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

DeKok, Alan
FreeRADIUS

5 March 2022

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

Abstract

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

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

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

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

Table of Contents

1. Introduction

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 All of these methods use key derivation functions which are no longer applicable to TLS 1.3.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

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

1.1. Requirements Language

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

2. Using TLS-based EAP methods with TLS 1.3

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

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

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

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

2.1. Key Derivation

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

> Type = value of the EAP Method type

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

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

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

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

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

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

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

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

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

2.2. TEAP

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

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

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

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

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

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

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

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

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

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

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

```
CMK = CMK[j]

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

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

2.3. FAST

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

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

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

IMSK instead of MSK.

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

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

2.4. TTLS

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

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

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

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

2.4.1. Client Certificates

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

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

2.5. PEAP

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

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

> ... the TK is the first 60 octets of the Key_Material, as specified in [RFC5216]: TLS-PRF-128 (master secret, "client EAP encryption", client.random || server.random).

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

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

2.5.1. Client Certificates

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

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

3. Application Data

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

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

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

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

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

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

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

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

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

The NewSessionTicket message SHOULD also be sent along with other

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

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

3.1. Identities

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

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

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

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

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

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

redundant.

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

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

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

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

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

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

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

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

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

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

4. Resumption

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

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

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

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

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

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

5. Implementation Status

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

The wpa_supplicant implementation requires that a configuration flag be set "tls_disable_tlsv1_3=0", and describes the flag as "enable TLSv1.3 (experimental - disabled by default)". However, interoperability testing shows that PEAP and TTLS both work with Radiator and FreeRADIUS.

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

6. Security Considerations

[RFC9190] Section 5 is included here by reference.

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

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

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

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

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

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

6.1. Protected Success and Failure indicators

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

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

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

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

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

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

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

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

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

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

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

7. IANA Considerations

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

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

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

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

* TEAPbindkey@ietf.org

8. References

8.1. Normative References

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

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

[RFC5216]
Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008

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

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

[RFC7170]
Zhou, H., et al., "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, May 2014.

[RFC8126]
Cotton, M., et al, "Guidelines for Writing an IANA Considerations Section in RFCs", 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.

[RFC9190]
Mattsson, J., and Sethi, M., "Using EAP-TLS with TLS 1.3", draft-

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

[IANA]
https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml#eap-numbers-4

8.2. Informative References

[MSPEAP]
https://msdn.microsoft.com/en-us/library/cc238354.aspx

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

[PEAP-MPPE]
https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/e75b0385-915a-4fc3-a549-fd3d06b995b0

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

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

[RFC1994]
Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[RFC2433]
Zorn, G. and Cobb, S., "Microsoft PPP CHAP Extensions", RFC 2433, October 1998.

[RFC2759]
Zorn, G., "Microsoft PPP CHAP Extensions, Version 2", RFC 2759, January 2000.

[RFC4137]
Vollbrecht, J., et al, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator ", RFC 4137, August 2005.

[RFC4851]
Cam-Winget, N., et al, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

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

[RFC5422]
Cam-Winget, N., et al, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, March 2009.

[RFC7542]
DeKoK, A, "The Network Access Identifier", RFC 7542, May 2015.

[RFC7585]
Winter, S, and McCauley, M., "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, October 2015.

Acknowledgments

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

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

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org