

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 25, 2011

Y. Nir
Check Point
Y. Sheffer
Independent
H. Tschofenig
NSN
P. Gutmann
University of Auckland
June 23, 2011

A Flexible Authentication Framework for the Transport Layer Security
(TLS) Protocol using the Extensible Authentication Protocol (EAP)
draft-nir-tls-eap-12

Abstract

Many of today's Web security problems have their root in the widespread usage of weak authentication mechanisms bundled with the usage of password based credentials. Dealing with both of these problems is the basis of this publication.

This document extends the Transport Layer Security (TLS) protocol with a flexible and widely deployed authentication framework, namely the Extensible Authentication Protocol (EAP), to improve security of Web- as well as non-Web-based applications. The EAP framework allows so-called EAP methods, i.e. authentication and key exchange protocols, to be plugged into EAP without having to re-design the underlying protocol. The benefit of such an easy integration is the ability to run authentication protocols that fit a specific deployment environment, both from a credential choice as well as from the security and performance characteristics of the actual protocol.

This work follows the example of IKEv2, where EAP has been added to allow clients to seamlessly use different forms of authentication credentials, such as passwords, token cards, and shared secrets.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

Internet-Drafts are draft documents valid for a maximum of six months

Internet-Draft

TLS with EAP Extension (TEE)

June 2011

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 25, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) 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.	EAP Applicability	5
1.2.	Comparison with Design Alternatives	5
1.3.	Conventions Used in This Document	5
2.	Operating Environment	6
3.	Protocol Overview	7
3.1.	The tee_supported Extension	8
3.2.	The EapFinished Handshake Message	8
3.3.	The EapMsg Handshake Message	9
3.4.	Calculating the EapFinished message	9
4.	Security Considerations	11
4.1.	EapFinished vs. Finished	11
4.2.	Identity Protection	11
4.3.	Mutual Authentication	12
5.	Performance Considerations	13
6.	Operational Considerations	14
7.	IANA Considerations	15
8.	Acknowledgments	16
9.	Changes from Previous Versions	17
9.1.	Changes in version -02	17
9.2.	Changes in version -01	17
9.3.	Changes from the protocol model draft	17
10.	References	18
10.1.	Normative References	18
10.2.	Informative References	18
	Authors' Addresses	20

1. Introduction

This document describes a new extension to [\[TLS\]](#). This extension allows a TLS client to authenticate using [\[EAP\]](#) instead of performing the authentication at the application level. The extension follows [\[TLS-EXT\]](#). For the remainder of this document we will refer to this extension as TEE (TLS with EAP Extension).

TEE extends the TLS handshake beyond the regular setup, to allow the EAP protocol to run between the TLS server (called an "authenticator" in EAP) and the TLS client (called either a "supplicant" or a "peer"). This allows the TLS architecture to handle client authentication before exposing the server application software to an unauthenticated client. In doing this, we follow the approach taken for IKEv2 in [\[RFC5996\]](#). However, similar to regular TLS, we protect the user identity by only sending the client identity after the server has authenticated. In this our solution differs from that of IKEv2.

Currently used applications that rely on non-certificate user credentials use TLS to authenticate the server only. After that, the application takes over, and presents a login screen where the user is expected to present their credentials.

This creates several problems. It allows a client to access the application before authentication, thus creating a potential for anonymous attacks on non-hardened applications. Additionally, web pages are not particularly well suited for long shared secrets and for interfacing with certain devices such as USB tokens.

TEE allows full mutual authentication to occur for all these applications within the TLS exchange. The application receives control only when the user is identified and authenticated. The authentication can be built into the server infrastructure by connecting to an AAA server. The client side can be integrated into client software such as web browsers and mail clients. An EAP infrastructure is already built into some operating systems providing a user interface for each authentication method within EAP.

We intend TEE to be used for various protocols that use TLS such as HTTPS, in cases where certificate based client authentication is not practical. This includes web-based mail services, online banking, premium content websites and mail clients.

Another class of applications that may see benefit from TEE are TLS based VPN clients used as part of so-called "SSL VPN" products. No such client protocols have so far been standardized.

1.1. EAP Applicability

Section 1.3 of [[EAP](#)] states that EAP is only applicable for network access authentication, rather than for "bulk data transfer". It then goes on to explain why the transport properties of EAP indeed make it unsuitable for bulk data transfer, e.g. for large file transport. Our proposed use of EAP falls squarely within the applicability as defined, since we make no further use of EAP beyond access authentication.

