voucher
TBD5-Invalid-Signature	The signature on the voucher is invalid
TBD6-Invalid-Voucher	The form or content of the voucher is invalid
TBD7-Invalid-TLS-Signer	The certificate used for the TLS connection could not be validated.
TBD9-CSR-Attribute-Fail	Unable to supply the requested attributes.
TBD11-Retry-PKCS#10	Retry PKCS#10 Request (1000 range code)
TBD12-Retry-PKCS#10	Retry PKCS#10 Request (2000 range code)
TBD13-NAI-Rejected	The device will not use the indicated NAI (1000 range code)

[[TODO: is there a registry of NAIs that map to TEAP methods? e.g. @eap-teap.net is reserved to indicate the peer wants to use TEAP method]]

11. Security Considerations

BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] provides a zero touch way for devices to enroll in a certification authority (CA). It assumes the device has IP connectivity. For networks that will not grant IP connectivity before authenticating (with a local credential) this poses a Catch-22- can't get on the network without a credential and can't get a credential without getting on the network.

This protocol provides a way for BRSKI to be in an EAP method which allows the BRSKI conversation to happen as part of EAP authentication and prior to obtaining IP connectivity.

The security considerations of [I-D.ietf-anima-bootstrapping-keyinfra] apply to this protocol. Running BRSKI through EAP introduces some additional areas of concern though.

11.1. Issues with Provisionally Authenticated TEAP

This protocol establishes an unauthenticated TLS connection and passes data through it. Provided that the only messages passed in this state are self-protected BRSKI messages this does not present a problem. Passing any other messages or TLVs prior to authentication of the provisional TLS connection could potentially introduce security issues.

While the TLS connection is unauthenticated, it must still be validated to the fullest extent possible. It is critical that the device and the TEAP server perform all steps in TLS- checking the validity of the presented certificate, validating the signature using the public key of the certificate, etc- except ensuring the trust of the presented certificate.

11.2. Attack Against Discovery

The device discovery technique specified in this protocol is the standard EAP server discovery process. Since it is trivial to set up an 802.11 wireless access point and advertise any network, an attacker can impersonate a legitimate wireless network and attract unprovisioned pledges. Given that an unprovisioned device will not know the legitimate network to connect to, it will probably attempt the first network it finds, making the attack that much easier. This allows for a "rogue registrar" to provision and take control of the device.

If the MASA verifies ownership prior to issuance of a voucher, this attack can be thwarted. But if the MASA is in reduced security mode and does not verify ownership this attack cannot be prevented. Registrars SHOULD use the audit log of a MASA when deploying newly purchased equipment in order to mitigate this attack.

Another way to mitigate this attack is through normal "rogue AP" detection and prevention.

11.3. TEAP Server as Registration Authority

If the TEAP server is logically separate from the Certification Authority (CA) (see Section 2) it will be acting as a Registration Authority (RA) when it obtains the PKCS#10 TLV and replies with a PKCS#7 TLV (see [RFC7170], Sections 4.2.16 and 4.2.17, respectively). The assurance a RA makes to a CA is that the public key in the presented CSR is bound to an authenticated identity in way that will assure non-repudiation.

To make such an assurance, the TEAP server MUST authenticate the provisional TLS connection with the device by validating the voucher response received from the MASA. In addition, it is RECOMMENDED that the TEAP server indicate that proof-of-possession (see [RFC7170], Section 3.8.2) is required by including the challengePassword OID in the CSR Attributes TLV.

11.4. Trust of Registrar

The device accepts a trusted server (CA) certificate and installs it in its trust anchor database during step 5 (see Section 3.2). This can happen only after the provisional TLS connection has been authenticated using the voucher and the Crypto-Binding TLV has been validated.

12. Acknowledgments

The authors would like to thank Brian Weis for his assistance, and Alan Dakok for improving language consistency. In addition, with ruthlessly "borrowed" the concept around NAI handling from Tuomas Aura and Mohit Sethi.

13. References

13.1. Normative References

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M. C., Eckert, T., Behringer, M. H., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-bootstrapping-keyinfra-45, 11 November 2020, <<https://www.ietf.org/archive/id/draft-ietf-anima-bootstrapping-keyinfra-45.txt>>.
- [IEEE8021AR]
Institute for Electrical and Electronics Engineers, "Secure Device Identity", 1998.
- [IEEE8021X]
Institute for Electrical and Electronics Engineers, "IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control", 2010.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.

- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<https://www.rfc-editor.org/info/rfc7030>>.
- [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>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert,
"A Voucher Artifact for Bootstrapping Protocols",
RFC 8366, DOI 10.17487/RFC8366, May 2018,
<<https://www.rfc-editor.org/info/rfc8366>>.

13.2. Informative References

- [RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542,
DOI 10.17487/RFC7542, May 2015,
<<https://www.rfc-editor.org/info/rfc7542>>.

Appendix A. Changes from Earlier Versions

Draft -06: * nothing more than version bump

Draft -03: * Merge EAP server and Registrar * Security considerations
* References improvements * Add Dan Harkins as co-author

Draft -02: * Flow corrections

Draft -01: * Add packet descriptions, IANA considerations, smooth out
language.

Draft -00:

* Initial revision

Authors' Addresses

Eliot Lear
Cisco Systems
Richtistrasse 7
CH-8304 Wallisellen
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Owen Friel
Cisco Systems
170 W. Tasman Dr.
San Jose, CA, 95134
United States

Email: ofriel@cisco.com

Nancy Cam-Winget
Cisco Systems
170 W. Tasman Dr.
San Jose, CA, 95134
United States

Email: ncamwing@cisco.com

Dan Harkins
HP Enterprise
3333 Scott Boulevard
Santa Clara, CA, 95054
United States

Email: dharkins@arubanetworks.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2020

M. Pala
CableLabs
October 28, 2019

Credentials Provisioning and Management via EAP (EAP-CREDS)
draft-pala-eap-creds-05

Abstract

With the increase number of devices, protocols, and applications that rely on strong credentials (e.g., digital certificates, keys, or tokens) for network access, the need for a standardized credentials provisioning and management framework is paramount. The 802.1x architecture allows for entities (e.g., devices, applications, etc.) to authenticate to the network by providing a communication channel where different methods can be used to exchange different types of credentials. However, the need for managing these credentials (i.e., provisioning and renewal) is still a hard problem to solve.

EAP-CREDS, if implemented in Managed Networks (e.g., Cable Modems), could enable our operators to offer a registration and credentials management service integrated in the home WiFi thus enabling visibility about registered devices. During initialization, EAP-CREDS also allows for MUD files or URLs to be transferred between the EAP Peer and the EAP Server, thus giving detailed visibility about devices when they are provisioned with credentials for accessing the networks. The possibility provided by EAP-CREDS can help to secure home or business networks by leveraging the synergies of the security teams from the network operators thanks to the extended knowledge of what and how is registered/authenticated.

This specifications define how to support the provisioning and management of authentication credentials that can be exploited in different environments (e.g., Wired, WiFi, cellular, etc.) to users and/or devices by using EAP together with standard provisioning protocols.

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 April 30, 2020.

Copyright Notice

Copyright (c) 2019 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. Requirements notation	4
2. Introduction	4
2.1. Overview of existing solutions	4
2.2. Scope Statement	5
2.3. EAP-CREDS as tunneled mechanism only	5
2.4. Fragmentation Support	5
2.5. Encapsulating Provisioning Protocols in EAP-CREDS	6
2.6. Algorithm Requirements	6
2.7. Notation	7
3. EAP-CREDS Protocol	7
3.1. Message Flow	7
3.2. Phase Transitioning Rules	8
3.3. Phase One: Initialization	9
3.4. Phase Two: Provisioning	11
3.5. Phase Three: Validation	13
4. EAP-CREDS Message Format	15
4.1. Message Header	15
4.2. Message Payload	17
4.3. EAP-CREDS defined TLVs	18
4.3.1. The Action TLV	18
4.3.2. The Certificate-Data TLV	19
4.3.3. The Challenge-Data TLV	20
4.3.4. The Challenge-Response TLV	21
4.3.5. The Credentials-Information TLV	22

4.3.6.	The Credentials-Data TLV	25
4.3.7.	The Error TLV	25
4.3.8.	The Network-Usage TLV	26
4.3.9.	The Profile TLV	28
4.3.10.	The Protocol TLV	29
4.3.11.	The Provisioning-Data TLV	29
4.3.12.	The Provisioning-Headers TLV	30
4.3.13.	The Provisioning-Params TLV	31
4.3.14.	The Certificate-Request TLV	33
4.3.15.	The Storage-Info TLV	34
4.3.16.	The Supported-Formats TLV	35
4.3.17.	The Supported-Encoding TLV	36
4.3.18.	The Token-Data TLV	36
4.3.19.	The Version TLV	37
5.	EAP-CREDS Messages	38
5.1.	The EAP-CREDS-Init Message	38
5.1.1.	EAP Server's Init Message	38
5.1.2.	EAP Peer's Init Message	39
5.1.2.1.	Bootstrapping Peer's Trustworthiness	39
5.1.3.	The EAP-CREDS-Provisioning Message	40
5.1.4.	The EAP-CREDS-Validate Message	41
6.	Error Handling in EAP-CREDS	42
7.	The Simple Provisioning Protocol (SPP)	42
7.1.	SPP Message Format	43
7.2.	SPP Message Flow	43
7.2.1.	SPP Symmetric Secrets Management	46
7.2.1.1.	Server Side Only Generation	47
7.2.1.2.	Client Side Only Generation	47
7.2.1.3.	Client and Server Side Generation	49
7.2.2.	SPP Key Pair Provisioning	49
7.2.2.1.	Server Side Only Generation	49
7.2.2.2.	Client Side Only Generation	49
7.2.2.3.	Client and Server Side Generation	49
7.2.3.	SPP Certificate Provisioning	49
7.2.3.1.	Server Side Only Generation	49
7.2.3.2.	Client Side Only Generation	50
7.2.3.3.	Client and Server Side Generation	50
7.2.4.	SPP Token Provisioning	50
7.2.4.1.	Server Side Only Generation	50
7.2.4.2.	Client Side Only Generation	50
7.2.4.3.	Client and Server Side Generation	50
8.	IANA Considerations	50
8.1.	Provisioning Protocols	51
8.2.	Token Types	51
8.3.	Credentials Types	51
8.4.	Credentials Algorithms	52
8.5.	Credentials Datatypes	52
8.6.	Challenge Types	53

8.7. Network Usage Datatypes	53
8.8. Credentials Encoding	54
8.9. Action Types	54
8.10. Usage Metadata Types	54
9. Security Considerations	55
10. Acknowledgments	55
11. Normative References	55
Author's Address	56

1. Requirements notation

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

2. Introduction

Many environments are, today, moving towards requiring strong authentication when it comes to gain access to networks. The 802.1x architecture provides network administrators with the possibility to check credentials presented by a device even before providing any connectivity or IP services to it.

However, the provisioning and management of these credentials is a hard problem to solve and many vendors opt for long-lived credentials that can not be easily revoked, replaced, or simply renewed.

This specification addresses the problem of providing a simple-to-use and simple-to-deploy conduit for credentials management by extending the EAP protocol to support credentials provisioning and management functionality. In particular, the EAP-CREDS method defined here provides a generic framework that can carry the messages for provisioning different types of credentials. EAP-CREDS cannot be used as a stand-alone method, it is required that EAP-CREDS is used as an inner method of EAP-TLS, EAP-TEAP, or any other tunnelling method that can provide the required secrecy and (at minimum) server-side authentication to make sure that the communication is protected and with the right server.

2.1. Overview of existing solutions

Currently there are many protocols that address credentials lifecycle management. In particular, when it comes to digital certificates, some of the most deployed management protocols are: Certificate Management Protocol (CMP) [RFC4210], Certificate Management over CMS (CMC) [RFC5272][RFC6402], Enrollment over Secure Transport (EST) [RFC7030], and Automated Certificate Management Environment (ACME) . However, none of these protocols provide native support for client

that do not have IP connectivity yet (e.g., because they do not have network-access credentials, yet). EAP-CREDS provides the possibility to use such protocols (i.e., message-based) by defining a series of messages that can be used to encapsulate the provisioning messages for the selected provisioning protocol.

In addition to these protocols, EAP-CREDS also defines a series of simple messages that provide a generic enrollment protocol that allows not only certificates but also other types of credentials (e.g., username/password pairs, tokens, or symmetric secrets) to be delivered to the client as part of the provisioning and/or renewal process. The set of messages that make up the generic provisioning protocol is referred to as the Simple Provisioning Protocol or SPP.

2.2. Scope Statement

This document focuses on the definition of the EAP-CREDS method to convey credentials provisioning and managing messages between the client and the AAA server. Moreover, the document defines how to encode messages for the main IETF provisioning protocols.

This document, however, does not provide specifications for how and where the credentials are generated. In particular, the credentials could be generated directly within the AAA server or at a different location (i.e., the Certificate Service Provider or CSP) site. Different authentication mechanisms (e.g., TLS, etc.) can be used to secure the communication between the server's endpoint and the CSP.

2.3. EAP-CREDS as tunneled mechanism only

EAP-CREDS requires that an outer mechanism is in place between the Peer and the Server in order to provide authentication and confidentiality of the messages exchanged via EAP-CREDS. In other words, EAP-CREDS assumes that an appropriately encrypted and authenticated channel has been established to prevent the possibility to leak information or to allow man-in-the-middle attacks.

This choice was taken to simplify the message flow between Peer and Server, and to abstract EAP-CREDS from the secure-channel establishment mechanism. EAP-TLS, or EAP-TEAP are examples of such mechanisms.

2.4. Fragmentation Support

EAP does not directly support handling fragmented packets and it requires the outer method to provide fragmentation support.

Because of the outer method requirements in particular, removing any support for fragmented messages in EAP-CREDS removes the duplication of packets (e.g., Acknowledgment Packets) sent across the Peer and the Server, thus resulting in a smaller number of exchanged messages

2.5. Encapsulating Provisioning Protocols in EAP-CREDS

In order to use EAP-CREDS together with your favorite provisioning protocol, the messages from the provisioning protocol need to be sent to the other party. In EAP-CREDS, this is done by encoding the provisioning protocol messages inside the ('Provisioning-Data') TLV. In case the provisioning protocol uses additional data for its operations (e.g., uses HTTP Headers), this data can be encoded in a separate ('Provisioning-Headers') TLV.

Since the implementation of the provisioning endpoint could happen in a (logically or physically) different component, a method is needed to identify when a provisioning protocol has actually ended. In EAP-CREDS, the 'D' bit in the message headers is used for this purpose.

In the first message of Phase Two, the Server provides the client with all the selected parameters for one specific credential that needs attention (or for a new credential) to be managed by the network. In particular, the server provides, at minimum, the ('Protocol') TLV, the ('Action') TLV, and the ('Provisioning-Params') or the ('Credentials-Info') TLV.

After checking the parameters sent by the Server, if the Peer does not support any of the proposed ones, it MUST send a message with one single ('Error') TLV with the appropriate error code(s). The server, can then decide if to manage a different set of credentials (if more were reported by the Peer in its Phase One message) or if to terminate the EAP session with an error.

The Peer and the Server exchange Provisioning messages until an error is detected (and the appropriate error message is sent to the other party) or until Phase Two is successfully completed.

2.6. Algorithm Requirements

EAP-CREDS uses the SHA-256 hashing algorithm to verify credentials in phase three of the protocol. Peers and Servers MUST support SHA-256 for this purpose.

2.7. Notation

In this document we use the following notation in the diagrams to provide information about the cardinality of the data structures (TLVs) within EAP-CREDS messages:

Symbol	Example	Usage
{ }	{TLV1}	Curly Brackets are used to indicate a set
[]	{[TLV2]}	Square Brackets are used to indicate that a field is optional
()	{TLV1(=V)}	Round Squares are used to specify a value
+	{TLV_2+}	The Plus character indicates that one or more instances are allowed

Table 1: EAP-CREDS Notation

3. EAP-CREDS Protocol

In a nutshell, EAP-CREDS provides the abstraction layer on top of which credentials provisioning/managing protocols can be deployed thus enabling their use even before provisioning IP services.

This section outlines the operation of the protocol and message flows. The format of the CREDS messages is given in Section 4.

3.1. Message Flow

EAP-CREDS message flow is logically subdivided into three different phases: Initialization, Provisioning, and Validation. EAP-CREDS enforces the order of phases, i.e. it is not possible to move to an earlier phase.

Phase transitioning is controlled by the Server. In particular, the server, after the last message of a phase, it can decide to either (a) start the next phase by sending the first message of the next phase, or (b) continue the same phase by sending another "first" message of the phase (e.g., managing a second set of credentials) - this is allowed only in Phase Two and Phase Three but NOT in Phase One, or (c) terminate the EAP session.

Phase One (Required). Initialization. During this phase the Peer and the Server exchange the information needed to select the appropriate credentials management protocol. Phase One flow is composed by only messages. In particular, the Server sends its initial message of type ('EAP-CREDS-Init'). The Peer replies with

the details about which provisioning protocols are supported, and additional information such as the list of installed credentials and, optionally, authorization data (for new credentials registration).

Phase Two (Optional). Provisioning Protocol Flow. In this phase, the Peer and the Server exchange the provisioning protocol's messages encapsulated in a EAP-CREDS message of type Provisioning. The messages use two main TLVs. The first one is the ('Provisioning-Headers') TLV which is optional and carries information that might be normally conveyed via the transport protocol (e.g., HTTP headers). The second one is the ('Provisioning-Data'), which is required and carries the provisioning protocol's messages. The server can decide to repeat phase two again to register new credentials or to renew a separate set of credentials by issuing a new ('Provisioning') message for the new target. When no more credentials have to be managed, the Server can start phase three or simply terminate the EAP session.

Phase Three (Optional). Credentials Validation. This optional phase can be initiated by the server and it is used to validate that the Peer has properly installed the credentials and can use them to authenticate itself. Depending on the credentials' type, the messages can carry a challenge/nonce, the value of the secret/token, or other information. The format of the credentials is supposed to be known by the provider and the device.

3.2. Phase Transitioning Rules

In order to keep track of starting and ending a phase, EAP-CREDS defines several bits and fields in the EAP-CREDS message headers. In particular, as described in Section 4.1, the 'S' bit is used to indicate the beginning (or Start) of a phase, while the 'Phase' field (4 bits) is used to indicate the phase for this message.

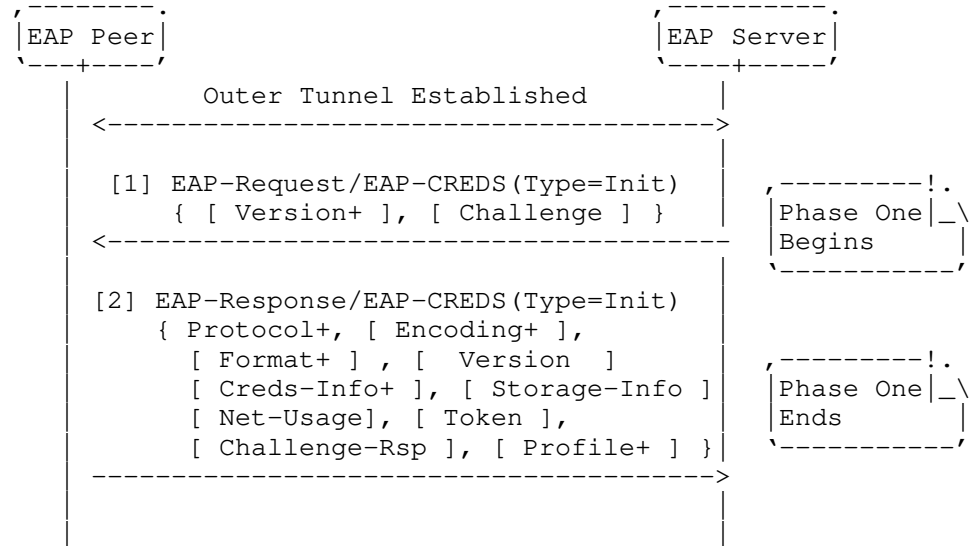
In EAP-CREDS, phase transitioning is under the sole control of the Server, therefore the value of the 'S' bit is meaningful only in messages sent by the Server. The value of the 'S' bit in Peer's messages SHALL be set to '0x0' and SHALL be ignored by the server.

When starting a new phase, the Server MUST set the 'S' bit to '1' and the 'Phase' field to the current phase number (e.g., one, two, or three).

In case the first message of a phase is to be repeated (e.g., because of processing multiple credentials), the 'S' bit SHALL be set to '0' (i.e., it should be set to '1' only on the first occurrence and set to '0' in subsequent messages).

3.3. Phase One: Initialization

The following figure provides the message flow for Phase One:



EAP-CREDS Phase One Message Flow

[1] Server sends EAP-Request/EAP-CREDS (Type=Init):

After the establishment of the outer mechanism (e.g., EAP-TLS, EAP-TEAP, EAP-TTLS, etc.), the server MAY decide to start a credentials management session. In order to do that, the Server sends an EAP-Request/EAP-CREDS (Type=Init) message to the Peer with one ('Phase-Control') TLV with the 'S' bit set to '1' and the value set to '1' (thus indicating the beginning of Phase One). Also, the Server MAY use one or more ('Version') TLVs to indicate the supported versions.

The Server MAY also specify which versions of EAP-CREDS are supported by adding one or more ('Version') TLVs. If no ('Version') TLV is added to the message, the Peer SHOULD assume the supported version is 1 ('0x1').

[2] The Peer sends EAP-Response/EAP-CREDS (Type=Init)

The Peer, sends back a message that carries one ('Version') TLV to indicate the selected version of EAP-CREDS (i.e. from the list

provided by the server) (optional). If the client does not include the ('Version') TLV, the Server MUST use the most recent supported version of EAP-CREDS. Moreover, the Server includes one or more ('Protocol') TLVs to indicate the list of supported provisioning protocols, followed by one ('Credentials-Info') TLVs for each installed credentials to provide their status to the server (i.e., if multiple credentials are configured on the Peer for this Network, then the Peer MUST include one ('Credentials-Info') TLV for each of them).

The Peer also provides the list of supported Encodings and Formats by adding one or more ('Supported-Encodings') and ('Supported-Formats') TLVs. The Peer MAY also provide the Server with information about the Peer's credentials storage by using the 'Storage-Status' TLV.

