Network Working Group
INTERNET-DRAFT
Updates: 5247, 5281, 7170
Category: Standards Track
Expires: August 21, 2021

DeKok, Alan
FreeRADIUS

21 February 2021

TLS-based EAP types and TLS 1.3
draft-ietf-emu-tls-eap-types-02.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/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.

Copyright Notice

Table of Contents

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 which 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. Unless otherwise discusses herein, implementations of EAP methods that wish to use TLS 1.3 MUST follow the guidelines in [EAPTLS].

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

An additional difference is that the [EAPTLS] Section 2.5 requires a Commitment Message to be sent once the EAP-TLS handshake has completed. Other TLS-based EAP methods also use the Commitment Message, but only during resumption. When the other TLS-based EAP methods send application data inside of the TLS tunnel, the Commitment Message is not used. This topic is explained in more detail below, in Section 3.

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 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 255. Where expanded EAP Type Codes are used, the Type-Code is defined to be the Expanded Type Code (including the Type, Vendor-Id (in network byte order) and Vendor-Type fields (in network byte order) defined in [RFC3748] Section 5.7).

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

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 they 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.

It is RECOMMENDED that vendor-defined TLS-based EAP methods use the above definitions for TLS 1.3. There is insufficient 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. We update that definition for TLS 1.3 as:

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

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-Code.

```
session_key_seed = TLS-Exporter("EXPORTER: session key seed",
Type-Code, 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. 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 hash function used is the same as the ciphersuite hash function 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 used as 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-Code must be used when deriving the session_key_seed, and not the TEAP Type-Code.

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 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.

The T-PRF given in [RFC4851] Section 5.5 is not used for TLS 1.3.

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 no "context_value" ([RFC8446] Section 7.5) passed to the TLS-Exporter function. The value "n" given here is the length of the challenge required, which varies according to the challenge.

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 truncated output will be different from the output calculated using the correct length.

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 instead be derived from the Key_Material defined above in Section 2.1.

3. Application Data

Unlike previous TLS versions, TLS 1.3 can continue 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 one or more "NewSessionTicket" messages after the "Finished" state. This change can cause many implementations to fail.

In order to correct this failure, if the underlying TLS connection is still performing negotiations, 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.

[EAPTLS] Section 2.5 requires a Commitment message 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.

4. Resumption

[EAPTLS] 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 type code is different.

All TLS-based EAP methods support resumption. All EAP servers and peers MUST support resumption. 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 faster authentications, and less load on the EAP server.

EAP servers peers MUST NOT resume sessions across different EAP types, and EAP servers MUST reject resumptions in which the EAP Type code is different from the original authentication.

As the packet flows for resumption are essentially identical across all TLS-based EAP types, it is technically possible to authenticate using EAP-TLS (EAP Type code 13), and then perform resumption using another EAP type, just as EAP-TTLS (EAP Type code 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.

5. 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 privace 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 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.

For PEAP, some derivation use HMAC-SHA1 [PEAP-MPPE]. There are no known security issues with HMAC-SHA1. In the interests of interoperability and minimal changes, we do not change that definition here.

5.1. Protected Success and Failure indicators

[EAPTLS] 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 authenticaton, 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 Commitment Message is not used, and there is no TLS-layer alert sent when the inner authentication fails. Instead, there is simple 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.

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 indicates, but the final result MUST be a failed authentication.

Similarly, when the inner authentication protocol indicates that authentication has succeeed, 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.

6. 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 2.

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

* EXPORTER: session key seed * EXPORTER: Inner Methods Compound Keys * EXPORTER: Session Key Generating Function * EXPORTER: Extended Session Key Generating Function * TEAPbindkey@ietf.org

7. References

7.1. 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-tls13-14, February, 2021.

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

7.2. 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, March 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.

Acknowledgments

Thanks to Jorge Vergara for a detailed review of the requirements for various EAP types, and for assistance with interoperability testing.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org