1.2. Comparison with Design Alternatives

It has been suggested to implement EAP authentication as part of the protected application, rather than as part of the TLS handshake. A BCP document could be used to describe a secure way of doing this. The drawbacks we see in such an approach are listed below:

- o EAP does not have a pre-defined transport method. Application designers would need to specify an EAP transport for each application. Making this a part of TLS has the benefit of a single specification for all protected applications.
- o The integration of EAP and TLS is security-sensitive and should be standardized and interoperable. We do not believe that it should

be left to application designers to do this in a secure manner. Specifically on the server-side, integration with AAA servers adds complexity and is more naturally part of the underlying infrastructure.

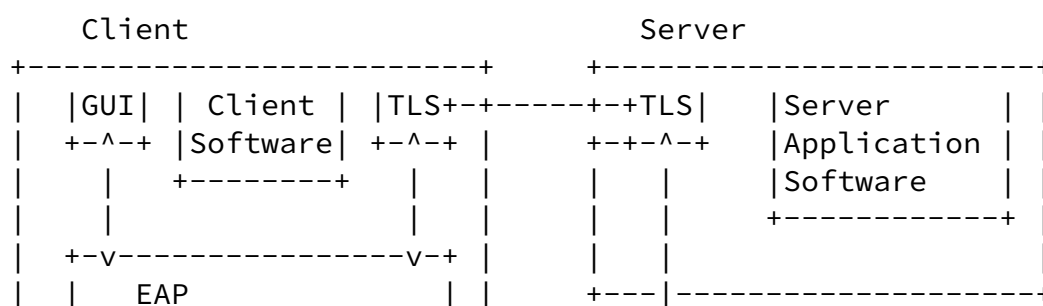
- o Our current proposal provides channel binding between TLS and EAP, to counter the MITM attacks described in [\[MITM\]](#). TLS does not provide any standard way of extracting cryptographic material from the TLS state, and in most implementations, the TLS state is not exposed to the protected application. Because of this, it is difficult for application designers to bind the user authentication to the protected channel provided by TLS.

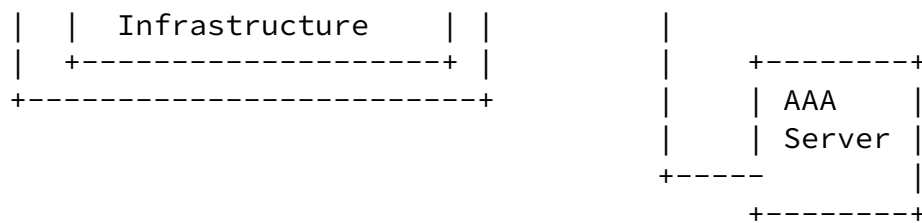
[1.3.](#) Conventions Used in This Document

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.](#) Operating Environment

TEE will work between a client application and a server application, performing either client authentication or mutual authentication within the TLS exchange.





The above diagram shows the typical deployment. The client has software that either includes a UI for some EAP methods, or else is able to invoke some operating system EAP infrastructure that takes care of the user interaction. Although the technical mechanisms have been standardized for a AAA client to dynamically discover a AAA server often the address of the AAA server is statically configured. Typically the AAA server communicates using the RADIUS protocol with EAP ([\[RADIUS\]](#) and [\[RAD-EAP\]](#)), or the Diameter protocol ([\[Diameter\]](#) and [\[Dia-EAP\]](#)).

As stated in the introduction, we expect TEE to be used in both browsers and applications. Further uses may be authentication and key generation for other protocols, and tunneling clients, which so far have not been standardized.

[3.](#) Protocol Overview

When TLS is used with EAP, additional records are sent after the ChangeCipherSpec protocol message and before the Finished message, effectively creating an extended handshake before the application layer data can be sent. Each EapMsg handshake record contains exactly one EAP message. Using EAP for client authentication allows TLS to be used with various AAA back-end servers such as RADIUS or

Diameter.

TLS with EAP may be used for securing a data connection such as HTTP or POP3. We believe it has three main benefits:

- o The ability of EAP to work with backend servers can remove that burden from the application layer.
- o Moving the user authentication into the TLS handshake protects the presumably less secure application layer from attacks by unauthenticated parties.
- o Using mutual authentication methods within EAP can help thwart certain classes of phishing attacks.

The TEE extension defines the following:

- o A new extension type called `tee_supported`, used to indicate that the communicating application (either client or server) supports this extension.
- o A new message type for the handshake protocol, called `EapMsg`, which is used to carry a single EAP message.
- o A new message type for the handshake protocol, called `EapFinished`, which is used to sign previous messages.