When there are no available credentials, the Peer MAY include an authorization token that can be consumed by the Server for registering new credentials. In particular, the Peer can include the ('Token-Data') TLV to convey the value of the token. The ('Challenge-Data') and ('Challenge-Response') TLVs, instead, can be used to convey a challenge and its response based on the authorization information (e.g., maybe a public key hash is present in the Token, then the peer can generate some random data - or use the one from the Server - and generate a signature on that value: the signature SHALL be encoded in the ('Challenge-Response') TLV and it should be calculated over the concatenation of values inside the ('Challenge-Data') TLV and the ('Token-Data') TLV.

Also, the Peer MAY add one or more ('Profile') TLVs to indicate to the Server which profiles are requested/supported (e.g., a pre-configuration MAY exist on the Peer with these ecosystem-specific identifiers).

Ultimately, the Peer MAY include additional metadata regarding the status of the Peer. To this end, the Peer can use a ('Storage-Info') TLV to provide the server with additional data about the Peer's capabilities and resources. Also, the ('Network-Usage') TLV can be used to provide the Server with the indication of which network resources are needed by the Peer and what is its intended utilization pattern(s).

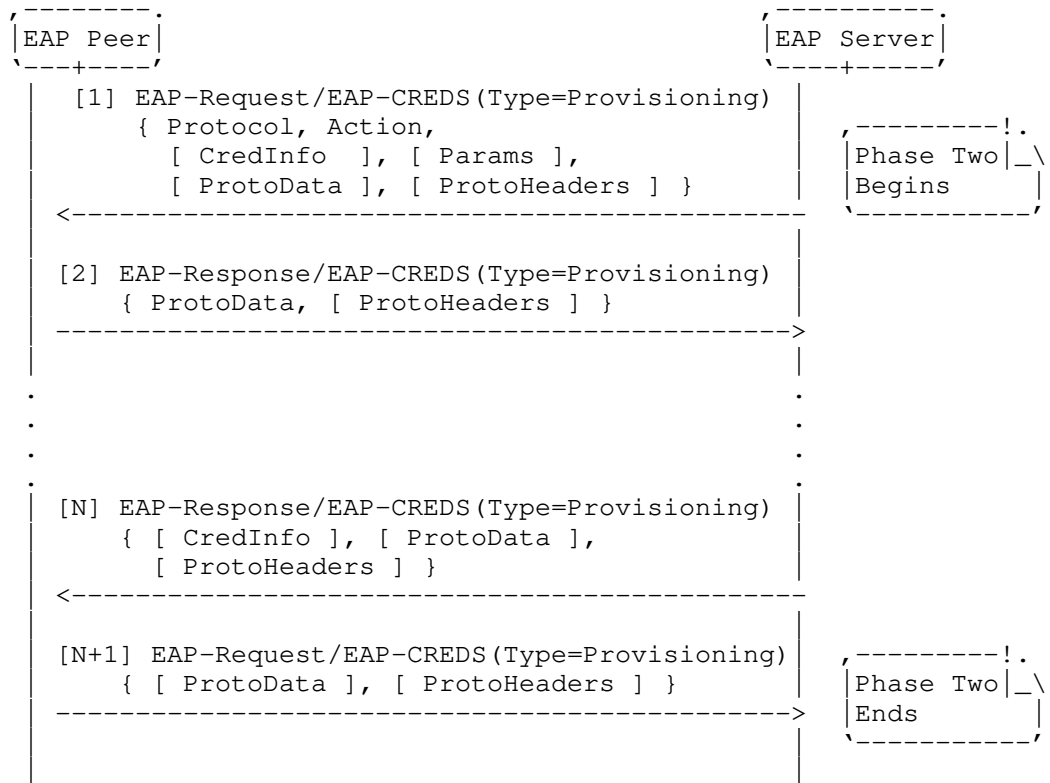
The server checks that the Peer's selected protocol, version, and parameters are supported and, if not (or if the server detects an error), it can (a) send a non-recoverable error message to the peer, notify the outer (tunneling) layer, and terminate the EAP-CREDS session, or (b) start phase one again by sending a new

('EAP-CREDS-Init') message that will also carry an ERROR TLV that provides the Peer with the reason the initial response was not acceptable. In this case, the ('Phase-Control') TLV MUST be omitted since it is not the first message of phase one. The server and the peer can repeat phase one until they reach an agreement or the session is terminated by the Server.

NOTE WELL: The determination of the need to start phase two or not is based on the contents of the ('Credentials-Info') TLV sent by the Peer (e.g., a credential is about to expire or a credential is simply missing).

3.4. Phase Two: Provisioning

The following figure provides the message flow for Phase 2:



EAP-CREDS Phase Two Message Flow

- [1] The Server sends EAP-Request/EAP-CREDS (Type=Init)

The first message of Phase Two indicates that the Server is ready to initiate the selected provisioning protocol.

- [2] The Peer sends EAP-Response/EAP-CREDS (Type=Init)

After that, the Peer sends its first message to the Server by sending the EAP-Response/EAP-CREDS (Type=Provisioning) message. This message contains the selected provisioning protocol's message data and some extra fields (e.g., transport-protocol headers) in the ('Provisioning-Data') and ('Protocol-Headers') TLVs respectively.

- [3] The Server sends EAP-Request/EAP-CREDS (Type=Init)

The Server replies to the Peer's message with EAP-Request/EAP-CREDS (Type=Provisioning) messages until the provisioning protocol reaches an end or an error condition arise (non-recoverable).

- [N] The Server sends EAP-Request/EAP-CREDS (Type=Provisioning)

When the provisioning protocol has been executed for the specific set of credentials, the server sends a last message that MUST include the description of the provisioned credentials in a ('Credentials-Info') TLV and MUST set the 'D' bit in the EAP-CREDS message header to '1' to indicates that the server does not have any more ('Provisioning') messages for this credenital. The final message does not need to be an empty one, i.e. other TLVs are still allowed in the same message (e.g., the 'Provisioning-Data' and the 'Provisioning-Headers' ones).

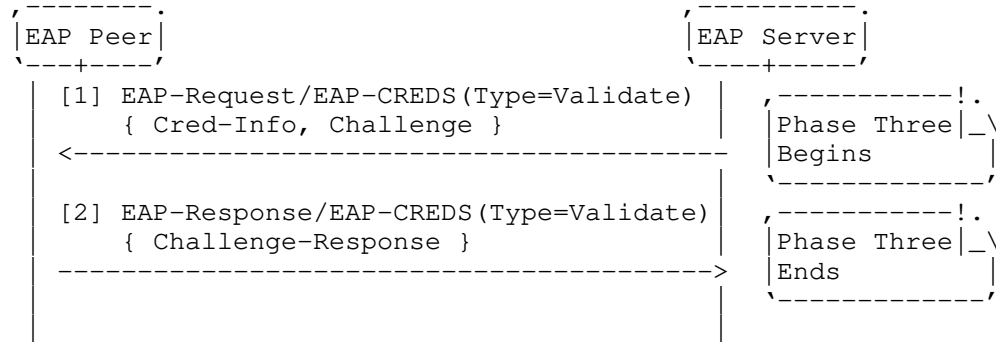
- [N+1] The Peer sends EAP-Request/EAP-CREDS (Type=Provisioning)

The Peer MUST reply to the server with a ('Provisioning') message that MUST have the 'D' bit in the EAP-CREDS message header set to '1', thus indicating that the credentials have been installed correctly. In case of errors, the Peer MUST include the appropriate ('Error') TLV. Also in this case, the final message does not need to be an empty one, i.e. other TLVs are still allowed in the same message (e.g., the 'Provisioning-Data' and the 'Provisioning-Headers' ones).

At this point, the Server can decide to provision (or manage) another set of credentials by issuing a new ('Provisioning') message, or it can decide to start Phase Three by sending its first ('Validate') message, or it can terminate the EAP session.

3.5. Phase Three: Validation

The following figure provides the message flow for Phase 3:



EAP-CREDS Phase Three Message Flow

Phase three is optional and it is used by the server to request the client to validate (proof) that the new credentials have been installed correctly before issuing the final Success message.

NOTE WELL: Phase Three introduces a dependency on the selected hashing algorithm to provide common and easy way to check the integrity and functionality of a newly installed set of credentials.

[1] The Server sends EAP-Request/EAP-CREDS (Type=Validate)

In order to start Phase Three, the Server sends an EAP-Request/EAP-CREDS (Type=Validate) message to the Peer. The Server MUST include the ('Credentials-Info') TLV to provide the indication about which set of credentials the Server intends to validate. The Server MUST also include a randomly generated challenge in the message to the client. The type of challenge determines how the ('Challenge-Response') is calculated. EAP-CREDS defines the asymmetric and symmetric challenges in Section 8.6 and others can be defined according to the specified rules.

As usual, the Server MUST set, in the headers, the 'S' bit to '1' in its first message of Phase Three and the 'Phase' value shall be set to '3' (beginning of Phase Three).

[2] The Peer sends EAP-Response/EAP-CREDS (Type=Validate)

When the client receives the Validate message from the server, it calculates the response to the challenge and sends the response back to the server in a EAP-Response/EAP-CREDS(Type=Validate) message. When the EAP-CREDS-ASYMMETRIC-CHALLENGE and EAP-CREDS-SYMMETRIC-CHALLENGE values are used in the Challenge type, the Peer MUST calculate the response as follows:

Public-Key

For any public-key based credentials (e.g., certificates or raw key pairs), the response to the challenge is calculated by generating a signature over the hashed value of the challenge. The hashing algorithm to be used for this purpose is specified in Section 2.6. The format of the signature in the ('Challenge-Response') TLV is the concatenation of:

- The signatureAlgorithm (DER encoded) which contains the identifier for the cryptographic algorithm used by the Peer to generate the signature. [RFC3279], [RFC4055], and [RFC4491] list supported signature algorithms, but other signature algorithms MAY also be supported. The definition of the signatureAlgorithm is provided in Section 4.1.1.2 of [RFC5280].
- The signatureValue (DER encoded) which contains the digital signature itself. The signature value is encoded as a BIT STRING and the details of how to generate the signatures' structures can be found in Section 4.1.1.3 of [RFC5280] and referenced material.

Symmetric Secret

For any symmetric based credentials (e.g., password or Key), the response to the challenge is calculated by using the selected hash function (see Section 2.6) on the concatenation of (a) the value carried in the server-provided ('Challenge-Data') TLV, and (b) the secret value itself (salted hash).

The initial values for the type of challenges are described in the Section 8.6. Other types of challenges MAY be defined according to the specified procedures.

In case of issues with the validation of newly deployed credentials, both the Server and the Peer should consider those credentials invalid (or unusable) and should issue the required failure message(s).

4. EAP-CREDS Message Format

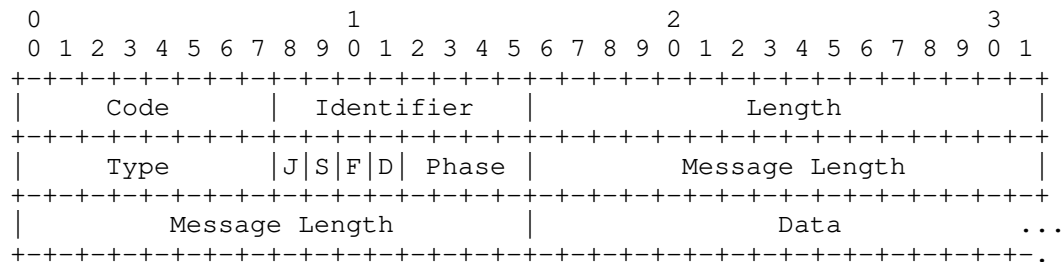
The EAP-CREDS defines the following message types:

1. EAP-CREDS/Init
2. EAP-CREDS/Provisioning
3. EAP-CREDS/Validate

Each of these message types have the basic structure as identified in Section 4.1. EAP-CREDS messages contain zero, one, or more TLVs. The internal structure of the different types of TLVs is described in Section 4.2, while a detailed description of the EAP-CREDS message types is provided in Section 5.

4.1. Message Header

The EAP-CREDS messages consist of the standard EAP header (see Section 4 of [RFC3748]), followed by the version of the EAP-CREDS (4 bits) and a field (4 bits) reserved for future use. The header has the following structure:



Where the Code, Identifier, Length, and Type fields are all part of the EAP header as defined in [RFC3748]. Since EAP-CREDS can only be used as a tunneled mechanism, the presence of these fields is only for backward compatibility with existing parsers. In particular, the 'Length' field is not used (can be ignored): the message length is carried in the 'Message Length' field instead.

The Type field in the EAP header is <TBD> for EAP-CREDS.

The Flags bitfield is used to convey status information (e.g., extra long message, phase number, phase transitioning state). The transition-control bit (i.e., the 'S' bit) are set in Server's messages and are ignored in Peer's messages (the Server is the entity

that unilaterally controls the phase transition process). The meanings of the bits in the 'Flags' field are as follows:

Bit 'J' (Jumbo Message) - If set, it indicates the presence of the 'Message Length' field. This bit SHALL be used only when the size of the message exceeds the maximum value allowed in the 'Length' field. In this case, the 'Message Length' field is added to the message and set to the whole message size and the 'Length' field is used for the current fragment length. If not set, the 'Message Length' field is not present in the Message and the 'Length' field is used for the message size (and the 'F' bit MUST be set to '0').

Bit 'S' (Start) - If set, this message is the first one of a new EAP-CREDS phase. The value of the new phase is encoded in the 'Phase' field.

Bit 'F' - If set, this message is a fragment of a message. In this case, the 'Data' field is to be concatenated with all messages with the 'F' bit set to '1' until the message with the 'F' bit set to '0' that indicates the end of the message. If the message is not fragmented, the 'F' bit MUST be set to '0'. The use of this bit is required when the tunneling method does not provide support for messages up to 2^{32} bits in size.

Bit 'D' - This bit is used in Phase Two and Phase Three to indicate that the specific operation for the identified credential is over. For example, when multiple credentials exist on the Peer and the Server needs to manage and validate one of them. In its last message, when the provisioning protocol is done, the server sets the 'D' (Done) bit to indicate that it is done. The Peer, in its reply, sets the bit to indicate the end of provisioning for this credentials is also over. After that, the Server can continue Phase Two, transition to Phase Three, or terminate the EAP session.

The Phase field is a 4-bits value and identifies the EAP-CREDS phase for the current message. The version of EAP-CREDS described in this document supports three values for this field:

0x01 - Phase One

0x02 - Phase Two

0x03 - Phase Three

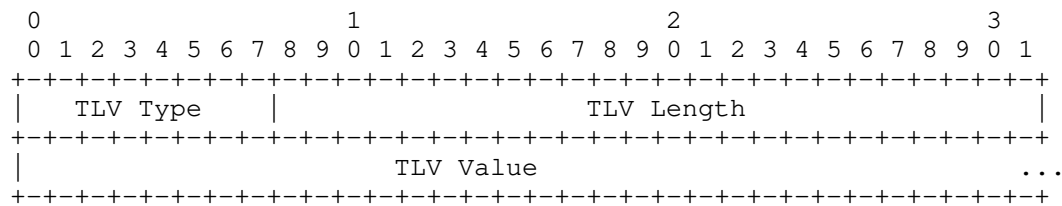
A detailed explanation of the 'Phase' and 'Flags' fields of the message headers is provided in Section 3.2.

The Data field is the message payload. The full description of this field is provided in the next section.

4.2. Message Payload

The Data part of the message is organized as zero, one, or more TLV objects whose structure is defined in this section.

Each TLV object has the same basic structure that is defined as follows:



Where:

TLV-Type (uint8)

This field is used to indicate the type of data that the TLV carries. The type of TLV determines its internal structure. The supported values for this fields are provided in the following table:

Length (uint24)

This field carries the size of the value of the TLV. In particular, the overall size of a TLV (i.e., the header plus the value) can be calculated by adding the size of the header (6 octets) to the value of the Length field (i.e., the size of the TLV's value).

TLV Name	TLV Type	Scope/Usage
<TBD>	Action TLV	Phase Two
<TBD>	Certificate-Data TLV	Phase Two/SPP
<TBD>	Challenge-Data TLV	Phase Two, Phase Three
<TBD>	Challenge-Response TLV	Phase Two, Phase Three
<TBD>	Credentials-Data TLV	Phase Two/SPP
<TBD>	Credentials-Info TLV	Phase Two, Phase Three
<TBD>	Error TLV	All Phases
<TBD>	Network-Usage TLV	Phase One
<TBD>	Profile TLV	Phase Two
<TBD>	Protocol TLV	Phase One, Phase Two
<TBD>	Provisioning-Data TLV	Phase Two
<TBD>	Provisioning-Headers TLV	Phase Two
<TBD>	Provisioning-Params TLV	Phase Two
<TBD>	Certificate-Request TLV	SPP
<TBD>	Storage-Info TLV	SPP
<TBD>	Supported-Format TLV	SPP
<TBD>	Supported-Encoding TLV	SPP
<TBD>	Token-Data TLV	Phase One
<TBD>	Version TLV	Phase One

Table 2: EAP-CREDS Supported TLVs Types

TLV Value (> 1 octet)

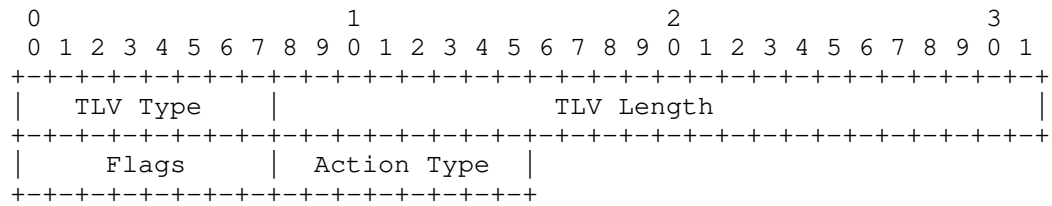
This field carries data for the identified TLV. The internal structure is determined by the TLV Type field.

The rest of this section describes the structure of the different supported TLVs and their usage in the different messages.

4.3. EAP-CREDS defined TLVs

EAP-CREDS messages's payload comprises zero, one, or more TLVs that are encoded in a single EAP-CREDS message. The values for the TLV Type that are supported by this specifications are listed in Table 2.

4.3.1. The Action TLV



TLV Type (uint8)

<TBD> - Action TLV

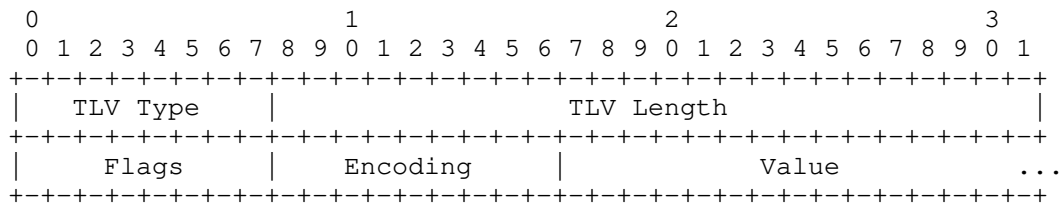
TLV Length (uint24)

Fixed value (=2)

Flags (uint8)

Reserved

4.3.2. The Certificate-Data TLV



TLV Type (uint8)

<TBD> - Certificate-Data TLV

Length (uint24)

Provides the length of the TLV (> 3 octets)

Flags (uint8)

Provides a BITMASK that can be used to provide additional information related to the encapsulated certificate. The bits have the following meaning:

Bit 0 - If set, the certificate is trusted (Trust Anchor)

- Bit 1 - If set, the certificate is a CA certificate
- Bit 2 - If set, the certificate is self-signed
- Bit 3 - If set, the certificate is a proxy certificate
- Bit 4 - If set, the certificate is an attribute certificate
- Bit 5 - Reserved
- Bit 6 - Reserved
- Bit 7 - Reserved

For a Trusted Root CA, the value of the flags shall be 0x7 (0000 0111). For an intermediate CA certificate that is not implicitly trusted, the value of the flags field should be set to 0x02 (0000 0010). For an End-Entity certificate, the value of the Flags will be 0x0 (0000 0000).

Format (uint8)

Provides the indication of the Format the certificate is in. The allowed values for this field are listed in Section 8.5.

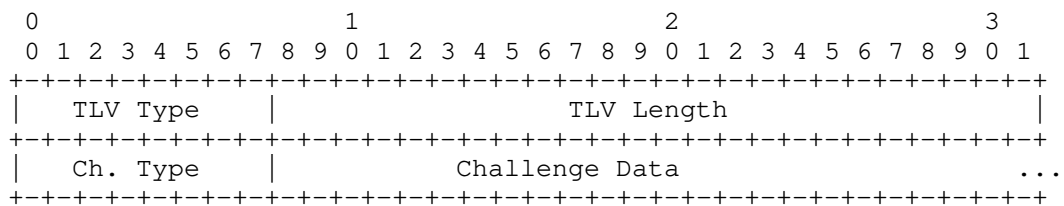
Encoding (uint8)

Provides the indication of the Encoding the certificate is in. The allowed values for this field are listed in Section 8.8.

Value (octet string)

This field carries the data for the certificate.

4.3.3. The Challenge-Data TLV



TLV Type (uint8)

<TBD> - Challenge-Data TLV

Length (uint24)

3 octets

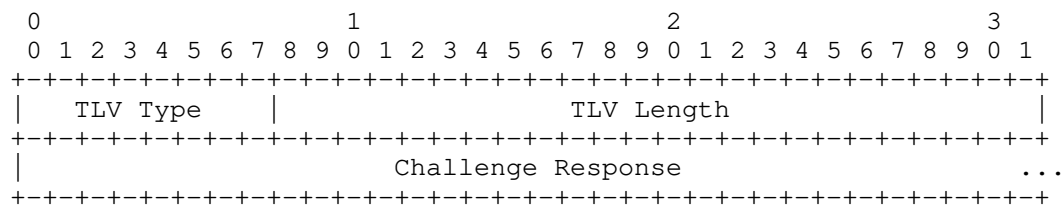
Challenge Type (uint8)

This field carries the type of Challenge. In particular, the challenge type determines how the Peer MUST calculate the ('Challenge-Response'). The initial values for this field are listed in Section 8.6. Please refer to Section 3.5 for a detailed explanation of how to calculate the response to the challenge for the challenge types defined in this document.

Challenge Data (> 1 octet)

This field carries the data to be used as a challenge when validating newly deployed credentials.

4.3.4. The Challenge-Response TLV



TLV Type (uint8)

<TBD> - Challenge-Response TLV

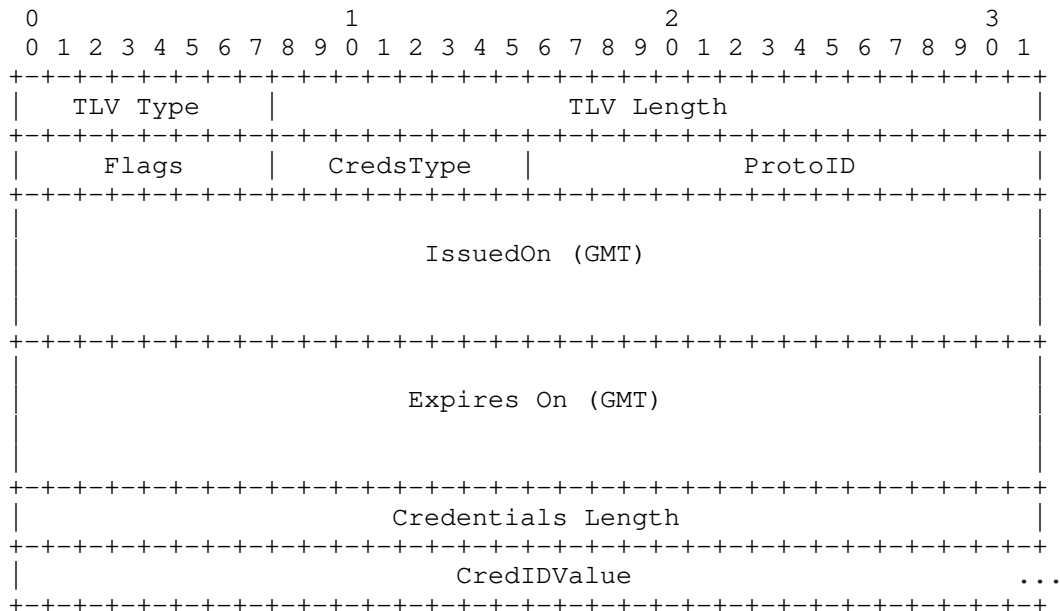
Length (uint24)

3 octets

Challenge Response (> 1 octet)

This field carries the data that resulted from the use of the credentials to be validated.

4.3.5. The Credentials-Information TLV



The Credential-Information TLV is used by the Peer to provide a description of the installed credentials that are relevant for the network that is being accessed.

For example, when a set of credentials need to be renewed, the server checks the ('Credentials-Info') from the Peer and eventually selects the right one for renewal. The TLV structure is as follows:

TLV Type (uint8)

<TBD> - Credentials-Information TLV

Length (uint24)

Provides the total length of the body of the Credential-Information TLV.

Flags (uint8)

Provides a BITMASK that can be used to provide information about the status of the credentials (e.g., if the use marks the

credentials to be compromised). The bits have the following meaning:

- Bit 0 - If set, the credential is marked as compromised
- Bit 1 - If set, the credential is immutable and cannot be updated
- Bit 2 - Private Key or Secret Immutable, the public part of the credential (e.g., a certificate) can still be updated
- Bit 3 - If set, the credential cannot be updated (both public and private parts)
- Bit 4 - If set, the credential is ready to be used
- Bit 5 - If set, the credential was generated on the server
- Bit 6 - If set, the Peer would like to update the credential even if they are not expired
- Bit 7 - Reserved

CredType (uint8)

This field provides the description of the type of credential. The type of credentials are listed in Section 8.3

ProtoID (uint16)

This field indicates the protocol that was used to retrieve the target credential. When the TLV is used in a Request by the Server, this field is ignored. The values for this field are listed in Section 8.1.

IssuedOn (16 octets)

This field carries the GMT date for when this credential was issued. This field is 16 bytes long (the last byte must be set to '0x00') and contains the NULL-terminated ASCII string that represents the timestamp where the credential was issued. When the value is not set, the field should be set to { 0x00 }. The format of the string is as follows:

YYYYMMDDHHmmssZ

Where:

YYYY - is the 4 digits representation of the year

MM - is the 2 digits representation of the month

DD - is the 2 digits representation of the day of the month

HH - is the 2 digits representation of the hour of the day (24 hour format)

mm - is the 2 digits representation of the minutes of the hour

ss - is the 2 digits representation of the seconds of the minute

Z - is the character 'Z'

ExpiresOn (16 octets)

This field carries the GMT date for when this credential is to be considered expired. This field is 16 bytes long (the last byte must be set to '0x00') and contains the NULL-terminated ASCII string that represents the timestamp where the credential was issued. The format is the same as the ('IssuedOn') field. When the value is not set, the field should be set to { 0x00 }.

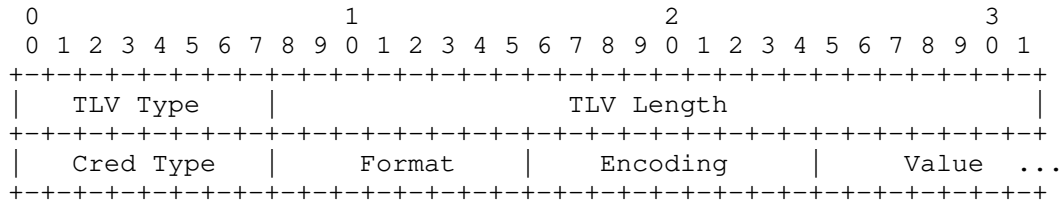
Credentials Length (uint16)

Length (in bytes) of the Credentials value. When used with a public-key type of credentials, this is the size of the key (e.g., for an RSA 2048 bit keys, this field should carry the value of 256). When used with a symmetric secret, this field carries the size of the secret (in bytes).

CredIDValue (> 1 octet)

The binary value of the credentials' identifier. This identifier can be the binary value of the SHA-256 calculated over the certificate, a username, or it could be a random handle. As long as the ID allows the peer and the server to uniquely (in its context) identify the credentials, the value of this field can be calculated in any way.

4.3.6. The Credentials-Data TLV



TLV Type (uint8)

<TBD> - Credentials-Data TLV

Length (uint24)

Provides the length of the TLV (> 3 octets)

Cred Type (uint8)

Provides the indication of the type of credentials. The allowed values for this field are listed in Section 8.3.

Format (uint8)

Provides the indication of the Format the credentials are in. The allowed values for this field are listed in Section 8.5.

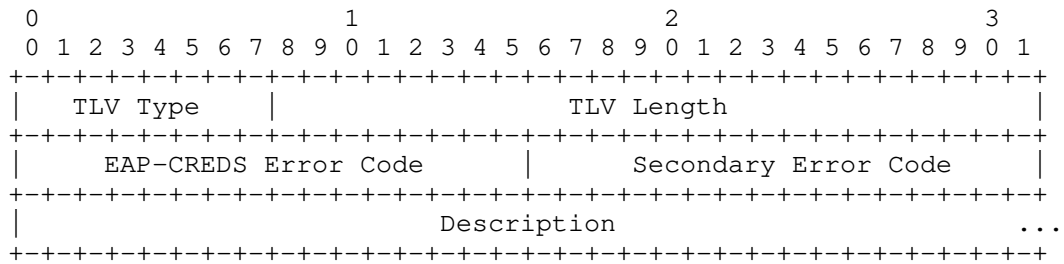
Encoding (uint8)

Provides the indication of the Encoding the credentials are in. The allowed values for this field are listed in Section 8.8.

Value (octet string)

This field carries the data for the credentials.

4.3.7. The Error TLV



TLV Type (uint8)

<TBD> - Challenge-Response-Data TLV

Length (uint24)

3 octets

EAP-CREDS Error Code (2 octets)

This field carries the EAP-CREDS error code. These code are related to the EAP-CREDS operations only and it should not be used to carry the Provisioning-Protocol specific error codes.

The error codes supported by this specifications are listed in Section 4.3.7.

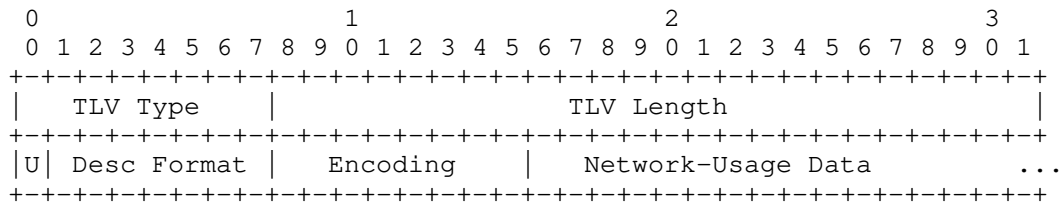
Secondary Error Code (2 octets)

This field is used to convey an error at the encapsulation layer (i.e., the provisioning protocol error). For example, this field can be used to convey a transport protocol error code (e.g., HTTP status code). Do not use this field to convey EAP-CREDS specific errors.

Description (> 1 octet)

The Description field is optional (i.e., when the Description Size is set to zero) and carries information about the error that occurred. The message may or may not be used by a user or an automated process for debugging purposes.

4.3.8. The Network-Usage TLV



TLV Type (uint8)

<TBD> - Network-Usage TLV

Length (uint24)

Variable Length TLV (Value must be > 2)

Description Format (uint8)

The Type of data encoded in the Peer Description Data. The initial values for this field are listed in Section 8.10.

Encoding (uint8)

Provides the indication of the Encoding the network usage description data is in. The allowed values for this field are listed in Section 8.8.

The 'U' field (1 bit)

The 'URL' bit ('U') is used to indicate if the value of the Network-Usage Data field is to be interpreted as a URL or as the actual data. In particular, if the value in the 'URL' bit is '1', then the value in the Network-Usage Data field is to be interpreted as the URL where the actual data can be downloaded from. Otherwise, if the 'URL' bit is set to '0', then the value in the Network-Usage Data field is to be interpreted as the actual data (not a URL referencing it).

An example use of this bit is when the Peer wants to convey the URL of the MUD file [RFC8520]. In this case, the Peer can set the Network-Usage Data field to the Url of the MUD file related to the Peer.

Desc Format (7 bits)

This field provide the expected data format for the Network-Usage Data. For example, the value in this field could be set to 'MUD'

and have the 'U' bit set to '1' to provide the MUD-related information at credentials management time instead of at network-provisioning time (DHCP option). This possibility could help the Network controller to decide if the device shall be allowed to register its credentials or not.

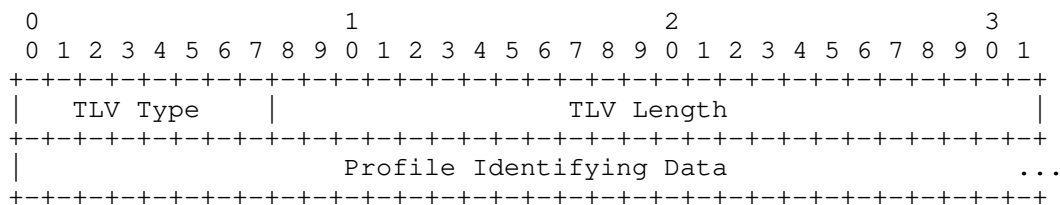
The list of initial values for this field is provided in Section 8.7.

Network-Usage Data (octet string)

This is additional information related to the device. In particular, this TLV can be used by the Peer to provide the Server with the description of the intended network usage or a URL that points to the same information.

For example, this field can be used to convey a MUD file (Manufacturer Usage Description) or the latest firmware-update manifest.

4.3.9. The Profile TLV



TLV Type (uint8)

<TBD> - Profile Identifying Data TLV

Length (uint24)

Length value should be ≥ 1

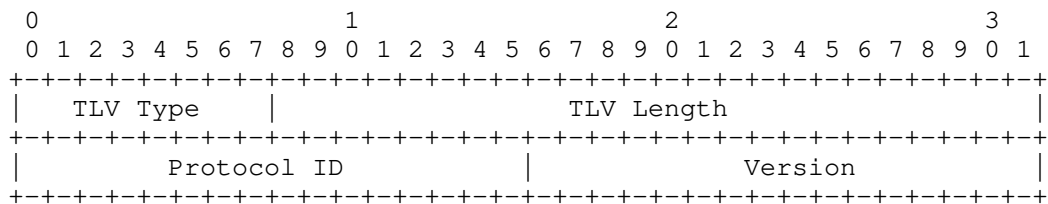
Profile Identifying Data (octet string)

The Profile Identifying Data is used to provide indication to the other party about which profiles are supported when requesting credentials management.

Also in this case, the data used in this field is left to be interpreted by the end-point and it is orthogonal to EAP-CREDS data types.

An example of values for this field, an end-point could use the string representation (i.e., dotted representation) of the Object Identifier (OID) of the specific profile supported (e.g., could be defined in the Certificate Policy of the credentials' provider).

4.3.10. The Protocol TLV



TLV Type (uint8)

<TBD> - Protocol TLV

TLV Length (uint24)

Fixed TLV Length value of 4.

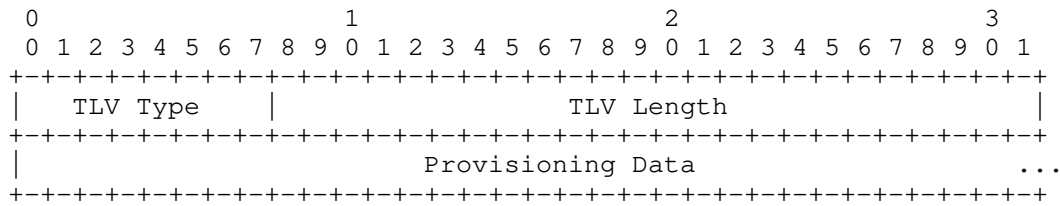
Protocol ID (uint16)

The Protocol ID value carries the id of a supported provisioning protocol. The initial list of values for the provisioning protocol identifiers can be found in Section 8.1.

Version (uint16)

The Version (Protocol Version) value represents the specific version of the identified provisioning protocol. When no version is specified for a protocol (i.e., either it does not support multiple versions or it does not matter), the value of this field should be set to '0x0'.

4.3.11. The Provisioning-Data TLV



TLV Type (uint8)

<TBD> - Provisioning-Data TLV

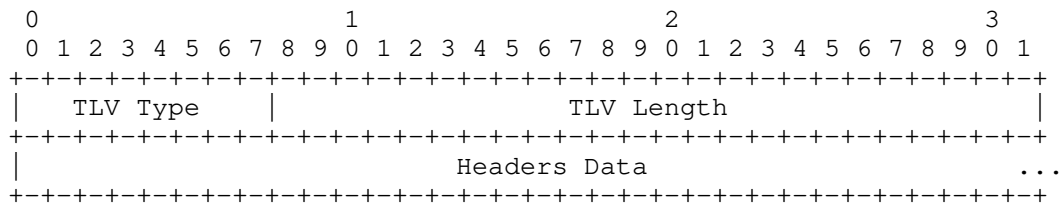
Length (uint24)

3 octets

Headers Data (> 1 octet)

This field carries the provisioning protocol's messages.

4.3.12. The Provisioning-Headers TLV



TLV Type (uint8)

<TBD> - Provisioning-Headers TLV

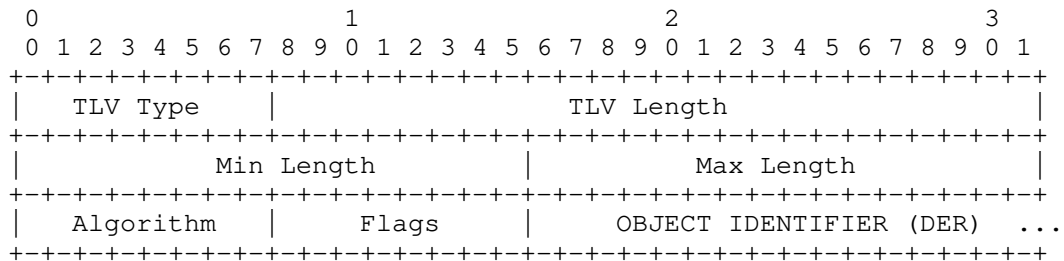
Length (uint24)

3 octets

Headers Data (> 1 octet)

This field carries the meta-data (if any) that might be associated with the transport-layer normally used with the provisioning protocol. For example, this TLV can carry the set of HTTP headers required by EST or ACME.

4.3.13. The Provisioning-Params TLV



TLV Type (uint8)

<TBD> - Provisioning-Params TLV

Length (uint24)

Provides the length of the TLV (≥ 6 octets)

Min Length (uint16)

Provides the minimum allowed size size for the credentials. This value has meaning depending on the context of the credentials, however sizes are always expressed in bytes.

For example, when used with a symmetric key or a password, the ('Min Length') and ('Max Length') refer to the minimum and maximum size of the password data. The ('Algor OID') field can be omitted in this case.

On the other hand, when referring public-key credentials, this field should carry the size of the modulus of the key. For example, for an RSA 2048 bit keys, the field should carry the value of 256. For an ECDSA that uses the prime256r1 curve, this field should carry the value of 32 and the Algor OID should be the DER representation of the specific value of the curve (i.e., the DER representation of '1.2.840.10045.3.1.7').

Max Length (uint16)

Provides the indication maximum size of the credentials. This value has meaning depending on the context of the credentials, however sizes are always expressed in bytes.

The same considerations apply to this field as well as the ('Min Length') one discussed above.

Flags (uint8)

Provides a BITMASK that can be used to provide information about the status of the credentials (e.g., if the use marks the credentials to be compromised). The bits have the following meaning:

Bit 0 - Credentials (or part of it) are to be generated on the server

Bit 1 - Credentials (or part of it) are to be generated on the peer

Bit 2 - Credentials are to be generated on dedicated hardware

Bit 3 - Reserved

Bit 4 - Reserved

Bit 5 - Reserved

Bit 6 - Reserved

Bit 7 - Reserved

When using public-key based credentials, the bits 0 and 1 are mutually exclusive.

When using passwords or shared secrets, if bit 0 is set, then the secret is generated by the server and then sent to the client. On the other hand, if bit 1 is set, then the secret is generated by the peer and then sent to the server. Ultimately, if both bits are set, then the Server generates the first part of the password and sends it to the Peer, while the Peer generates the second part of the password and sends it to the Server. The password to be used for future authentication is the concatenation of the two shares of the password: first the one from the Server, then the one from the Client.

NOTE WELL: Last but not least, since these passwords/secrets are meant to be used in a automated fashion, there is no restriction around the character set to use or their interpretation. Therefore, it is good practice to generate

random passphrases that use the full 8-bit character set (on client and server) to maximize the secret's search space.

Algorithm (uint8)

Provides the indication of the algorithm used for the generation of the credentials. The allowed values for this field are listed in Section 8.4.

Object Identifier (binary; > 1 octet)

Provides the indication of additional parameters that are needed to be encoded for the credentials. This value is used only when the credentials use public-key cryptography - this field carries additional information about the generation algorithm to be used. We provide some useful values that can be used as reference:

OID Name	Dotted Representation	Binary Encoding
secp256r1 curve	1.2.840.10045.3.1.7	06 08 2A 86 48 CE 3D 03 01 07
secp384r1 curve	1.2.840.10045.3.1.34	06 08 2A 86 48 CE 3D 03 01 22
secp521r1 curve	1.2.840.10045.3.1.35	06 08 2A 86 48 CE 3D 03 01 23
X25519 curve	1.3.101.110	06 03 2B 65 6E
X25519 curve	1.3.101.110	06 03 2B 65 6E
X448 curve	1.3.101.111	06 03 2B 65 6F
Ed25519 curve	1.3.101.112	06 03 2B 65 70
Ed448 curve	1.3.101.113	06 03 2B 65 71

Table 3: Object Identifiers Examples

4.3.14. The Certificate-Request TLV

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								
TLV Type										TLV Length																													
Encoding										Format										Value										...									

TLV Type (uint8)

<TBD> - Token-Data TLV

TLV Length (uint24)

Provides the length of the TLV (> 3 octets)

Encoding (uint8)

Provides the indication of the Encoding the credentials are in.
The allowed values for this field are listed in Section 8.8.

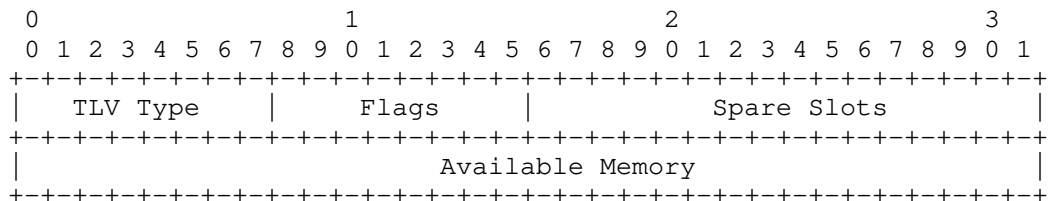
Format (uint8)

Provides the indication of the type of credentials. The allowed values for this field are listed in Section 8.5.

Value (octet string)

This field carries the data for the credentials.

4.3.15. The Storage-Info TLV



TLV Type (uint8)

<TBD> - Store-Info TLV

Flags (8 bits)

Provides information about the status and type of store and limited information about its capabilities. The bits have the following meaning:

Bit 0 - If set, the store supports RSA keys (software)

Bit 1 - If set, the store supports RSA keys (hardware)

- Bit 2 - If set, the store supports ECDSA keys (software)
- Bit 3 - If set, the store supports ECDSA keys (hardware)
- Bit 4 - If set, the store supports symmetric keys
- Bit 5 - If set, the store supports generic tokens
- Bit 6 - If set, the store is immutable (no key generation or deletion)
- Bit 7 - Not Used

Spare Slots (uint16)

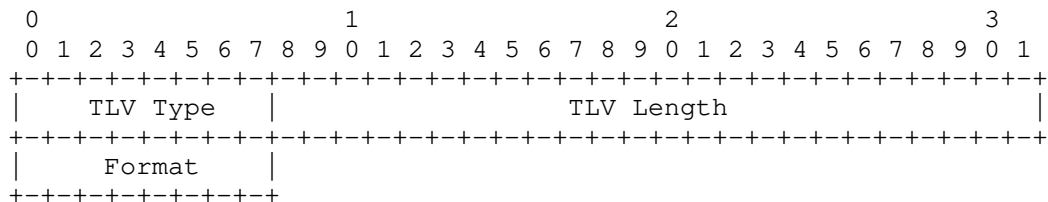
Provides the number of available slots where to store credentials. When no more slots are available, the value of '0' should be used to indicate to the Server that a credential must be deleted before a new one can be created.