The diagram below outlines the protocol structure. For illustration purposes only, we use EAP Generalized Pre-Shared Key (EAP-GPSK) method [[RFC5433](#)]. This method is a lightweight shared-key authentication protocol supporting mutual authentication and key derivation.


```

-----
ClientHello(*)          ----->
                               ServerHello(*)
                               (Certificate)
                               ServerKeyExchange
                               EapMsg(Identity-Request)
                               <----- ServerHelloDone
ClientKeyExchange
(CertificateVerify)
ChangeCipherSpec
Finished
EapMsg(Identity-Reply)  ----->
                               ChangeCipherSpec
                               Finished
                               EapMsg(GPSK-Request)
                               <-----
EapMsg(GPSK-Reply)      ----->
                               EapMsg(GPSK-Request)
                               <-----
EapMsg(GPSK-Reply)      ----->
                               EapMsg(Success)
                               EapFinished
EapFinished             ----->

```

(*) The ClientHello and ServerHello include the tee_supported extension to indicate support for TEE

The client indicates in the first message its support for TEE. The server sends an EAP identity request in the reply. The client sends the identity reply after the handshake completion. The EAP request-response sequence continues until the client is either authenticated or rejected.

[3.1.](#) The tee_supported Extension

The tee_supported extension is a ClientHello and ServerHello extension as defined in section 2.3 of [\[TLS-EXT\]](#). The extension_type field is TBA by IANA. The extension_data is zero-length.

[3.2.](#) The EapFinished Handshake Message

The EapFinished message is identical in syntax to the Finished message described in section 7.4.9 of [\[TLS\]](#). It is calculated in exactly the same way.

When TEE is used, application data cannot follow the "Finished" message. Instead, it may only begin after the EapFinished message

The HandshakeType value for the EapFinished handshake message is TBA by IANA.

[3.3.](#) The EapMsg Handshake Message

The EapMsg handshake message carries exactly one EAP message as defined in [\[EAP\]](#).

The HandshakeType value for the EapMsg handshake message is TBA by IANA.

The EapMsg message is used to tunnel EAP messages between the authentication server, which may be co-located with the TLS server, or else may be a separate AAA server, and the supplicant, which is co-located with the TLS client. TLS on either side receives the EAP data from the EAP infrastructure, and treats it as opaque. TLS does not make any changes to the EAP payload or make any decisions based on the contents of an EapMsg handshake message.

Note that it is expected that the authentication server notifies the TLS server about authentication success or failure, and so TLS need not inspect the eap_payload within the EapMsg to detect success or failure.

```
struct {  
    opaque eap_payload[4..65535];  
} EapMsg;
```

eap_payload is defined in [section 4 of RFC 3748](#). It includes the Code, Identifier, Length and Data fields of the EAP packet.

[3.4.](#) Calculating the EapFinished message

If the EAP method is key-generating (see [\[RFC5247\]](#)), the Finished message is calculated as follows:

```
struct {  
    opaque verify_data[12];  
} Finished;  
  
verify_data  
    PRF(MSK, finished_label, MD5(handshake_messages) +  
    SHA-1(handshake_messages)) [0..11];
```

The finished_label and the PRF are as defined in [section 7.4.9](#) of

[\[TLS\]](#).

The handshake_messages field does not include all the data in the handshake. Instead it includes only the data that has not been signed by the previous Finished message. The handshake_messages field includes all of the octets beginning with and including the Finished message, up to but not including this EapFinished message. This is the concatenation of all the Handshake structures exchanged thus far, and not yet signed, as defined in section 7.4 of [\[TLS\]](#) and in this document.

The Master Session Key (MSK) is derived by the AAA server and by the client if the EAP method is key-generating. On the server-side, it is typically received from the AAA server over the RADIUS or Diameter protocol. On the client-side, it is passed to TLS by some other method.

If the EAP method is not key-generating, then the master_secret is used to sign the messages instead of the MSK. For a discussion on the use of such methods, see [Section 4.1](#).

[4.](#) Security Considerations

[4.1.](#) EapFinished vs. Finished

In regular TLS, the Finished message provides two functions: it signs all preceding messages, and it signals that application data can now be sent. In TEE, it only signs, and the signal is given by EapFinished.

Many EAP methods, such as EAP-TLS, EAP-IKEv2 and EAP-SIM, generate keys in addition to authenticating clients. Such methods are resistant to man-in-the-middle (MITM) attacks, as discussed in [[MITM](#)] and are called key-generating methods.