When the number of slots is not fixed or not known, the value of { 0xFF, 0xFF } shall be used.

Available Memory (uint32)

This field carries the size (in bytes) of the spare memory on the Peer's secrets' store.

4.3.16. The Supported-Formats TLV



TLV Type (uint8)

<TBD> - Supported-Format TLV

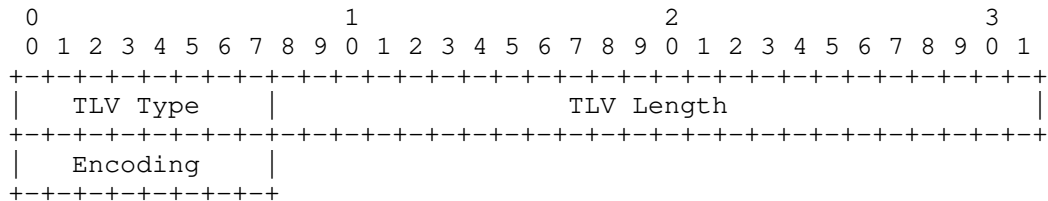
TLV Length (uint24)

Provides the length of the TLV. This field must be set to 1.

Format (uint8)

Provides the details about the supported format. Multiple formats TLVs can be used in the Peer's ('Init') message to provide the Server with the Peer's capabilities.

4.3.17. The Supported-Encoding TLV



TLV Type (uint8)

<TBD> - Store-Info TLV

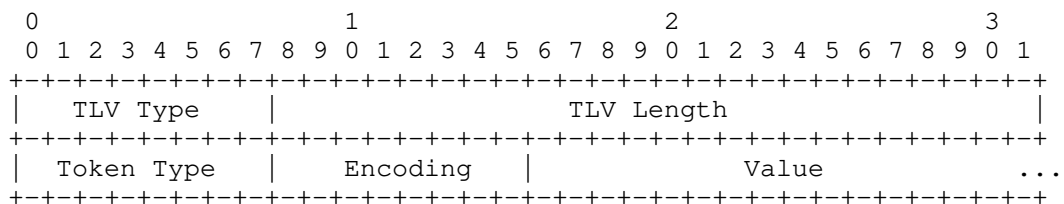
TLV Length (uint24)

Provides the length of the TLV. The field has a fixed value of 1.

Encoding (uint8)

Provides the indication of the supported Encoding by the End Point. This provides the indication to the Server of the capability of the Peer. The allowed values for this field are listed in Section 8.8.

4.3.18. The Token-Data TLV



TLV Type (uint8)

<TBD> - Token-Data TLV

TLV Length (uint24)

Provides the length of the TLV (> 3 octets)

Token Type (uint8)

Provides the indication of the type of credentials. The allowed values for this field are listed in Section 8.2.

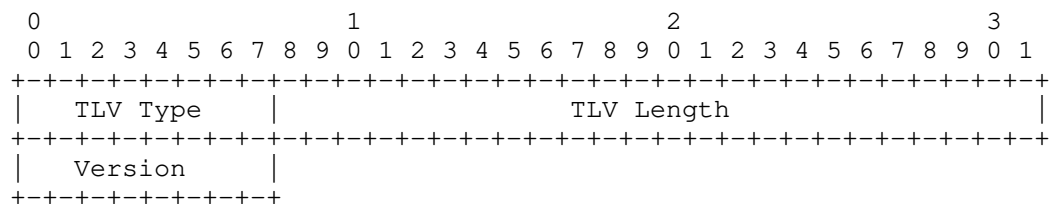
Encoding (uint8)

Provides the indication of the Encoding the credentials are in. The allowed values for this field are listed in Section 8.8.

Value (octet string)

This field carries the data for the credentials.

4.3.19. The Version TLV



TLV Type (uint8)

<TBD> - Version TLV

TLV Length (uint24)

Provides the length of the TLV. The field has a fixed value of 1.

Version (uint8)

The Version field represents the specific version of the EAP-CREDS protocol that are supported by the end point. When multiple versions of EAP-CREDS are supported, multiple ('Version') TLVs can be used.

When no version is specified (i.e., either it does not support multiple versions or it does not matter), the value of this field should be set to '0x0' (any version).

5. EAP-CREDS Messages

This section describes each message and what TLVs are allowed or required. EAP-CREDS defines the following values for the Message Type (Type):

Message Type	Name	Description
0	EAP-CREDS-Init	Initialization Phase
1	EAP-CREDS-Provisioning	Carries Provisioning Protocol Messages
2	EAP-CREDS-Validate	Validates newly installed credentials

Table 4: EAP-CREDS Message Types

5.1. The EAP-CREDS-Init Message

The EAP-CREDS-Init message type is used in Phase One only of EAP-CREDS. The message flow is depicted in Section 3.3. This message supports the following TLVs: Version, Protocol, Credentials-Info, and Error.

5.1.1. EAP Server's Init Message

EAP-CREDS starts with an ('EAP-CREDS-Init') message from the server. This message MAY contain zero, one, or more ('Version') TLVs and, optionally, a ('Challenge-Data') TLV.

The first message from the server is the one that starts Phase One, therefore the Server MUST set the headers' 'S' bit to '1' (Start) and the headers' 'Phase' value to '1' (Phase One).

The Server uses one or more ('Version') TLVs in the EAP-Request/EAP-CREDS(Type=Init) message to provide the Peer with the list of EAP-CREDS versions supported. If omitted, the implicit version of EAP-CREDS used in the session is one ('0x1'). If the Server detects multiple occurrences of this TLV in the reply from the Peer, an error shall be issued and the EAP-CREDS session should be terminated.

In case Token-Based registration is enabled on the Server, the Server MUST include, in its Init message, a ('Challenge-Data') field that can be used by the client to provide challenge data for proof-of-possession of secrets.

5.1.2. EAP Peer's Init Message

The Peer MUST reply to the Server's ('EAP-CREDS-Init') message with its own ('EAP-CREDS-Init') one. The Peer SHOULD include one ('Version') TLV in its first message to indicate the version of EAP-CREDS that the client wants to use for the session. The Peer MUST also provide the list of supported provisioning protocols (via one or more the 'Protocol' TLV), the list and status of the installed credentials (via the 'Credentials-Info' TLV). The Peer MAY include authorization data when registering new credentials (e.g., an authorization token or a device certificate) via the ('Token-Data') and ('Challenge-Response') TLV.

The Peer MUST include one ('Credentials-Info') TLV for each credential the Network is authorized to manage. Typically, a Peer will include only one ('Credentials-Info') TLV in its ('EAP-CREDS-Init') message, but there might be cases where multiple types of credentials are available and selected depending on the location and other factors (e.g., X.509 certificate and username/password combination).

In case the Peer does not have any credentials available yet, it does not add any ('Credentials-Info') TLV - leaving the Server with the only action possible: Registration. In this case, the Peer SHOULD include authorization information via the ('Token-Data') TLV as described in Section 5.1.2.1. Additionally, the Peer can add the ('Profile') TLV to indicate a preferred profile for the credentials.

5.1.2.1. Bootstrapping Peer's Trustworthiness

When the Peer does not have any valid credentials for the Network that it is authenticating to, it does not provide any ('Credentials-Info') TLV. This indicates to the Server that new credentials MUST be registered before the Peer is allowed on the network.

The Registration process might rely on information exchanged during the Provisioning Process in Phase Two. However, if an authorization mechanism is not available from the supported provisioning protocol and no credentials are available on the Peer, EAP-CREDS provides a simple mechanism for the Peer to leverage an out-of-band token/passphrase/ott that may be already available on the Peer (e.g., a device certificate or a 'spendable' credentials token like a kerberos ticket or a crypto-currency transaction) and that can be verified by the Server.

In particular, when the Peer wants to register new credentials (and the Server requires the use of additional authorization data) it may need to provide (a) a Token, (b) a challenge value, and (c) a

response to the challenge value. To do so, the Peer MUST encode the token in a ('Token-Data') TLV, the challenge value in a ('Challenge-Data') TLV, and, finally, the response to the challenge in the ('Challenge-Response') TLV.

The use of ('Challenge-Data') and ('Challenge-Response') TLVs is optional, however it is suggested that if a token is used for bootstrapping the trust, it should provide a way to verify a secret associated with it.

It is also very important that the authorization token is disclosed only to authorized servers - the Peer MUST NOT disclose authorization tokens that are not meant for the network that is being accessed. This can be done, usually, by verifying the identity of the Server first (in the outer mechanism) and then verify that the target of the Token is the Server the Client is talking to.

5.1.3. The EAP-CREDS-Provisioning Message

The EAP-CREDS-Provisioning message type is used in Phase Two only of EAP-CREDS. The message flow is depicted in Section 3.4. This message type supports the following TLVs: Protocol, Profile, Credentials-Info, Provisioning-Headers, Provisioning-Data, Token-Data, and Error.

After the exchange of phase one messages, the Server MAY start phase two by issuing an ('EAP-CREDS-Provisioning') message for the Peer where it encodes all the required details for starting the provisioning process. In particular, the server sends the selected ('Action'), ('Protocol'), and metadata to the client in a EAP-Request/EAP-CREDS(Type=Provisioning) message. The header's 'S' bit MUST be set to '1' (Start) and the 'Phase' value set to '2' (Phase Two begins).

NOTE WELL: After the initial message, the only TLVs that are allowed in messages coming from the server are the usual ('Provisioning-Headers') ('Provisioning-Data'), and ('Error').

The client checks that all the selected parameters are supported for the selected credentials and, if no errors are detected, it sends its first ('EAP-CREDS-Provisioning') message to the Server with the ('Provisioning-Headers') and ('Provisioning-Data') TLVs only.

From now on, the conversation between the Peer and the Server continues until an error is detected or the provisioning protocol completes successfully.

If no other actions, the server MAY continue with phase three or issue a success message and terminate the EAP session.

NOTE WELL: When the SPP protocol is used, the protocol messages that are encoded inside the ('Protocol-Data') TLV are composed of sets of TLVs as defined in this document. The overall message size is provided by the size of the ('Protocol-Data') TLV that encapsulates the SPP-specific TLVs. This design choice provides symmetry in implementing support for SPP when compared to other provisioning protocols.

5.1.4. The EAP-CREDS-Validate Message

The EAP-CREDS-Validate message type is used in Phase Three only of EAP-CREDS. The message flow is depicted in Section 3.5. This message type supports the following TLVs: Protocol, Credentials-Info, Provisioning-Headers, Provisioning-Data, Token-Data, and Error.

After Phase One (and/or Phase Two) ends, the Server MAY start phase three by issuing an ('EAP-CREDS-Validate') message for the Peer where it encodes all the required details for starting the validation process. In particular, the server sends the ('Credentials-Info'), a ('Challenge'), and the ('Phase-Control') TLVs in a EAP-Request/EAP-CREDS(Type=Validate) message. The ('Phase-Control') TLV should carry the '1' value for the 'S' bit (Start) and the number '3' for its value (Phase Three begins).

The Peer generates the answer to the Challenge and sends back a EAP-Response/EAP-CREDS(Type=Validate) message with the ('Challenge-Response') and an optional ('Challenge') field (only for server-side validation of the symmetric credentials). If the Peer requested server-side validation of the credentials, the Server MUST include (if a symmetric secret) the response to the Peer-issued ('Challenge') TLV by computing the response and adding it to the ('Challenge-Response') TLV in its reply.

Finally, in the last message, the Server (if Phase Three is to be ended) SHALL include the ('Phase-Control') TLV with the 'S' bit set to '0' (end of phase) and the value set to '3' (Phase Three ended).

At this point, EAP-CREDS has terminated all possible operations and can be terminated. The Server can now terminate the EAP session successfully. In case the Peer was not authenticated during the tunnel establishment (i.e., no credentials were already available on the Peer), the Server should terminate the EAP session with a Failure (thus requiring the device to re-attach and authenticate to the network - phase two should have provided the Peer with the credentials to use for authenticating to the Network).

6. Error Handling in EAP-CREDS

This section provides a description of the error handling by using the CREDSError-TLV in a CREDSError message.

7. The Simple Provisioning Protocol (SPP)

EAP-CREDS supports a Simple Provisioning Protocol (SPP) which comprises of a series of messages that enable the management not only of certificates, but also of other types of credentials like username/password pairs, asymmetric keys, and symmetric keys.

The Simple Provisioning Protocol (SPP), described in this section, behaves as any other provisioning protocol: its messages are encapsulated in the ('Provisioning-Data') TLVs in the second phase of the protocol. SPP does not make use of any ('Provisioning-Headers') TLVs because its messages are all self-contained (no transport-protocol specific options are needed).

When no ('Credentials-Info') TLVs have been provided by the client, the Server knows that the device does not have valid credentials it wants to use to access the Network. In this case, EAP-CREDS/SPP supports the use of Tokens to kick-off the registration process. The type, format, or encoding of the Token is orthogonal to EAP-CREDS/SPP which treats the token as a black-box field (i.e., it SHOULD NOT try to interpret or parse its contents).

NOTE WELL: During Phase One, the Peer MAY include the ('Token-Data') TLV in its EAP-CREDS-Init message to provide the needed authorization to register a new set of credentials. The Server might not allow the registration of new credentials if the required authorization (i.e., the Token) was not provided during the initialization phase.

In the case where an authorization token is used, different usage patterns are supported. For tokens that require an associated verifiable proof-of-possession, the Peer can include a ('Challenge-Response') TLVs.

The ('Challenge-Data') TLV provided by the Server MUST be used to convey the challenge data (usually some random value) to compute the contents of the ('Challenge-Response') TLV.

The ('Challenge-Response') TLV is used, instead, to encode the response to the challenge data. The ('Challenge-Response') TLV is generated by the Peer and verified by the Server. At minimum, the ('Challenge-Response') TLV SHOULD be calculated over the values of

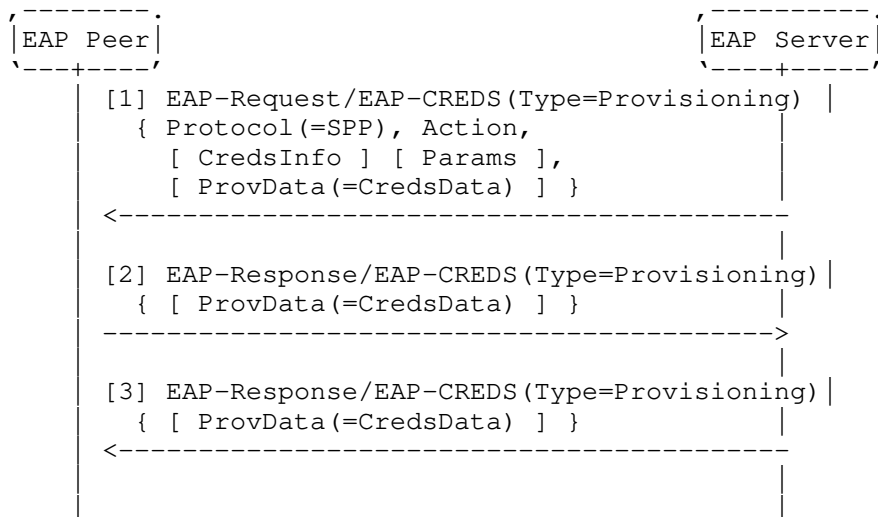
the ('Token-Data') and the ('Challenge-Data') TLVs to make sure that the authentication covers the token's data as well.

NOTE WELL: The use of the ('Token-Data'), ('Challenge-Data'), and ('Challenge-Response') TLVs in the Peer's Init message is be used only to bootstrap trust between the Server and the Peer. If the Server accepts the authorization information as valid, the Peer is enabled for registering new credentials. This should happen only when the Peer does not have valid credentials or when the server wants to provision a different type of credentials (i.e., Action=(Register)). Other methods to provide authorization information might be provided by the selected provisioning protocol: in this case, the Server MAY enable registration of new credentials when no authorization data is provided in the 'Init' message from the client and delegate the validation of the authorization data during Phase Two.

7.1. SPP Message Format

The SPP Messages are constructed with zero, one, or more TLVs and encoded in the ('Provisioning-Data') TLV in EAP-CREDS/Provisioning message types. The size of the encapsulating ('Provisioning-Data') TLV provides the size of the whole message.

7.2. SPP Message Flow



SPP was designed to provide an easy alternative to more complex provisioning protocols. When no extra flexibility is needed, SPP provides an easy-to-implement alternative that can handle not only certificates, but also symmetric secrets and access tokens provisioning. In this section we provide the generic flow of messages for SPP and specific examples for certificates, username/password, and token provisioning.

EAP-CREDS defines several actions for a set of credentials and they are listed in Section 8.9.

When a Peer wants to join a network it may or may not have the needed credentials to do so. In case the Peer does not have valid credentials yet, the Server MAY start Phase Two with the intention of registering a new set of credentials. Alternatively, the Server MAY start Phase Two when the presented credentials information from the Peer triggers the Renew or the Remove action.

[1] The Server sends EAP-Request/EAP-CREDS (Type=Provisioning)

When registering new credentials, the first message from the Server, MUST not carry a ('Credentials-Info') TLV since there is no targeted credentials to apply the action on (i.e., for other actions - like 'renew' or 'remove' - the TLV would be required to identify the right set of credentials to renew or delete).

In SPP, the Server sets the ('Protocol') TLV to SPP, the ('Action') TLV to 'Register', 'Renew', or 'Remove'. When provisioning (or registering) new credentials for the Peer, the Server also sets the ('Provisioning-Params') TLV (or Params) to the type of credentials to be provisioned. The Server also sets any relevant constraints, and, optionally, the ('Profile') TLV.

NOTE WELL: If the Peer is authorized to register a new set of credentials, then the first message from the Server will have the ('Action') TLV set to 'register' and no ('Credentials-Info') TLV is present in the Server's message. In case server-side generation is used, an additional ('Credentials-Info') TLV MAY be encoded inside the ('Provisioning-Data') TLV.

If the type of credentials is symmetric and the parameters call for server-side generation of a symmetric key share, the Server MUST also include its own generated share in a ('Credentials-Data') TLV inside the ('Provisioning-Data') one (the data for the provisioning protocol are encapsulated in the 'Provisioning-Data')

TLV for any protocol used during Phase Two - SPP is no exception to this rule).

In case Server-side only is selected, the Server MUST send the new credentials in its message and include the ('Credentials-Info') TLV. If no other credentials need to be managed, the Server MUST end Phase Two by setting the appropriate bits in the EAP-CREDS headers as well.

[2] The Peer sends EAP-Response/EAP-CREDS (Type=Provisioning)

When Peer-generation is selected (either Peer-only or combined Peer and Server side) and Phase Two has not terminated yet, the Peer MUST reply to the Server's message with its own 'Provisioning' response. The response MUST carry either (a) its own generated share of the key in a ('Credentials-Data') TLV (if the credentials that are provisioned are symmetric and the configuration calls for a share of the key to be provided by the Peer) or (b) a PKCS#10 request in a ('Certificate-Request') TLV (also in this case, only if client-side generation was enabled by the Server) that is generated by using the parameters provided by the Server in the ('Provisioning-Params') TLV.

[3] The Server sends EAP-Request/EAP-CREDS (Type=Provisioning)

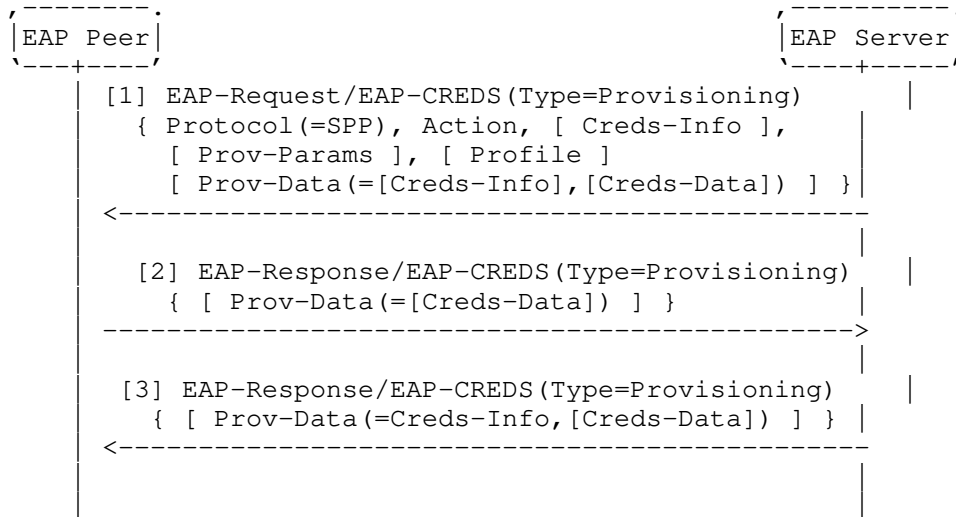
The last message of SPP is from the Server and it is used to deliver the finalized value of the credentials and/or associated metadata. In case the credentials being provisioned are Certificate-based, the Server MUST include the issued certificate in its reply. The issued credentials shall be encoded in a ('Credentials-Data') TLV inside the ('Provisioning-Data') one. In case that the selected format supported/selected by the Peer and the Server does not provide the possibility to encode the full chain (i.e., intermediate and Root CAs) in the response, the Server MUST add one ('Certificate-Data') TLV for each certificate in the chain (including the Root CA's certificate).

The Server MUST include the ('Credentials-Info') TLV in its message. This provide the Peer with some additional data (e.g., the 'Profile' or the 'Identifier' associated with the credentials that were provisioned/managed).

In the case where additional credentials need to be managed, the Server can continue Phase Two by issuing a new [1] message where the tuple Action/Credentials must be unique for the current EAP-CREDS session.

The Server can now decide to start Phase Three (suggested if new credentials were provisioned or renewed) or to terminate the EAP session successfully.

7.2.1. SPP Symmetric Secrets Management



EAP-CREDS/SPP can provision symmetric secrets (e.g, username/password, API keys, or SIM-based keys), tokens (e.g., username/password OAuth or Kerberos tokens), or asymmetric credentials (e.g., X.509 certificates or Key Pairs). This section focuses on provisioning symmetric secrets only. The message flow is provided in Section 7.2.1

EAP-CREDS/SPP provides the possibility for shared secret to be generated in different ways:

1. Server-Side Generated
2. Client-Side Generated
3. Both Client-Side and Server-Side Generated

In particular, when initiating the second phase of the protocol, the ('Provisioning-Params') TLV is used to specify how to generate the secret (see Section 4.3.13).

7.2.1.1. Server Side Only Generation

[TO BE EDITED]

Figure 1: SPP Message Flow for Server-Side only secrets provisioning

The message flow for deploying a server-side only credential (i.e., during registration or renewal) consists of only one message from the server. The flow is depicted in Figure 1.

In this case, the Server sends the first Provisioning message (which is also the last one), which MUST carry, the following data:

- o The ('Credentials-Info') TLV that specifies the info for the provisioned secret, and
- o The ('Protocol') TLV that specifies the provisioning protocol to be used, and
- o The ('Action') TLV that provides the action to be performed ('Registration') or ('Renew'), and
- o The ('Provisioning-Params') TLV that provides the generation parameters to the Peer, and

The Server also includes, encoded in the ('Provisioning-Data') TLV, the following data:

The ('Credentials-Info') TLV that provides the metadata associated with the generated secret

The ('Credentials-Data') TLV that provides the secret that is provisioned to the Peer

Server-side secrets' generation can be used to generate username/password combinations, API Keys, SIM-based credentials, or tokens.

7.2.1.2. Client Side Only Generation

[TO BE EDITED]

Figure 2: SPP Message Flow for Client-Side only secrets provisioning

The message flow for deploying a client-side only credential (i.e., during registration or renewal) consists of the full three messages exchange. The flow is depicted in Figure 2.

In this case, the Server MUST include, in its first Provisioning message and encoded in the ('Provisioning-Data') TLV, the following data:

- o The ('Credentials-Info') TLV that specifies the target credentials, and
- o The ('Protocol') TLV that specifies the provisioning protocol to be used, and
- o The ('Action') TLV that provides the action to be performed ('Registration') or ('Renew'), and
- o The ('Provisioning-Params') TLV that provides the generation parameters to the Peer, and

Notice that the Server does not include any ('Credentials-Data') TLV in its first message because the Server is not involved in the secret generation (client-side only).

The Peer MUST reply with its own Provisioning message where the Peer MUST encode the following data in the ('Provisioning-Data') TLV:

The ('Credentials-Data') TLV that provides the secret that is being registered

The credentials data MUST conform to the specifications the Server provided in the ('Provisioning-Params') TLV.

The final message is from the Server and it MUST contain (if no errors were detected), the following TLVs encoded, as usual, in the ('Provisioning-Data') TLV:

The ('Credentials-Info') TLV that specifies the metadata associated with the generated secret, and

The ('Credentials-Data') TLV that provides the secret that is provisioned to the Peer

Client-side secrets' generation should be used with caution and an evaluation of the quality of the generated credentials MUST be performed to make sure that the security of the generated secret is adequate for accessing the network. Since evaluating the quality of a secret is quite a difficult tasks, the use of this generation mode MUST be evaluated carefully and selected accordingly to acceptable risk profiles.

7.2.1.3. Client and Server Side Generation

When registering or renewing credentials and the secret generation is split between the Server (1st share) and the Peer (2nd share), the message flow is the same as Section 7.2.1.2 with the following exceptions:

- o The Server MUST send its own share of the secret by including a ('Credentials-Data') TLV in its first message.

All other parameters remain the same.

Co-generation of the secret is the most secure option because both parties can provide the required randomness in their own share of the secret.

7.2.2. SPP Key Pair Provisioning

EAP-CREDS/SSP defines the following flow of messages for requesting the provisioning of key pairs (public and private keys).

7.2.2.1. Server Side Only Generation

[This case covers the server-side generation of KeyPair and Certificate]

7.2.2.2. Client Side Only Generation

[This case covers the registration of a self-signed or already available (e.g., device) certificate]

7.2.2.3. Client and Server Side Generation

This use-case is not supported. In other words, for the provisioning of Key Pairs, the ('Provisioning-Params') can not have both the peer-generation and server-generation bits set.

7.2.3. SPP Certificate Provisioning

EAP-CREDS/SSP defines the following flow of messages for requesting the provisioning of credentials.

7.2.3.1. Server Side Only Generation

[This case covers the server-side generation of KeyPair and Certificate]

7.2.3.2. Client Side Only Generation

[This case covers the registration of a self-signed or already available (e.g., device) certificate]

7.2.3.3. Client and Server Side Generation

[This case covers the generation of the KeyPair on the Peer and the generation of the certificate on the Server]

7.2.4. SPP Token Provisioning

EAP-CREDS/SSP defines the following flow of messages for requesting the provisioning of token-based credentials.

7.2.4.1. Server Side Only Generation

[This case covers the server-side generation of the Token and possibly associated key]

7.2.4.2. Client Side Only Generation

[This case covers the registration of a self-signed or already available (e.g., device) certificate]

7.2.4.3. Client and Server Side Generation

[This case covers the generation of the KeyPair on the Peer and the generation of the Token that contains the reference to the key on the Server]

8. IANA Considerations

This document uses a new EAP type, EAP-CREDS, whose value (TBD) MUST be allocated by IANA from the EAP TYPEs subregistry of the RADIUS registry. This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP-CREDS protocol, in accordance with [RFC8126].

The EAP Method Type number for EAP-CREDS needs to be assigned.

This document also requires IANA to create new registries as defined in the following subsections.

8.1. Provisioning Protocols

Message Type	Purpose
0	Unspecified
1	Simple Provisioning Protocol (SPP)
2	Basic Certificate Management Protocol (CMP-S)
3	Full Certificate Management Protocol (CMP-F)
4	Enrollment over Secure Transport (EST)
5	Certificate Management over CMS (CMC)
6	Automatic Certificate Management Environment (ACME)
...	...
49141 ... 65534	Vendor Specific

Table 5: EAP-CREDS Inner Protocol Identifiers

Assignment of new values for new cryptosuites MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126].

8.2. Token Types

Token Type	Description
0	Unspecified
1	JWT
2	Kerberos
3	OAuth
4	Certificate
200..254	Vendor Specific

Table 6: Token Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.3. Credentials Types

Credentials Type	Description
0	X.509 Certificate
1	Public Key
2	Symmetric Key
3	Username and Password
4	AKA Subscriber Key
5	Bearer Token
6	One-Time Token
7	API Key

Table 7: Credentials Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.4. Credentials Algorithms

ID	Algorithm
0	None
1	RSA
2	ECDSA
3	XMMS
4	AKA Subscriber Key
5	OAuth
6	Kerberos4
7	Kerberos5
200-254	Reserved

Table 8: Credentials Algorithms

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.5. Credentials Datatypes

ID	Data Type
0	None (Binary)
1	PKCS#8
2	PKCS#10
3	PKCS#12
4	PublicKeyInfo
200-254	Reserved

Table 9: Credentials Datatypes

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.6. Challenge Types

ID	Data Type
0	Not Specified
1	EAP-CREDS-ASYMMETRIC
2	EAP-CREDS-SYMMETRIC

Table 10: Challenge Type

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.7. Network Usage Datatypes

ID	Data Type
0	Vendor-Specific
1	Manufacturer Usage Description [RFC8520]
2	Network Access Granting System
3	Firmware Manifest
4..127	Reserved for Future Use

Table 11: Network Usage Datatypes

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.8. Credentials Encoding

ID	Encoding
0	None (Raw)
1	DER
2	PEM
3	Base64
4	JSON
5	XML
6	ASCII
7	UTF-8
200-254	Reserved

Table 12: Credentials Encoding

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.9. Action Types

ID	Data Type	Description
0	Registration	Registers New Credentials
1	Renewal	Renew an Existing Credential
2	Remove	Removes an Existing Credential
200-254	n/a	Reserved

Table 13: Action Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8.10. Usage Metadata Types

Type	Description
0	Binary (Unspecified)
1	MUD File
2	TEEP Manifest

Table 14: Usage Metadata Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

9. Security Considerations

Several security considerations need to be explicitly considered for the system administrators and application developers to understand the weaknesses of the overall architecture.

The most important security consideration when deploying EAP-CREDS is related to the security of the outer channel. In particular, EAP-CREDS assumes that the communication channel has been properly authenticated and that the information exchanged between the Peer and the Server are protected (i.e., confidentiality and integrity).

For example, if certificate-based authentication is used, the server presents a certificate to the peer as part of the trust establishment (or negotiation). The peer SHOULD verify the validity of the EAP server certificate and SHOULD also examine the EAP server name presented in the certificate in order to determine whether the EAP server can be trusted. When performing server certificate validation, implementations MUST provide support for the rules in [RFC5280] for validating certificates against a known trust anchor.

10. Acknowledgments

The authors would like to thank everybody who provided insightful comments and helped in the definition of the deployment considerations.

11. 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>>.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.

- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <<https://www.rfc-editor.org/info/rfc6402>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [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>>.
- [RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <<https://www.rfc-editor.org/info/rfc8520>>.

Author's Address

Massimiliano Pala
CableLabs
858 Coal Creek Cir
Louisville, CO 80027
US

Email: m.pala@openca.org
URI: <http://www.linkedin.com/in/mpala>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 5, 2020

M. Pala
CableLabs
June 3, 2020

Credentials Provisioning and Management via EAP (EAP-CREDS)
draft-pala-eap-creds-07

Abstract

With the increase number of devices, protocols, and applications that rely on strong credentials (e.g., digital certificates, keys, or tokens) for network access, the need for a standardized credentials provisioning and management framework is paramount. The 802.1x architecture allows for entities (e.g., devices, applications, etc.) to authenticate to the network by providing a communication channel where different methods can be used to exchange different types of credentials. However, the need for managing these credentials (i.e., provisioning and renewal) is still a hard problem to solve.

EAP-CREDS, if implemented in Managed Networks (e.g., Cable Modems), could enable our operators to offer a registration and credentials management service integrated in the home WiFi thus enabling visibility about registered devices. During initialization, EAP-CREDS also allows for MUD files or URLs to be transferred between the EAP Peer and the EAP Server, thus giving detailed visibility about devices when they are provisioned with credentials for accessing the networks. The possibility provided by EAP-CREDS can help to secure home or business networks by leveraging the synergies of the security teams from the network operators thanks to the extended knowledge of what and how is registered/authenticated.

This specifications define how to support the provisioning and management of authentication credentials that can be exploited in different environments (e.g., Wired, WiFi, cellular, etc.) to users and/or devices by using EAP together with standard provisioning protocols.

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 December 5, 2020.

Copyright Notice

Copyright (c) 2020 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. Requirements notation	3
2. Introduction	3
2.1. Overview of existing solutions	4
2.2. Scope Statement	4
2.3. EAP-CREDS as tunneled mechanism only	5
2.4. Fragmentation Support	5
2.5. Encapsulating Provisioning Protocols in EAP-CREDS	5
2.6. Algorithm Requirements	6
2.7. Notation	6
3. EAP-CREDS Protocol	6
3.1. Message Flow	6
3.2. Phase Transitioning Rules	7
3.3. Phase One: Initialization	8
3.4. Phase Two: Provisioning	10
3.5. Phase Three: Validation	12
4. EAP-CREDS Message Format	15
4.1. Message Header	15
4.2. Message Payload	17
4.3. EAP-CREDS defined TLVs	18
4.3.1. The Action TLV	18
4.3.2. The Certificate-Data TLV	19
4.3.3. The Challenge-Data TLV	20
4.3.4. The Challenge-Response TLV	21
4.3.5. The Credentials-Information TLV	21

4.3.6.	The Credentials-Data TLV	24
4.3.7.	The Error TLV	25
4.3.8.	The Network-Usage TLV	26
4.3.9.	The Profile TLV	28
4.3.10.	The Protocol TLV	29
4.3.11.	The Provisioning-Data TLV	29
4.3.12.	The Provisioning-Headers TLV	30
4.3.13.	The Provisioning-Params TLV	31
4.3.14.	The Token-Data TLV	33
4.3.15.	The Version TLV	34
5.	EAP-CREDS Messages	35
5.1.	The EAP-CREDS-Init Message	35
5.1.1.	EAP Server's Init Message	35
5.1.2.	EAP Peer's Init Message	36
5.1.2.1.	Bootstrapping Peer's Trustworthiness	36
5.1.3.	The EAP-CREDS-Provisioning Message	37
5.1.4.	The EAP-CREDS-Validate Message	38
6.	Error Handling in EAP-CREDS	39
7.	IANA Considerations	39
7.1.	Provisioning Protocols	39
7.2.	Token Types	39
7.3.	Credentials Types	40
7.4.	Credentials Algorithms	40
7.5.	Credentials Datatypes	41
7.6.	Challenge Types	41
7.7.	Network Usage Datatypes	42
7.8.	Credentials Encoding	42
7.9.	Action Types	42
7.10.	Usage Metadata Types	43
8.	Security Considerations	43
9.	Acknowledgments	44
10.	Normative References	44
	Author's Address	45

1. Requirements notation

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

2. Introduction

Many environments are, today, moving towards requiring strong authentication when it comes to gain access to networks. The 802.1x architecture provides network administrators with the possibility to check credentials presented by a device even before providing any connectivity or IP services to it.

However, the provisioning and management of these credentials is a hard problem to solve and many vendors opt for long-lived credentials that can not be easily revoked, replaced, or simply renewed.

This specification addresses the problem of providing a simple-to-use and simple-to-deploy conduit for credentials management by extending the EAP protocol to support credentials provisioning and management functionality. In particular, the EAP-CREDS method defined here provides a generic framework that can carry the messages for provisioning different types of credentials. EAP-CREDS cannot be used as a stand-alone method, it is required that EAP-CREDS is used as an inner method of EAP-TLS, EAP-TEAP, or any other tunneling method that can provide the required secrecy and (at minimum) server-side authentication to make sure that the communication is protected and with the right server.

2.1. Overview of existing solutions

Currently there are many protocols that address credentials lifecycle management. In particular, when it comes to digital certificates, some of the most deployed management protocols are: Certificate Management Protocol (CMP) [RFC4210], Certificate Management over CMS (CMC) [RFC5272][RFC6402], Enrollment over Secure Transport (EST) [RFC7030], and Automated Certificate Management Environment (ACME) . However, none of these protocols provide native support for client that do not have IP connectivity yet (e.g., because they do not have network-access credentials, yet). EAP-CREDS provides the possibility to use such protocols (i.e., message-based) by defining a series of messages that can be used to encapsulate the provisioning messages for the selected provisioning protocol.

2.2. Scope Statement

This document focuses on the definition of the EAP-CREDS method to convey credentials provisioning and managing messages between the client and the AAA server. Moreover, the document defines how to encode messages for the main IETF provisioning protocols.

This document, however, does not provide specifications for how and where the credentials are generated. In particular, the credentials could be generated directly within the AAA server or at a different location (i.e., the Certificate Service Provider or CSP) site. Different authentication mechanisms (e.g., TLS, etc.) can be used to secure the communication between the server's endpoint and the CSP.

2.3. EAP-CREDS as tunneled mechanism only

EAP-CREDS requires that an outer mechanism is in place between the Peer and the Server in order to provide authentication and confidentiality of the messages exchanged via EAP-CREDS. In other words, EAP-CREDS assumes that an appropriately encrypted and authenticated channel has been established to prevent the possibility to leak information or to allow man-in-the-middle attacks.

This choice was taken to simplify the message flow between Peer and Server, and to abstract EAP-CREDS from the secure-channel establishment mechanism. EAP-TLS, or EAP-TEAP are examples of such mechanisms.s

2.4. Fragmentation Support

EAP does not directly support handling fragmented packets and it requires the outer method to provide fragmentation support.

Because of the outer method requirements in particular, removing any support for fragmented messages in EAP-CREDS removes the duplication of packets (e.g., Acknowledgment Packets) sent across the Peer and the Server, thus resulting in a smaller number of exchanged messages

2.5. Encapsulating Provisioning Protocols in EAP-CREDS

In order to use EAP-CREDS together with your favorite provisioning protocol, the messages from the provisioning protocol need to be sent to the other party. In EAP-CREDS, this is done by encoding the provisioning protocol messages inside the ('Provisioning-Data') TLV. In case the provisioning protocol uses additional data for its operations (e.g., uses HTTP Headers), this data can be encoded in a separate ('Provisioning-Headers') TLV.

Since the implementation of the provisioning endpoint could happen in a (logically or physically) different component, a method is needed to identify when a provisioning protocol has actually ended. In EAP-CREDS, the 'D' bit in the message headers is used for this purpose.

In the first message of Phase Two, the Server provides the client with all the selected parameters for one specific credential that needs attention (or for a new credential) to be managed by the network. In particular, the server provides, at minimum, the ('Protocol') TLV, the ('Action') TLV, and the ('Provisioning-Params') or the ('Credentials-Info') TLV.

After checking the parameters sent by the Server, if the Peer does not support any of the proposed ones, it MUST send a message with one

single ('Error') TLV with the appropriate error code(s). The server, can then decide if to manage a different set of credentials (if more where reported by the Peer in its Phase One message) or if to terminate the EAP session with an error.

The Peer and the Server exchange Provisioning messages until an error is detected (and the appropriate error message is sent to the other party) or until Phase Two is successfully completed.

2.6. Algorithm Requirements

EAP-CREDS uses the SHA-256 hashing algorithm to verify credentials in phase three of the protocol. Peers and Servers MUST support SHA-256 for this purpose.

2.7. Notation

In this document we use the following notation in the diagrams to provide information about the cardinality of the data structures (TLVs) within EAP-CREDS messages:

Symbol	Example	Usage
{ }	{TLV1}	Curly Brackets are used to indicate a set
[]	{[TLV2]}	Square Brackets are used to indicate that a field is optional
()	{TLV1(=V)}	Round Squares are used to specify a value
+	{TLV_2+}	The Plus character indicates that one or more instances are allowed

Table 1: EAP-CREDS Notation

3. EAP-CREDS Protocol

In a nutshell, EAP-CREDS provides the abstraction layer on top of which credentials provisioning/managing protocols can be deployed thus enabling their use even before provisioning IP services.

This section outlines the operation of the protocol and message flows. The format of the CREDS messages is given in Section 4.

3.1. Message Flow

EAP-CREDS message flow is logically subdivided into three different phases: Initialization, Provisioning, and Validation. EAP-CREDS

enforces the order of phases, i.e. it is not possible to move to an earlier phase.

Phase transitioning is controlled by the Server. In particular, the server, after the last message of a phase, it can decide to either (a) start the next phase by sending the first message of the next phase, or (b) continue the same phase by sending another "first" message of the phase (e.g., managing a second set of credentials) - this is allowed only in Phase Two and Phase Three but NOT in Phase One, or (c) terminate the EAP session.

Phase One (Required). Initialization. During this phase the Peer and the Server exchange the information needed to select the appropriate credentials management protocol. Phase One flow is composed by only messages. In particular, the Server sends its initial message of type ('EAP-CREDS-Init'). The Peer replies with the details about which provisioning protocols are supported, and additional information such as the list of installed credentials and, optionally, authorization data (for new credentials registration).

Phase Two (Optional). Provisioning Protocol Flow. In this phase, the Peer and the Server exchange the provisioning protocol's messages encapsulated in a EAP-CREDS message of type Provisioning. The messages use two main TLVs. The first one is the ('Provisioning-Headers') TLV which is optional and carries information that might be normally conveyed via the transport protocol (e.g., HTTP headers). The second one is the ('Provisioning-Data'), which is required and carries the provisioning protocol's messages. The server can decide to repeat phase two again to register new credentials or to renew a separate set of credentials by issuing a new ('Provisioning') message for the new target. When no more credentials have to be managed, the Server can start phase three or simply terminate the EAP session.

Phase Three (Optional). Credentials Validation. This optional phase can be initiated by the server and it is used to validate that the Peer has properly installed the credentials and can use them to authenticate itself. Depending on the credentials' type, the messages can carry a challenge/nonce, the value of the secret/token, or other information. The format of the credentials is supposed to be known by the provider and the device.

3.2. Phase Transitioning Rules

In order to keep track of starting and ending a phase, EAP-CREDS defines several bits and fields in the EAP-CREDS message headers. In particular, as described in Section 4.1, the 'S' bit is used to

indicate the beginning (or Start) of a phase, while the 'Phase' field (4 bits) is used to indicate the phase for this message.

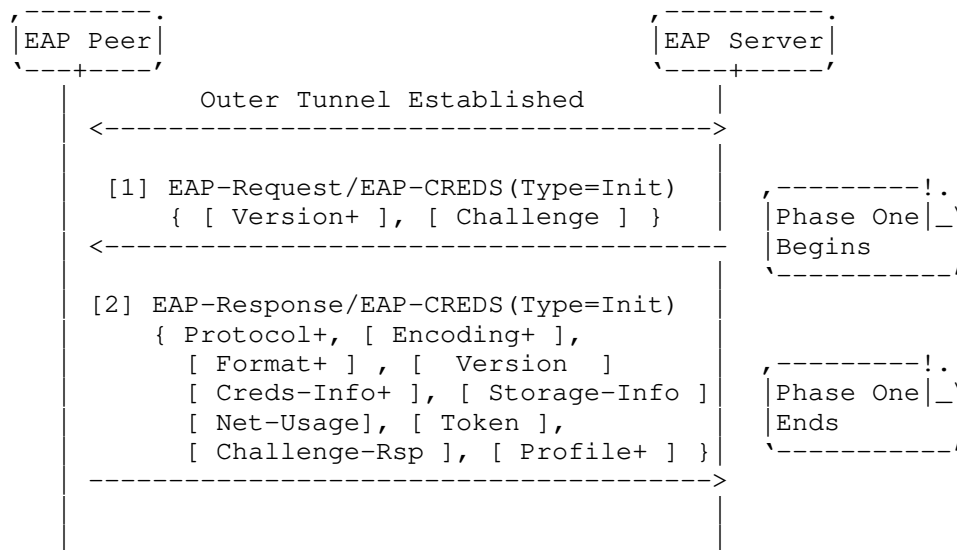
In EAP-CREDS, phase transitioning is under the sole control of the Server, therefore the value of the 'S' bit is meaningful only in messages sent by the Server. The value of the 'S' bit in Peer's messages SHALL be set to '0x0' and SHALL be ignored by the server.

When starting a new phase, the Server MUST set the 'S' bit to '1' and the 'Phase' field to the current phase number (e.g., one, two, or three).

In case the first message of a phase is to be repeated (e.g., because of processing multiple credentials), the 'S' bit SHALL be set to '0' (i.e., it should be set to '1' only on the first occurrence and set to '0' in subsequent messages).

3.3. Phase One: Initialization

The following figure provides the message flow for Phase One:



EAP-CREDS Phase One Message Flow

[1] Server sends EAP-Request/EAP-CREDS (Type=Init):

After the establishment of the outer mechanism (e.g., EAP-TLS, EAP-TEAP, EAP-TTLS, etc.), the server MAY decide to start a credentials management session. In order to do that, the Server sends an EAP-Request/EAP-CREDS(Type=Init) message to the Peer with one ('Phase-Control') TLV with the 'S' bit set to '1' and the value set to '1' (thus indicating the beginning of Phase One). Also, the Server MAY use one or more ('Version') TLVs to indicate the supported versions.

The Server MAY also specify which versions of EAP-CREDS are supported by adding one or more ('Version') TLVs. If no ('Version') TLV is added to the message, the Peer SHOULD assume the supported version is 1 ('0x1').

[2] The Peer sends EAP-Response/EAP-CREDS(Type=Init)

The Peer, sends back a message that carries one ('Version') TLV to indicate the selected version of EAP-CREDS (i.e. from the list provided by the server) (optional). If the client does not include the ('Version') TLV, the Server MUST use the most recent supported version of EAP-CREDS. Moreover, the Server includes one or more ('Protocol') TLVs to indicate the list of supported provisioning protocols, followed by one ('Credentials-Info') TLVs for each installed credentials to provide their status to the server (i.e., if multiple credentials are configured on the Peer for this Network, then the Peer MUST include one ('Credentials-Info') TLV for each of them).

The Peer also provides the list of supported Encodings and Formats by adding one or more ('Supported-Encodings') and ('Supported-Formats') TLVs. The Peer MAY also provide the Server with information about the Peer's credentials storage by using the 'Storage-Status' TLV.

When there are no available credentials, the Peer MAY include an authorization token that can be consumed by the Server for registering new credentials. In particular, the Peer can include the ('Token-Data') TLV to convey the value of the token. The ('Challenge-Data') and ('Challenge-Response') TLVs, instead, can be used to convey a challenge and its response based on the authorization information (e.g., maybe a public key hash is present in the Token, then the peer can generate some random data - or use the one from the Server - and generate a signature on that value: the signature SHALL be encoded in the ('Challenge-Response') TLV and it should be calculated over the concatenation of values inside the ('Challenge-Data') TLV and the ('Token-Data') TLV.

Also, the Peer MAY add one or more ('Profile') TLVs to indicate to the Server which profiles are requested/supported (e.g., a pre-configuration MAY exist on the Peer with these ecosystem-specific identifiers).

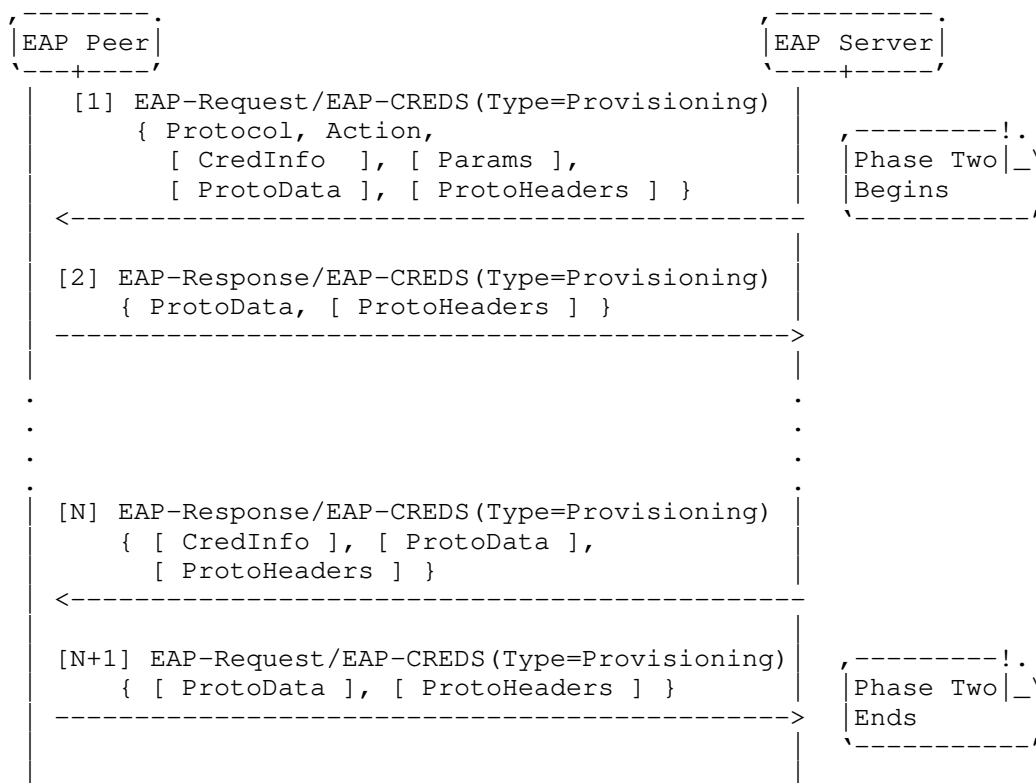
Ultimately, the Peer MAY include additional metadata regarding the status of the Peer. To this end, the Peer can use a ('Storage-Info') TLV to provide the server with additional data about the Peer's capabilities and resources. Also, the ('Network-Usage') TLV can be used to provide the Server with the indication of which network resources are needed by the Peer and what is its intended utilization pattern(s).

The server checks that the Peer's selected protocol, version, and parameters are supported and, if not (or if the server detects an error), it can (a) send a non-recoverable error message to the peer, notify the outer (tunneling) layer, and terminate the EAP-CREDS session, or (b) start phase one again by sending a new ('EAP-CREDS-Init') message that will also carry an ERROR TLV that provides the Peer with the reason the initial response was not acceptable. In this case, the ('Phase-Control') TLV MUST be omitted since it is not the first message of phase one. The server and the peer can repeat phase one until they reach an agreement or the session is terminated by the Server.

NOTE WELL: The determination of the need to start phase two or not is based on the contents of the ('Credentials-Info') TLV sent by the Peer (e.g., a credential is about to expire or a credential is simply missing).

3.4. Phase Two: Provisioning

The following figure provides the message flow for Phase 2:



EAP-CREDS Phase Two Message Flow

- [1] The Server sends EAP-Request/EAP-CREDS (Type=Init)

The first message of Phase Two indicates that the Server is ready to initiate the selected provisioning protocol.

- [2] The Peer sends EAP-Response/EAP-CREDS (Type=Init)

After that, the Peer sends its first message to the Server by sending the EAP-Response/EAP-CREDS (Type=Provisioning) message. This message contains the selected provisioning protocol's message data and some extra fields (e.g., transport-protocol headers) in the ('Provisioning-Data') and ('Protocol-Headers') TLVs respectively.

- [3] The Server sends EAP-Request/EAP-CREDS (Type=Init)

The Server replies to the Peer's message with EAP-Request/EAP-CREDS (Type=Provisioning) messages until the provisioning protocol reaches an end or an error condition arise (non-recoverable).

[N] The Server sends EAP-Request/EAP-CREDS (Type=Provisioning)

When the provisioning protocol has been executed for the specific set of credentials, the server sends a last message that MUST include the description of the provisioned credentials in a ('Credentials-Info') TLV and MUST set the 'D' bit in the EAP-CREDS message header to '1' to indicate that the server does not have any more ('Provisioning') messages for this credential. The final message does not need to be an empty one, i.e. other TLVs are still allowed in the same message (e.g., the 'Provisioning-Data' and the 'Provisioning-Headers' ones).

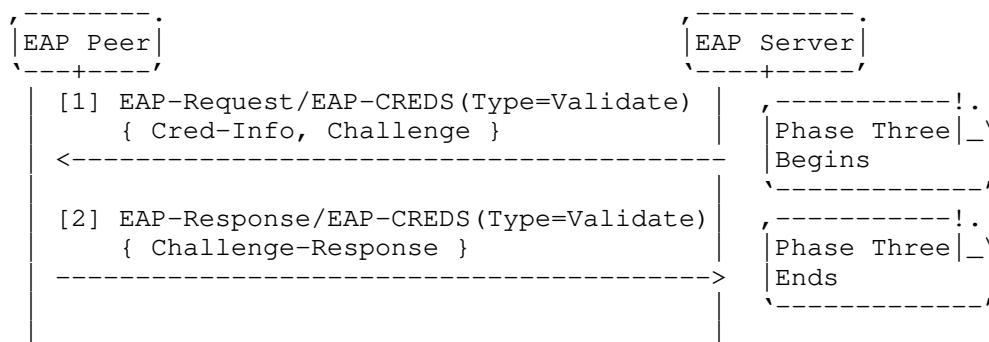
[N+1] The Peer sends EAP-Request/EAP-CREDS (Type=Provisioning)

The Peer MUST reply to the server with a ('Provisioning') message that MUST have the 'D' bit in the EAP-CREDS message header set to '1', thus indicating that the credentials have been installed correctly. In case of errors, the Peer MUST include the appropriate ('Error') TLV. Also in this case, the final message does not need to be an empty one, i.e. other TLVs are still allowed in the same message (e.g., the 'Provisioning-Data' and the 'Provisioning-Headers' ones).

At this point, the Server can decide to provision (or manage) another set of credentials by issuing a new ('Provisioning') message, or it can decide to start Phase Three by sending its first ('Validate') message, or it can terminate the EAP session.

3.5. Phase Three: Validation

The following figure provides the message flow for Phase 3:



EAP-CREDS Phase Three Message Flow

Phase three is optional and it is used by the server to request the client to validate (proof) that the new credentials have been installed correctly before issuing the final Success message.

NOTE WELL: Phase Three introduces a dependency on the selected hashing algorithm to provide common and easy way to check the integrity and functionality of a newly installed set of credentials.

[1] The Server sends EAP-Request/EAP-CREDS (Type=Validate)

In order to start Phase Three, the Server sends an EAP-Request/EAP-CREDS (Type=Validate) message to the Peer. The Server MUST include the ('Credentials-Info') TLV to provide the indication about which set of credentials the Server intends to validate. The Server MUST also include a randomly generated challenge in the message to the client. The type of challenge determines how the ('Challenge-Response') is calculated. EAP-CREDS defines the asymmetric and symmetric challenges in Section 7.6 and others can be defined according to the specified rules.

As usual, the Server MUST set, in the headers, the 'S' bit to '1' in its first message of Phase Three and the 'Phase' value shall be set to '3' (beginning of Phase Three).

[2] The Peer sends EAP-Response/EAP-CREDS (Type=Validate)

When the client receives the Validate message from the server, it calculates the response to the challenge and sends the response back to the server in a EAP-Response/EAP-CREDS (Type=Validate) message. When the EAP-CREDS-ASYMMETRIC-CHALLENGE and EAP-CREDS-

SYMMETRIC-CHALLENGE values are used in the Challenge type, the Peer MUST calculate the response as follows:

Public-Key

For any public-key based credentials (e.g., certificates or raw key pairs), the response to the challenge is calculated by generating a signature over the hashed value of the challenge. The hashing algorithm to be used for this purpose is specified in Section 2.6. The format of the signature in the ('Challenge-Response') TLV is the concatenation of:

- The signatureAlgorithm (DER encoded) which contains the identifier for the cryptographic algorithm used by the Peer to generate the signature. [RFC3279], [RFC4055], and [RFC4491] list supported signature algorithms, but other signature algorithms MAY also be supported. The definition of the signatureAlgorithm is provided in Section 4.1.1.2 of [RFC5280].
- The signatureValue (DER encoded) which contains the digital signature itself. The signature value is encoded as a BIT STRING and the details of how to generate the signatures' structures can be found in Section 4.1.1.3 of [RFC5280] and referenced material.

Symmetric Secret

For any symmetric based credentials (e.g., password or Key), the response to the challenge is calculated by using the selected hash function (see Section 2.6) on the concatenation of (a) the value carried in the server-provided ('Challenge-Data') TLV, and (b) the secret value itself (salted hash).

The initial values for the type of challenges are described in the Section 7.6. Other types of challenges MAY be defined according to the specified procedures.

In case of issues with the validation of newly deployed credentials, both the Server and the Peer should consider those credentials invalid (or unusable) and should issue the required failure message(s).

4. EAP-CREDS Message Format

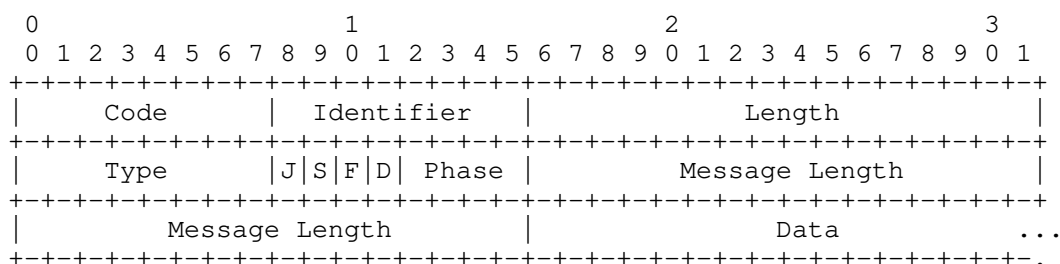
The EAP-CREDS defines the following message types:

1. EAP-CREDS/Init
2. EAP-CREDS/Provisioning
3. EAP-CREDS/Validate

Each of these message types have the basic structure as identified in Section 4.1. EAP-CREDS messages contain zero, one, or more TLVs. The internal structure of the different types of TLVs is described in Section 4.2, while a detailed description of the EAP-CREDS message types is provided in Section 5.

4.1. Message Header

The EAP-CREDS messages consist of the standard EAP header (see Section 4 of [RFC3748]), followed by the version of the EAP-CREDS (4 bits) and a field (4 bits) reserved for future use. The header has the following structure:



Where the Code, Identifier, Length, and Type fields are all part of the EAP header as defined in [RFC3748]. Since EAP-CREDS can only be used as a tunneled mechanism, the presence of these fields is only for backward compatibility with existing parsers. In particular, the 'Length' field is not used (can be ignored): the message length is carried in the 'Message Length' field instead.

The Type field in the EAP header is <TBD> for EAP-CREDS.

The Flags bitfield is used to convey status information (e.g., extra long message, phase number, phase transitioning state). The transition-control bit (i.e., the 'S' bit) are set in Server's messages and are ignored in Peer's messages (the Server is the entity

that unilaterally controls the phase transition process). The meanings of the bits in the 'Flags' field are as follows:

Bit 'J' (Jumbo Message) - If set, it indicates the presence of the 'Message Length' field. This bit SHALL be used only when the size of the message exceeds the maximum value allowed in the 'Length' field. In this case, the 'Message Length' field is added to the message and set to the whole message size and the 'Length' field is used for the current fragment length. If not set, the 'Message Length' field is not present in the Message and the 'Length' field is used for the message size (and the 'F' bit MUST be set to '0').

Bit 'S' (Start) - If set, this message is the first one of a new EAP-CREDS phase. The value of the new phase is encoded in the 'Phase' field.

Bit 'F' - If set, this message is a fragment of a message. In this case, the 'Data' field is to be concatenated with all messages with the 'F' bit set to '1' until the message with the 'F' bit set to '0' that indicates the end of the message. If the message is not fragmented, the 'F' bit MUST be set to '0'. The use of this bit is required when the tunneling method does not provide support for messages up to 2^{32} bits in size.

Bit 'D' - This bit is used in Phase Two and Phase Three to indicate that the specific operation for the identified credential is over. For example, when multiple credentials exist on the Peer and the Server needs to manage and validate one of them. In its last message, when the provisioning protocol is done, the server sets the 'D' (Done) bit to indicate that it is done. The Peer, in its reply, sets the bit to indicate the end of provisioning for this credentials is also over. After that, the Server can continue Phase Two, transition to Phase Three, or terminate the EAP session.

The Phase field is a 4-bits value and identifies the EAP-CREDS phase for the current message. The version of EAP-CREDS described in this document supports three values for this field:

0x01 - Phase One

0x02 - Phase Two

0x03 - Phase Three

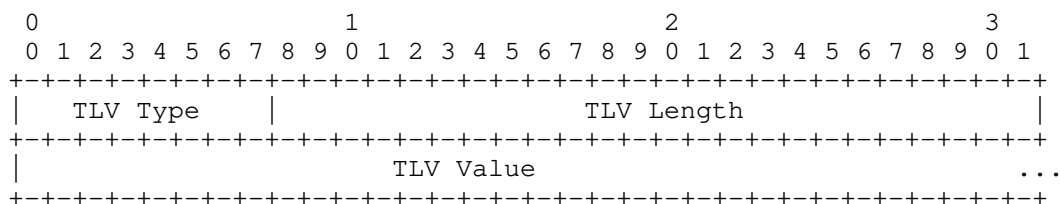
A detailed explanation of the 'Phase' and 'Flags' fields of the message headers is provided in Section 3.2.

The Data field is the message payload. The full description of this field is provided in the next section.

4.2. Message Payload

The Data part of the message is organized as zero, one, or more TLV objects whose structure is defined in this section.

Each TLV object has the same basic structure that is defined as follows:



Where:

TLV-Type (uint8)

This field is used to indicate the type of data that the TLV carries. The type of TLV determines its internal structure. The supported values for this fields are provided in the following table:

Length (uint24)

This field carries the size of the value of the TLV. In particular, the overall size of a TLV (i.e., the header plus the value) can be calculated by adding the size of the header (6 octets) to the value of the Length field (i.e., the size of the TLV's value).

TLV Name	TLV Type	Scope/Usage
<TBD>	Action TLV	Phase Two
<TBD>	Certificate-Data TLV	Phase Two
<TBD>	Challenge-Data TLV	Phase Two, Phase Three
<TBD>	Challenge-Response TLV	Phase Two, Phase Three
<TBD>	Credentials-Data TLV	Phase Two
<TBD>	Credentials-Info TLV	Phase Two, Phase Three
<TBD>	Error TLV	All Phases
<TBD>	Network-Usage TLV	Phase One
<TBD>	Profile TLV	Phase Two
<TBD>	Protocol TLV	Phase One, Phase Two
<TBD>	Provisioning-Data TLV	Phase Two
<TBD>	Provisioning-Headers TLV	Phase Two
<TBD>	Provisioning-Params TLV	Phase Two
<TBD>	Token-Data TLV	Phase One
<TBD>	Version TLV	Phase One

Table 2: EAP-CREDS Supported TLVs Types

TLV Value (> 1 octet)

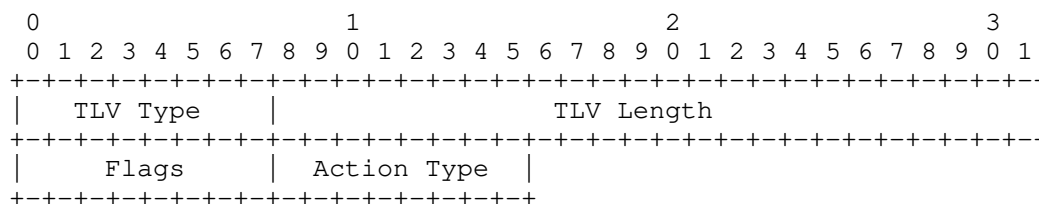
This field carries data for the identified TLV. The internal structure is determined by the TLV Type field.

The rest of this section describes the structure of the different supported TLVs and their usage in the different messages.

4.3. EAP-CREDS defined TLVs

EAP-CREDS messages's payload comprises zero, one, or more TLVs that are encoded in a single EAP-CREDS message. The values for the TLV Type that are supported by this specifications are listed in Table 2.

4.3.1. The Action TLV



TLV Type (uint8)

<TBD> - Action TLV

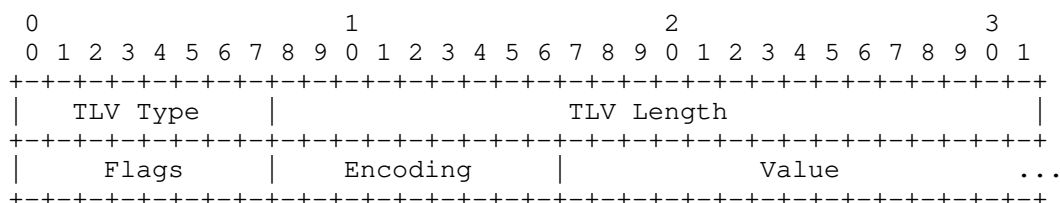
TLV Length (uint24)

Fixed value (=2)

Flags (uint8)

Reserved

4.3.2. The Certificate-Data TLV



TLV Type (uint8)

<TBD> - Certificate-Data TLV

Length (uint24)

Provides the length of the TLV (> 3 octets)

Flags (uint8)

Provides a BITMASK that can be used to provide additional information related to the encapsulated certificate. The bits have the following meaning:

Bit 0 - If set, the certificate is trusted (Trust Anchor)

Bit 1 - If set, the certificate is a CA certificate

Bit 2 - If set, the certificate is self-signed

Bit 3 - If set, the certificate is a proxy certificate

Bit 4 - If set, the certificate is an attribute certificate

Bit 5 - Reserved

Bit 6 - Reserved

Bit 7 - Reserved

For a Trusted Root CA, the value of the flags shall be 0x7 (0000 0111). For an intermediate CA certificate that is not implicitly trusted, the value of the flags field should be set to 0x02 (0000 0010). For an End-Entity certificate, the value of the Flags will be 0x0 (0000 0000).

Format (uint8)

Provides the indication of the Format the certificate is in. The allowed values for this field are listed in Section 7.5.

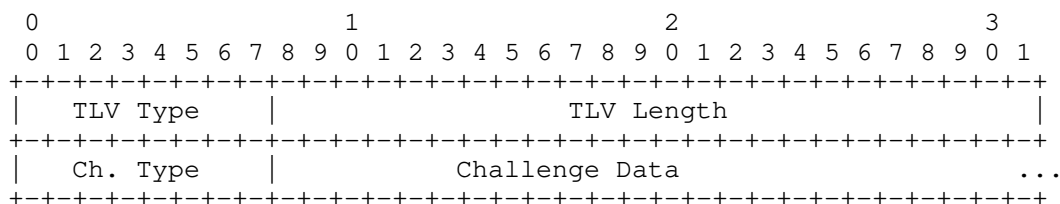
Encoding (uint8)

Provides the indication of the Encoding the certificate is in. The allowed values for this field are listed in Section 7.8.

Value (octet string)

This field carries the data for the certificate.

4.3.3. The Challenge-Data TLV



TLV Type (uint8)

<TBD> - Challenge-Data TLV

Length (uint24)

3 octets

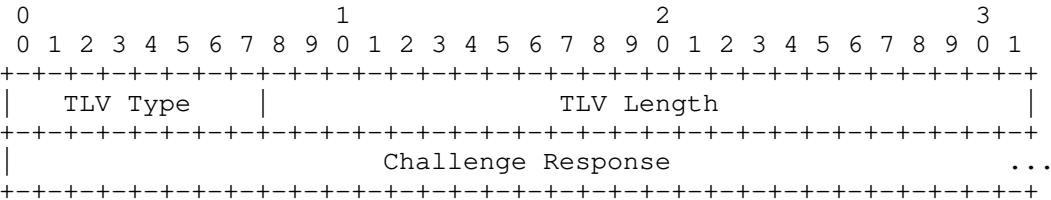
Challenge Type (uint8)

This field carries the type of Challenge. In particular, the challenge type determines how the Peer MUST calculate the ('Challenge-Response'). The initial values for this field are listed in Section 7.6. Please refer to Section 3.5 for a detailed explanation of how to calculate the response to the challenge for the challenge types defined in this document.

Challenge Data (> 1 octet)

This field carries the data to be used as a challenge when validating newly deployed credentials.

4.3.4. The Challenge-Response TLV



TLV Type (uint8)

<TBD> - Challenge-Response TLV

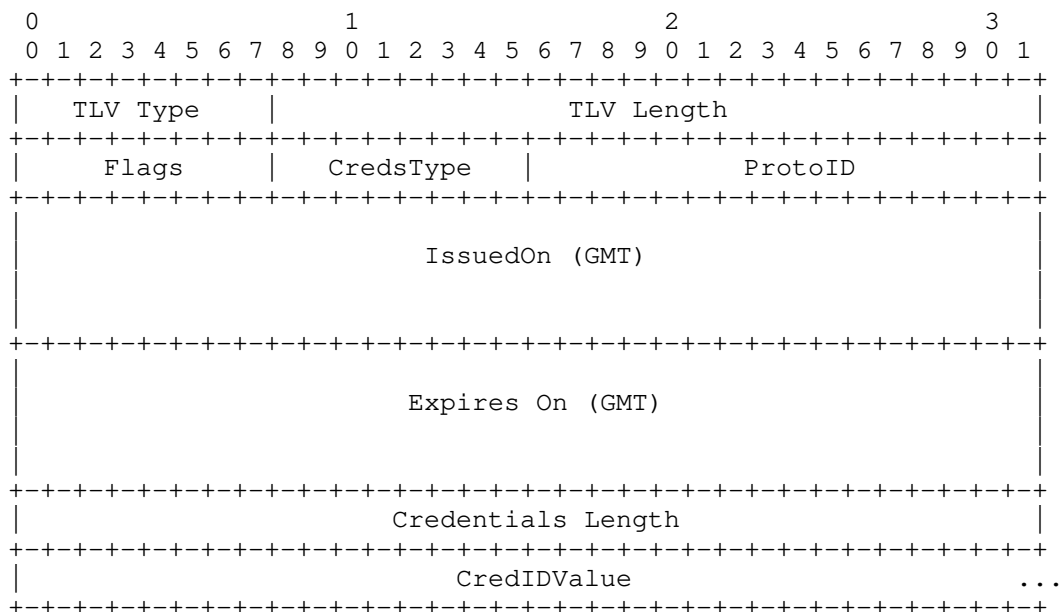
Length (uint24)

3 octets

Challenge Response (> 1 octet)

This field carries the data that resulted from the use of the credentials to be validated.

4.3.5. The Credentials-Information TLV



The Credential-Information TLV is used by the Peer to provide a description of the installed credentials that are relevant for the network that is being accessed.

For example, when a set of credentials need to be renewed, the server checks the ('Credentials-Info') from the Peer and eventually selects the right one for renewal. The TLV structure is as follows:

TLV Type (uint8)

<TBD> - Credentials-Information TLV

Length (uint24)

Provides the total length of the body of the Credential-Information TLV.

Flags (uint8)

Provides a BITMASK that can be used to provide information about the status of the credentials (e.g., if the use marks the credentials to be compromised). The bits have the following meaning:

Bit 0 - If set, the credential is marked as compromised

Bit 1 - If set, the credential is immutable and cannot be updated

Bit 2 - Private Key or Secret Immutable, the public part of the credential (e.g., a certificate) can still be updated

Bit 3 - If set, the credential cannot be updated (both public and private parts)

Bit 4 - If set, the credential is ready to be used

Bit 5 - If set, the credential was generated on the server

Bit 6 - If set, the Peer would like to update the credential even if they are not expired

Bit 7 - Reserved

CredType (uint8)

This field provides the description of the type of credential. The type of credentials are listed in Section 7.3

ProtoID (uint16)

This field indicates the protocol that was used to retrieve the target credential. When the TLV is used in a Request by the Server, this field is ignored. The values for this field are listed in Section 7.1.

IssuedOn (16 octets)

This field carries the GMT date for when this credential was issued. This field is 16 bytes long (the last byte must be set to '0x00') and contains the NULL-terminated ASCII string that represents the timestamp where the credential was issued. When the value is not set, the field should be set to { 0x00 }. The format of the string is as follows:

YYYYMMDDHHmmssZ

Where:

YYYY - is the 4 digits representation of the year

MM - is the 2 digits representation of the month

DD - is the 2 digits representation of the day of the month

HH - is the 2 digits representation of the hour of the day (24 hour format)

mm - is the 2 digits representation of the minutes of the hour

ss - is the 2 digits representation of the seconds of the minute

Z - is the character 'Z'

ExpiresOn (16 octets)

This field carries the GMT date for when this credential is to be considered expired. This field is 16 bytes long (the last byte must be set to '0x00') and contains the NULL-terminated ASCII string that represents the timestamp where the credential was issued. The format is the same as the ('IssuedOn') field. When the value is not set, the field should be set to { 0x00 }.

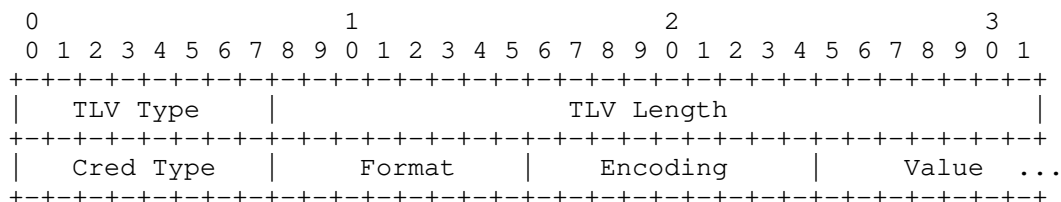
Credentials Length (uint16)

Length (in bytes) of the Credentials value. When used with a public-key type of credentials, this is the size of the key (e.g., for an RSA 2048 bit keys, this field should carry the value of 256). When used with a symmetric secret, this field carries the size of the secret (in bytes).

CredIDValue (> 1 octet)

The binary value of the credentials' identifier. This identifier can be the binary value of the SHA-256 calculated over the certificate, a username, or it could be a random handle. As long as the ID allows the peer and the server to uniquely (in its context) identify the credentials, the value of this field can be calculated in any way.

4.3.6. The Credentials-Data TLV



TLV Type (uint8)

<TBD> - Credentials-Data TLV

Length (uint24)

Provides the length of the TLV (> 3 octets)

Cred Type (uint8)

Provides the indication of the type of credentials. The allowed values for this field are listed in Section 7.3.

Format (uint8)

Provides the indication of the Format the credentials are in. The allowed values for this field are listed in Section 7.5.

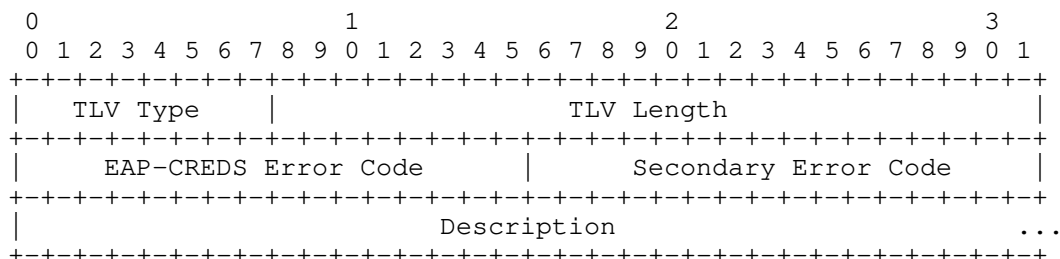
Encoding (uint8)

Provides the indication of the Encoding the credentials are in. The allowed values for this field are listed in Section 7.8.

Value (octet string)

This field carries the data for the credentials.

4.3.7. The Error TLV



TLV Type (uint8)

<TBD> - Challenge-Response-Data TLV

Length (uint24)

3 octets

EAP-CREDS Error Code (2 octets)

This field carries the EAP-CREDS error code. These code are related to the EAP-CREDS operations only and it should not be used to carry the Provisioning-Protocol specific error codes.

The error codes supported by this specifications are listed in Section 4.3.7.

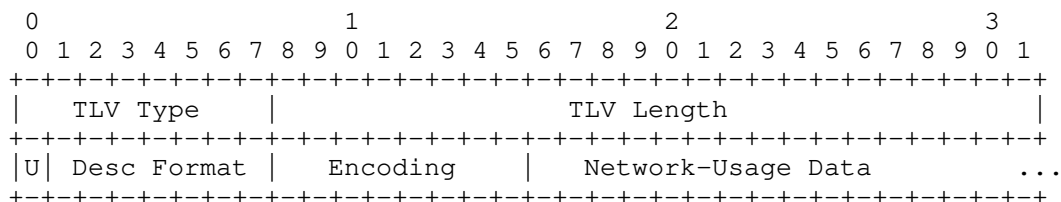
Secondary Error Code (2 octets)

This field is used to convey an error at the encapsulation layer (i.e., the provisioning protocol error). For example, this field can be used to convey a transport protocol error code (e.g., HTTP status code). Do not use this field to convey EAP-CREDS specific errors.

Description (> 1 octet)

The Description field is optional (i.e., when the Description Size is set to zero) and carries information about the error that occurred. The message may or may not be used by a user or an automated process for debugging purposes.

4.3.8. The Network-Usage TLV



TLV Type (uint8)

<TBD> - Network-Usage TLV

Length (uint24)

Variable Length TLV (Value must be > 2)

Description Format (uint8)

The Type of data encoded in the Peer Description Data. The initial values for this field are listed in Section 7.10.

Encoding (uint8)

Provides the indication of the Encoding the network usage description data is in. The allowed values for this field are listed in Section 7.8.

The 'U' field (1 bit)

The 'URL' bit ('U') is used to indicate if the value of the Network-Usage Data field is to be interpreted as a URL or as the actual data. In particular, if the value in the 'URL' bit is '1', then the value in the Network-Usage Data field is to be interpreted as the URL where the actual data can be downloaded from. Otherwise, if the 'URL' bit is set to '0', then the value in the Network-Usage Data field is to be interpreted as the actual data (not a URL referencing it).

An example use of this bit is when the Peer wants to convey the URL of the MUD file [RFC8520]. In this case, the Peer can set the Network-Usage Data field to the Url of the MUD file related to the Peer.

Desc Format (7 bits)

This field provide the expected data format for the Network-Usage Data. For example, the value in this field could be set to 'MUD'

and have the 'U' bit set to '1' to provide the MUD-related information at credentials management time instead of at network-provisioning time (DHCP option). This possibility could help the Network controller to decide if the device shall be allowed to register its credentials or not.

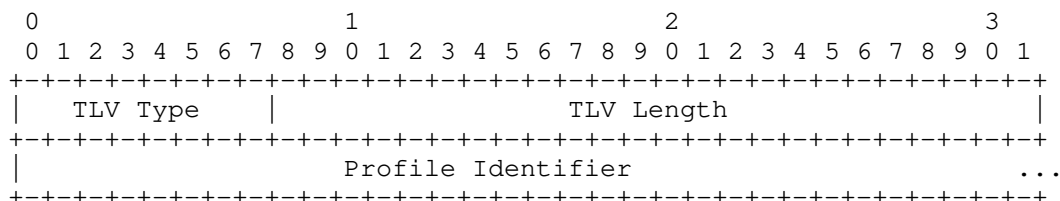
The list of initial values for this field is provided in Section 7.7.

Network-Usage Data (octet string)

This is additional information related to the device. In particular, this TLV can be used by the Peer to provide the Server with the description of the intended network usage or a URL that points to the same information.

For example, this field can be used to convey a MUD file (Manufacturer Usage Description) or the latest firmware-update manifest.

4.3.9. The Profile TLV



TLV Type (uint8)

<TBD> - Profile Identifying Data TLV

Length (uint24)

Length value should be ≥ 1

Profile Identifier (octet string)

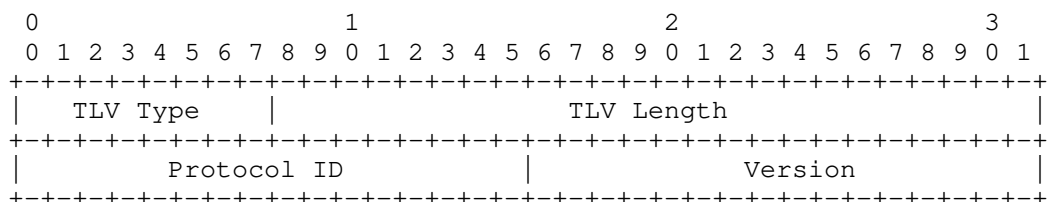
The Profile Identifier is used to provide indication to the other party about which profiles are supported when requesting credentials management.

Also in this case, the data used in this field is left to be interpreted by the end-point and it is independent from the EAP-

CREDS data types. This could be a raw byte value, a string, or a more complex structured data (e.g., an OID).

An example of values for this field, an end-point could use the string representation (i.e., dotted representation) of the Object Identifier (OID) of the specific profile supported (e.g., could be defined in the Certificate Policy of the credentials' provider).

4.3.10. The Protocol TLV



TLV Type (uint8)

<TBD> - Protocol TLV

TLV Length (uint24)

Fixed TLV Length value of 4.

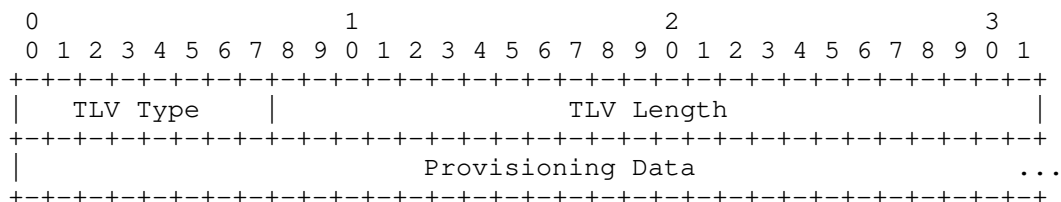
Protocol ID (uint16)

The Protocol ID value carries the id of a supported provisioning protocol. The initial list of values for the provisioning protocol identifiers can be found in Section 7.1.

Version (uint16)

The Version (Protocol Version) value represents the specific version of the identified provisioning protocol. When no version is specified for a protocol (i.e., either it does not support multiple versions or it does not matter), the value of this field should be set to '0x0'.

4.3.11. The Provisioning-Data TLV



TLV Type (uint8)

<TBD> - Provisioning-Data TLV

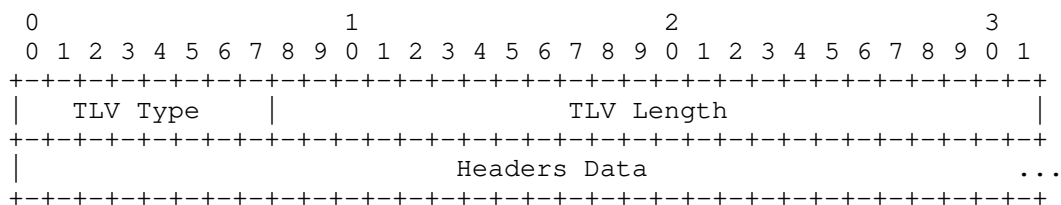
Length (uint24)

3 octets

Headers Data (> 1 octet)

This field carries the provisioning protocol's messages.

4.3.12. The Provisioning-Headers TLV



TLV Type (uint8)

<TBD> - Provisioning-Headers TLV

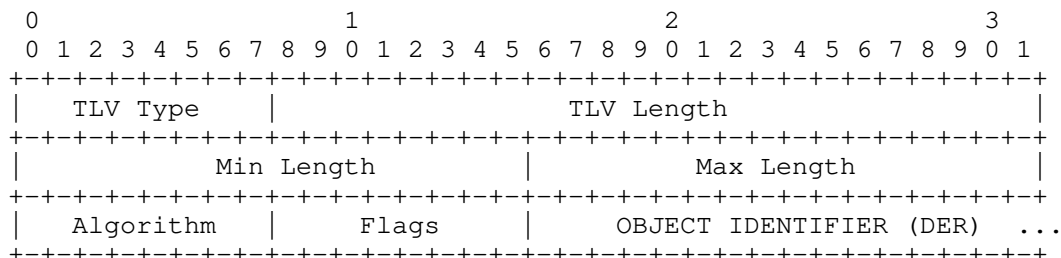
Length (uint24)

3 octets

Headers Data (> 1 octet)

This field carries the meta-data (if any) that might be associated with the transport-layer normally used with the provisioning protocol. For example, this TLV can carry the set of HTTP headers required by EST or ACME.

4.3.13. The Provisioning-Params TLV



TLV Type (uint8)

<TBD> - Provisioning-Params TLV

Length (uint24)

Provides the length of the TLV (>= 6 octets)

Min Length (uint16)

Provides the minimum allowed size size for the credentials. This value has meaning depending on the context of the credentials, however sizes are always expressed in bytes.

For example, when used with a symmetric key or a password, the ('Min Length') and ('Max Length') refer to the minimum and maximum size of the password data. The ('Algor OID') field can be omitted in this case.

On the other hand, when referring public-key credentials, this field should carry the size of the modulus of the key. For example, for an RSA 2048 bit keys, the field should carry the value of 256. For an ECDSA that uses the prime256r1 curve, this field should carry the value of 32 and the Algor OID should be the DER representation of the specific value of the curve (i.e., the DER representation of '1.2.840.10045.3.1.7').

Max Length (uint16)

Provides the indication maximum size of the credentials. This value has meaning depending on the context of the credentials, however sizes are always expressed in bytes.

The same considerations apply to this field as well as the ('Min Length') one discussed above.

Flags (uint8)

Provides a BITMASK that can be used to provide information about the status of the credentials (e.g., if the use marks the credentials to be compromised). The bits have the following meaning:

Bit 0 - Credentials (or part of it) are to be generated on the server

Bit 1 - Credentials (or part of it) are to be generated on the peer

Bit 2 - Credentials are to be generated on dedicated hardware

Bit 3 - Reserved

Bit 4 - Reserved

Bit 5 - Reserved

Bit 6 - Reserved

Bit 7 - Reserved

When using public-key based credentials, the bits 0 and 1 are mutually exclusive.

When using passwords or shared secrets, if bit 0 is set, then the secret is generated by the server and then sent to the client. On the other hand, if bit 1 is set, then the secret is generated by the peer and then sent to the server. Ultimately, if both bits are set, then the Server generates the first part of the password and sends it to the Peer, while the Peer generates the second part of the password and sends it to the Server. The password to be used for future authentication is the concatenation of the two shares of the password: first the one from the Server, then the one from the Client.

NOTE WELL: Last but not least, since these passwords/secrets are meant to be used in a automated fashion, there is no restriction around the character set to use or their interpretation. Therefore, it is good practice to generate

random pass-phrases that use the full 8-bit character set (on client and server) to maximize the secret's search space.

Algorithm (uint8)

Provides the indication of the algorithm used for the generation of the credentials. The allowed values for this field are listed in Section 7.4.

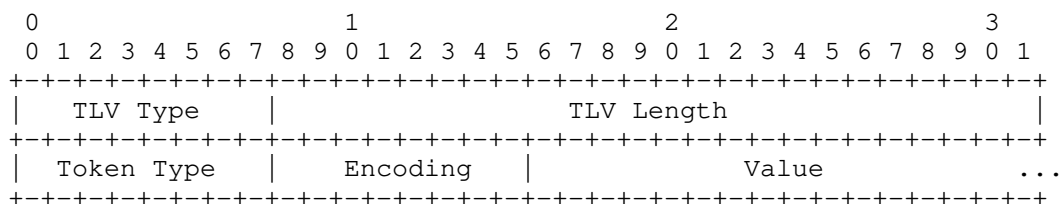
Object Identifier (binary; > 1 octet)

Provides the indication of additional parameters that are needed to be encoded for the credentials. This value is used only when the credentials use public-key cryptography - this field carries additional information about the generation algorithm to be used. We provide some useful values that can be used as reference:

OID Name	Dotted Representation	Binary Encoding
secp256r1 curve	1.2.840.10045.3.1.7	06 08 2A 86 48 CE 3D 03 01 07
secp384r1 curve	1.2.840.10045.3.1.34	06 08 2A 86 48 CE 3D 03 01 22
secp521r1 curve	1.2.840.10045.3.1.35	06 08 2A 86 48 CE 3D 03 01 23
X25519 curve	1.3.101.110	06 03 2B 65 6E
X25519 curve	1.3.101.110	06 03 2B 65 6E
X448 curve	1.3.101.111	06 03 2B 65 6F
Ed25519 curve	1.3.101.112	06 03 2B 65 70
Ed448 curve	1.3.101.113	06 03 2B 65 71

Table 3: Object Identifiers Examples

4.3.14. The Token-Data TLV



TLV Type (uint8)

<TBD> - Token-Data TLV

TLV Length (uint24)

Provides the length of the TLV (> 3 octets)

Token Type (uint8)

Provides the indication of the type of credentials. The allowed values for this field are listed in Section 7.2.

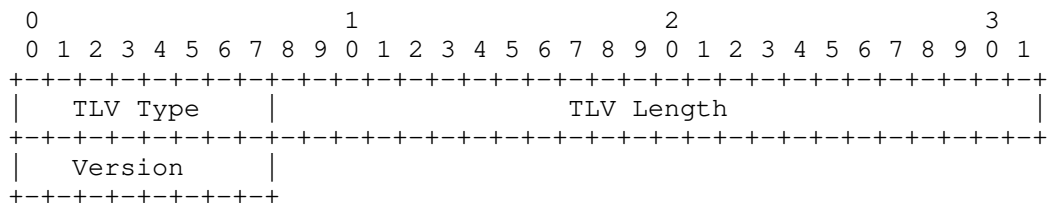
Encoding (uint8)

Provides the indication of the Encoding the credentials are in. The allowed values for this field are listed in Section 7.8.

Value (octet string)

This field carries the data for the credentials.

4.3.15. The Version TLV



TLV Type (uint8)

<TBD> - Version TLV

TLV Length (uint24)

Provides the length of the TLV. The field has a fixed value of 1.

Version (uint8)

The Version field represents the specific version of the EAP-CREDS protocol that are supported by the end point. When multiple versions of EAP-CREDS are supported, multiple ('Version') TLVs can be used.

When no version is specified (i.e., either it does not support multiple versions or it does not matter), the value of this field should be set to '0x0' (any version).

5. EAP-CREDS Messages

This section describes each message and what TLVs are allowed or required. EAP-CREDS defines the following values for the Message Type (Type):

Message Type	Name	Description
0	EAP-CREDS-Init	Initialization Phase
1	EAP-CREDS-Provisioning	Carries Provisioning Protocol Messages
2	EAP-CREDS-Validate	Validates newly installed credentials

Table 4: EAP-CREDS Message Types

5.1. The EAP-CREDS-Init Message

The EAP-CREDS-Init message type is used in Phase One only of EAP-CREDS. The message flow is depicted in Section 3.3. This message supports the following TLVs: Version, Protocol, Credentials-Info, and Error.

5.1.1. EAP Server's Init Message

EAP-CREDS starts with an ('EAP-CREDS-Init') message from the server. This message MAY contain zero, one, or more ('Version') TLVs and, optionally, a ('Challenge-Data') TLV.

The first message from the server is the one that starts Phase One, therefore the Server MUST set the headers' 'S' bit to '1' (Start) and the headers' 'Phase' value to '1' (Phase One).

The Server uses one or more ('Version') TLVs in the EAP-Request/EAP-CREDS(Type=Init) message to provide the Peer with the list of EAP-CREDS versions supported. If omitted, the implicit version of EAP-CREDS used in the session is one ('0x1'). If the Server detects multiple occurrences of this TLV in the reply from the Peer, an error shall be issued and the EAP-CREDS session should be terminated.

In case Token-Based registration is enabled on the Server, the Server MUST include, in its Init message, a ('Challenge-Data') field that can be used by the client to provide challenge data for proof-of-possession of secrets.

5.1.2. EAP Peer's Init Message

The Peer MUST reply to the Server's ('EAP-CREDS-Init') message with its own ('EAP-CREDS-Init') one. The Peer SHOULD include one ('Version') TLV in its first message to indicate the version of EAP-CREDS that the client wants to use for the session. The Peer MUST also provide the list of supported provisioning protocols (via one or more the 'Protocol' TLV), the list and status of the installed credentials (via the 'Credentials-Info' TLV). The Peer MAY include authorization data when registering new credentials (e.g., an authorization token or a device certificate) via the ('Token-Data') and ('Challenge-Response') TLV.

The Peer MUST include one ('Credentials-Info') TLV for each credential the Network is authorized to manage. Typically, a Peer will include only one ('Credentials-Info') TLV in its ('EAP-CREDS-Init') message, but there might be cases where multiple types of credentials are available and selected depending on the location and other factors (e.g., X.509 certificate and username/password combination).

In case the Peer does not have any credentials available yet, it does not add any ('Credentials-Info') TLV - leaving the Server with the only action possible: Registration. In this case, the Peer SHOULD include authorization information via the ('Token-Data') TLV as described in Section 5.1.2.1. Additionally, the Peer can add the ('Profile') TLV to indicate a preferred profile for the credentials.

5.1.2.1. Bootstrapping Peer's Trustworthiness

When the Peer does not have any valid credentials for the Network that it is authenticating to, it does not provide any ('Credentials-Info') TLV. This indicates to the Server that new credentials MUST be registered before the Peer is allowed on the network.

The Registration process might rely on information exchanged during the Provisioning Process in Phase Two. However, if an authorization mechanism is not available from the supported provisioning protocol and no credentials are available on the Peer, EAP-CREDS provides a simple mechanism for the Peer to leverage an out-of-band token/passphrase/ott that may be already available on the Peer (e.g., a device certificate or a 'spendable' credentials token like a

kerberos ticket or a crypto-currency transaction) and that can be verified by the Server.

In particular, when the Peer wants to register new credentials (and the Server requires the use of additional authorization data) it may need to provide (a) a Token, (b) a challenge value, and (c) a response to the challenge value. To do so, the Peer MUST encode the token in a ('Token-Data') TLV, the challenge value in a ('Challenge-Data') TLV, and, finally, the response to the challenge in the ('Challenge-Response') TLV.

The use of ('Challenge-Data') and ('Challenge-Response') TLVs is optional, however it is suggested that if a token is used for bootstrapping the trust, it should provide a way to verify a secret associated with it.

It is also very important that the authorization token is disclosed only to authorized servers - the Peer MUST NOT disclose authorization tokens that are not meant for the network that is being accessed. This can be done, usually, by verifying the identity of the Server first (in the outer mechanism) and then verify that the target of the Token is the Server the Client is talking to.

5.1.3. The EAP-CREDS-Provisioning Message

The EAP-CREDS-Provisioning message type is used in Phase Two only of EAP-CREDS. The message flow is depicted in Section 3.4. This message type supports the following TLVs: Protocol, Profile, Credentials-Info, Provisioning-Headers, Provisioning-Data, Token-Data, and Error.

After the exchange of phase one messages, the Server MAY start phase two by issuing an ('EAP-CREDS-Provisioning') message for the Peer where it encodes all the required details for starting the provisioning process. In particular, the server sends the selected ('Action'), ('Protocol'), and metadata to the client in a EAP-Request/EAP-CREDS(Type=Provisioning) message. The header's 'S' bit MUST be set to '1' (Start) and the 'Phase' value set to '2' (Phase Two begins).

NOTE WELL: After the initial message, the only TLVs that are allowed in messages coming from the server are the usual ('Provisioning-Headers') ('Provisioning-Data'), and ('Error').

The client checks that all the selected parameters are supported for the selected credentials and, if no errors are detected, it sends its first ('EAP-CREDS-Provisioning') message to the Server with the ('Provisioning-Headers') and ('Provisioning-Data') TLVs only.

From now on, the conversation between the Peer and the Server continues until an error is detected or the provisioning protocol completes successfully.

If no other actions, the server MAY continue with phase three or issue a success message and terminate the EAP session.

5.1.4. The EAP-CREDS-Validate Message

The EAP-CREDS-Validate message type is used in Phase Three only of EAP-CREDS. The message flow is depicted in Section 3.5. This message type supports the following TLVs: Protocol, Credentials-Info, Provisioning-Headers, Provisioning-Data, Token-Data, and Error.

After Phase One (and/or Phase Two) ends, the Server MAY start phase three by issuing an ('EAP-CREDS-Validate') message for the Peer where it encodes all the required details for starting the validation process. In particular, the server sends the ('Credentials-Info'), a ('Challenge'), and the ('Phase-Control') TLVs in a EAP-Request/EAP-CREDS(Type=Validate) message. The ('Phase-Control') TLV should carry the '1' value for the 'S' bit (Start) and the number '3' for its value (Phase Three begins).

The Peer generates the answer to the Challenge and sends back a EAP-Response/EAP-CREDS(Type=Validate) message with the ('Challenge-Response') and an optional ('Challenge') field (only for server-side validation of the symmetric credentials). If the Peer requested server-side validation of the credentials, the Server MUST include (if a symmetric secret) the response to the Peer-issued ('Challenge') TLV by computing the response and adding it to the ('Challenge-Response') TLV in its reply.

Finally, in the last message, the Server (if Phase Three is to be ended) SHALL include the ('Phase-Control') TLV with the 'S' bit set to '0' (end of phase) and the value set to '3' (Phase Three ended).

At this point, EAP-CREDS has terminated all possible operations and can be terminated. The Server can now terminate the EAP session successfully. In case the Peer was not authenticated during the tunnel establishment (i.e., no credentials were already available on the Peer), the Server should terminate the EAP session with a Failure (thus requiring the device to re-attach and authenticate to the network - phase two should have provided the Peer with the credentials to use for authenticating to the Network).

6. Error Handling in EAP-CREDS

This section provides a description of the error handling by using the CREDS-Error-TLV in a CREDS message.

7. IANA Considerations

This document uses a new EAP type, EAP-CREDS, whose value (TBD) MUST be allocated by IANA from the EAP TYPEs sub-registry of the RADIUS registry. This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP-CREDS protocol, in accordance with [RFC8126].

The EAP Method Type number for EAP-CREDS needs to be assigned.

This document also requires IANA to create new registries as defined in the following subsections.

7.1. Provisioning Protocols

Message Type	Purpose
0	Unspecified
1	Simple Provisioning Protocol (SPP)
2	Basic Certificate Management Protocol (CMP-S)
3	Full Certificate Management Protocol (CMP-F)
4	Enrollment over Secure Transport (EST)
5	Certificate Management over CMS (CMC)
6	Automatic Certificate Management Environment (ACME)
...	...
49141 ... 65534	Vendor Specific

Table 5: EAP-CREDS Inner Protocol Identifiers

Assignment of new values for new cryptosuites MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126].

7.2. Token Types

Token Type	Description
0	Unspecified
1	JWT
2	Kerberos
3	OAuth
4	Certificate
200..254	Vendor Specific

Table 6: Token Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.3. Credentials Types

Credentials Type	Description
0	X.509 Certificate
1	Public Key
2	Symmetric Key
3	Username and Password
4	AKA Subscriber Key
5	Bearer Token
6	One-Time Token
7	API Key

Table 7: Credentials Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.4. Credentials Algorithms

ID	Algorithm
0	None
1	RSA
2	ECDSA
3	XMMS
4	AKA Subscriber Key
5	OAuth
6	Kerberos4
7	Kerberos5
200-254	Reserved

Table 8: Credentials Algorithms

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.5. Credentials Datatypes

ID	Data Type
0	None (Binary)
1	PKCS#8
2	PKCS#10
3	PKCS#12
4	PublicKeyInfo
200-254	Reserved

Table 9: Credentials Datatypes

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.6. Challenge Types

ID	Data Type
0	Not Specified
1	EAP-CREDS-ASYMMETRIC
2	EAP-CREDS-SYMMETRIC

Table 10: Challenge Type

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.7. Network Usage Datatypes

ID	Data Type
0	Vendor-Specific
1	Manufacturer Usage Description [RFC8520]
2	Network Access Granting System
3	Firmware Manifest
4..127	Reserved for Future Use

Table 11: Network Usage Datatypes

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.8. Credentials Encoding

ID	Encoding
0	None (Raw)
1	DER
2	PEM
3	Base64
4	JSON
5	XML
6	ASCII
7	UTF-8
200-254	Reserved

Table 12: Credentials Encoding

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.9. Action Types

ID	Data Type	Description
0	Registration	Registers New Credentials
1	Renewal	Renew an Existing Credential
2	Remove	Removes an Existing Credential
200-254	n/a	Reserved

Table 13: Action Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

7.10. Usage Metadata Types

Type	Description
0	Binary (Unspecified)
1	MUD File
2	TEEP Manifest

Table 14: Usage Metadata Types

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

8. Security Considerations

Several security considerations need to be explicitly considered for the system administrators and application developers to understand the weaknesses of the overall architecture.

The most important security consideration when deploying EAP-CREDS is related to the security of the outer channel. In particular, EAP-CREDS assumes that the communication channel has been properly authenticated and that the information exchanged between the Peer and the Server are protected (i.e., confidentiality and integrity).

For example, if certificate-based authentication is used, the server presents a certificate to the peer as part of the trust establishment (or negotiation). The peer SHOULD verify the validity of the EAP server certificate and SHOULD also examine the EAP server name presented in the certificate in order to determine whether the EAP server can be trusted. When performing server certificate

validation, implementations MUST provide support for the rules in [RFC5280] for validating certificates against a known trust anchor.

9. Acknowledgments

The authors would like to thank everybody who provided insightful comments and helped in the definition of the deployment considerations.

10. 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. Levkowitz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <<https://www.rfc-editor.org/info/rfc4210>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <<https://www.rfc-editor.org/info/rfc6402>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

- [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>>.
- [RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <<https://www.rfc-editor.org/info/rfc8520>>.

Author's Address

Massimiliano Pala
CableLabs
858 Coal Creek Cir
Louisville, CO 80027
US

Email: m.pala@openca.org
URI: <http://www.linkedin.com/in/mpala>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 5, 2020

J. Rieckers
Uni Bremen
November 02, 2019

X509v3 EAP Parameter Extension
draft-rieckers-eapparameterextension-00

Abstract

This document specifies an extension to X509v3 certificates for EAP-TLS servers to mitigate some flaws in the specification to the use of TLS in EAP as specified in RFC5216. The specified extension enables clients to decide whether to trust the certificate presented by the EAP-TLS server by including information implicitly defined by login credentials or communication context in the server certificate.

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 May 5, 2020.

Copyright Notice

Copyright (c) 2019 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	2
2. Definitions	3
2.1. Requirements Language	3
2.2. Terminology	3
3. Syntax	4
4. EAP Parameter Types	4
4.1. Realm Suffix types	4
4.1.1. Syntax	5
4.1.2. Validation	5
4.1.3. CA Requirements	5
4.2. Identity	5
4.2.1. Syntax	5
4.2.2. Validation	6
4.2.3. CA Requirements	6
4.3. Login Medium	6
4.3.1. Syntax	6
4.3.2. Validation	6
4.3.3. CA Requirements	6
4.4. Handling of future EAP Parameter Types	7
5. IANA Considerations	7
6. Security Considerations	8
7. References	8
7.1. Normative References	8
7.2. Informative References	9
Appendix A. ASN.1 Modul	9
Acknowledgements	9
Author's Address	9

1. Introduction

Logging in with EAP-TLS based methods is a widely used mechanism for password based login with protocols like RADIUS [RFC2865]. Two mechanisms used in WPA2-Enterprise areas are EAP-TTLSv0 and EAP-PEAPv0. Unfortunately, the specification of EAP-TLS does not specify how the EAP-TLS peer can verify that the certificate presented by the server is valid apart from the Key Usage identifiers and user set configuration parameters.

The configuration parameters include especially the information about the trust anchor and the expected domain.

In contrast to the usage of X509v3 certificates in other contexts, such as HTTPS, in EAP-TLS the expected name can not be distinguished

from the context of the communication. This requires users to configure their supplicants accordingly. Especially in large setups with private devices this has led to insecure configurations with insufficient or even wrong name checks. Some security considerations for EAP-TLS in deployment in eduroam have been named in [RFC7593], Section 7.1.

The same basic security considerations apply for the certificate based login methods as they apply for password based methods, but are not that critical, since an attacker can not gain knowledge of the supplicant's private key with an attack based on insufficient server certificate validation done by the peer.

The aim of the extension introduced in this document is to give EAP-TLS peers an option to check the certificate of the EAP-TLS server against parameters implicitly defined by the communication context.

2. Definitions

2.1. 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.

2.2. Terminology

Readers are expected to be familiar with the terms and concepts used in EAP [RFC3748] and EAP-TLS [RFC5216].

In particular, this document frequently uses the following terms:

EAP-TLS server: The entity that authenticates the clients and is termination point of the EAP-TLS tunnel.

peer or supplicant: The client that authenticates to the server in order to gain access to the protected resource.

EAP Identity: The Identity sent by the peer in the first (unencrypted) EAP Identity Response as specified in [RFC3748] Section 5.1.

3. Syntax

TODO

There shall be a ASN.1 module in Appendix A

The EAP Parameter extension has the following format:

```
id-pe-eapparameter OBJECT IDENTIFIER ::= { id-pe XX }
```

```
EAPParameterValues ::= SEQUENCE {  
    EAPParameterType    OBJECT IDENTIFIER,  
    EAPParameterValue    OCTET STRING  
}
```

```
EAPParameter ::= SEQUENCE OF (1..MAX) EAPParameterValues
```

The extnValue of the id-pe-eapparameter extension is the ASN.1 DER encoding of the EAPParameter structure.

The EAP Parameter extension MAY be marked as critical. Certificate Authorities SHOULD allow both, critical and not critical, in their application process for a certificate with this extension, so applicants can choose. Making the extension critical may not be desirable in the early future after the release of this RFC, but, when marked critical, it will help forcing users to update their devices, which might be, at least in the authors opinion, a good idea. This makes use of the specification of the handling of critical extensions, specified in [RFC5280]: Any supplicant not understanding a critical extension MUST reject the certificate if it does not understand this extension.

4. EAP Parameter Types

This RFC specifies several EAP Parameter Types. Other parameter types MAY be specified in the future. The handling of new parameters is described in Section 4.4

4.1. Realm Suffix types

The Realm Suffix types can be used to bind the certificate to a specific realm. It is either a realm separated by '@' (EAPPARAMETERBASEOID.1) or a realm separated by '%' (EAPPARAMETERBASEOID.2)

TODO: Reference to NAI [RFC7542] for Realms. Maybe even remove the '%' seperated realm, since I have not found any usage. This is only here because it is in a default configuration file of FreeRADIUS

TODO: Replace EAPPARAMETERBASEOID with assigned OID

4.1.1. Syntax

The EAPParameterValue for these types is a DER-encoded UTF8String containing the full realm of the outer username. This is supposed to be a FQDN. It MAY not be a FQDN for testing purposes but MUST NOT contain the separator character, depending on the Suffix type. The value MAY contain one asterisk to indicate a wildcard validity for all realms under a specific domain. The asterisk MUST be the first character and MUST be followed by a dot. A certificate containing only an asterisk and a dot MUST NOT be issued by a trusted certificate authority.

4.1.2. Validation

To verify the supplicant will compare the sent EAP Identity with the realm contained in the EAPParameterValue. If the EAPParameterValue is not in a valid format, the supplicant MUST reject the certificate and SHOULD send a fatal bad_certificate alert (see [RFC5246], [RFC8446]). If the value contains an asterisk, the realm part should be matched as the dnsName of subjectAltName attribute would be matched.

4.1.3. CA Requirements

A trusted CA MUST validate that the applicant is authorized to request certificates under the domain represented by the realm. CAs SHOULD rely additionally on the CAA issuewild DNS value and SHOULD NOT issue a certificate with this extension if the CAA value forbids wildcard certificates.

4.2. Identity

EAPPARAMETERBASEOID.3

The Identity Parameter Type can be used to bind the server certificate to use with a specific EAP Identity.

4.2.1. Syntax

The EAPParameterValue for this type is a DER-encoded UTF8String containing the full EAP Identity.

The value MAY contain up to two asterisks, one for the local part of the Identity, one for the realm. This can be used to allow EAP-Identities with variable parts. If two asterisks are used, the

certificate extension MUST also contain a EAPParameter for verification of the Realm (e.g. Realm Suffix, separated by '@')

The asterisk for the realm part follows the same rules as described in the Realm suffix type.

The asterisk for the local part MAY be at any position.

4.2.2. Validation

To verify this parameter the supplicant has to verify the sent EAP Identity and the parameter value in the certificate match. The supplicant SHOULD do this with a case insensitive comparison.

TODO: Here should also be a description how to deal with asterisks.

4.2.3. CA Requirements

The CA requirements for this type depend on the used Identity format.

If the Identity is in a format that would also allow Realm Suffix Types (e.g. separated by @ or %) for the domain part the same CA Requirements as for the defined Realm Suffix Types apply. CAs SHOULD allow the local part to be chosen by the certificate requestor inside normal parameters.

Globally trusted CAs MUST NOT issue certificates with the Identity EAP Parameter Type if it does not contain a Realm or the Realm can not be mapped to a DNS name.

4.3. Login Medium

EAPPARAMETERBASEOID.4

TODO

This is intended to limit the validity of the certificate to e.g. 802.1x authentication

4.3.1. Syntax

4.3.2. Validation

4.3.3. CA Requirements

4.4. Handling of future EAP Parameter Types

TODO

5. IANA Considerations

TODO: Here the IANA considerations should be updated. The decimal id of the id-pe-eapparameter will be registered once the draft has reached a state where proof of concept implementations can be made.

On approval, IANA shall add in the SMI Security for PKIX Certificate Extension (1.3.6.1.5.5.7.1) registry the following entry:

Decimal	Description	References
XX	id-pe-eapparameter	{this RFC}

Additionally the IANA shall install a new registry for PKIX Certificate Extension EAP Parameter Types for the parameter types with the following initial content:

Decimal	Description	References
1	Realm Suffix, separated by '@'	{this RFC}
2	Realm Suffix, separated by '%'	{this RFC}
3	Identity	{this RFC}
4	Login Medium	{this RFC}

TODO: Here there may be also defined Realm Prefix Types (e.g. WINDOWS-NET/username). Obviously this can't be issued by a globally trusted CA, but might be issued by a company CA

Further EAP Parameter Types may be registered in future. New registration requests MUST include a detailed description how peers should validate the given parameter and a detailed description for Certificate Authorities how they must verify the authorization of a certificate request with this parameter.

6. Security Considerations

TODO

There will be security considerations. There are security considerations which lead to this draft.

Here might be a reference to [RFC4334]. It specifies X509v3 extensions to help the supplicant to choose the client certificate used for login based on connection parameters such as SSID or Login Medium.

7. References

7.1. 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>>.
- [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>>.
- [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>>.
- [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>>.

7.2. Informative References

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

Appendix A. ASN.1 Modul

This is obviously also an open TODO

Acknowledgements

There will be acknowledgements. I haven't done the work all by myself and a lot of people should and will be listed here for supporting me in all my work.

Carsten Bormann, ...

Author's Address

Jan-Frederik Rieckers
Universitaet Bremen
Bibliothekstr. 5
Bremen 28359
Germany

Email: rieckers@uni-bremen.de