To realize the benefit of such methods, we need to verify the key that was generated within the EAP method. This is referred to as the MSK in EAP. In TEE, the EapFinished message signs the messages that have not yet been signed by the Finished message using the MSK if such exists. If not, then the messages are signed with the master_secret as in regular TLS.

The need for signing twice arises from the fact that we need to use both the master_secret and the MSK. It was possible to use just one Finished record and blend the MSK into the master_secret. However, this would needlessly complicate the protocol and make security analysis more difficult. Instead, we have decided to follow the example of IKEv2, where two AUTH payloads are exchanged.

It should be noted that using non-key-generating methods may expose the client to a MITM attack if the same method and credentials are used in some other situation, in which the EAP is done outside of a protected tunnel with an authenticated server. Unless it can be

determined that the EAP method is never used in such a situation, non-key-generating methods SHOULD NOT be used. This issue is discussed extensively in [[Compound-Authentication](#)].

[4.2.](#) Identity Protection

Unlike [[TLS-PSK](#)], TEE provides identity protection for the client. The client's identity is hidden from a passive eavesdropper using TLS encryption. Active attacks are discussed in [Section 4.3](#).

We could save one round-trip by having the client send its identity within the Client Hello message. This is similar to TLS-PSK. However, we believe that identity protection is a worthy enough goal, so as to justify the extra round-trip.

Nir, et al.

Expires December 25, 2011

[Page 11]

Internet-Draft

TLS with EAP Extension (TEE)

June 2011

[4.3.](#) Mutual Authentication

In order to achieve our security goals, we need to have both the server and the client authenticate. Client authentication is obviously done using the EAP method. The server authentication can be done in either of two ways:

1. The client can verify the server certificate. This may work well depending on the scenario, but implies that the client or its user can recognize the right DN or alternate name, and distinguish it from plausible alternatives. The introduction to [[I.D.Webauth-phishing](#)] shows that at least in HTTPS, this is not always the case.
2. The client can use a mutually authenticated (MA) EAP method, such as EAP-GPSK. The client would be authenticated to the AAA server and vice versa. Additionally authenticating the application server to the client is not necessary and the TLS handshake may as well be anonymous. Note that the authenticated client identity may be sent from the AAA server to the application server (acting as a AAA client) if the authenticated identity indeed matters for the purpose of the service fulfillment and with the user's permission.

To summarize:

- o Clients MUST NOT propose anonymous ciphersuites, unless they support MA EAP methods.

- o Clients MUST NOT accept non-MA methods if the ciphersuite is anonymous.
- o Clients MUST NOT accept non-MA methods if they are not able to verify the server credentials. Note that this document does not define what verification involves. If the server DN is known and stored on the client, verifying certificate signature and checking revocation may be enough. For web browsers, the case is not as clear cut, and MA methods SHOULD be used.

[5.](#) Performance Considerations

Regular TLS adds two round-trips to a TCP connection. However, because of the stream nature of TCP, the client does not really need to wait for the server's Finished message, and can begin sending application data immediately after its own Finished message. In practice, many clients do so, and TLS only adds one round-trip of delay.

TEE adds as many round-trips as the EAP method requires. For example, EAP-MD5 requires 1 round-trip, while EAP-GPSK requires 2 round-trips. Additionally, the client MUST wait for the EAP-Success message before sending its own Finished message, so we need at least 3 round-trips for the entire handshake. The best a client can do is two round-trips plus however many round-trips the EAP method requires.

It should be noted, though, that these extra round-trips save

processing time at the application level. Two extra round-trips take a lot less time than presenting a log-in web page and processing the user's input.

It should also be noted, that TEE reverses the order of the Finished messages. In regular TLS the client sends the Finished message first. In TEE it is the server that sends the Finished message first. This should not affect performance, and it is clear that the client may send application data immediately after the Finished message.

[6.](#) Operational Considerations

[Section 4.3](#) defines a dependency between the TLS state and the EAP state in that it mandates that certain EAP methods should not be used with certain TLS ciphersuites. To avoid such dependencies, there are two approaches that implementations can take. They can either not use any anonymous ciphersuites, or else they can use only MA EAP methods.

Where certificate validation is problematic, such as in browser-based HTTPS, we recommend the latter approach.

In cases where the use of EAP within TLS is not known before opening the connection, it is necessary to consider the implications of requiring the user to type in credentials after the connection has already started. TCP sessions may time out, because of security considerations, and this may lead to session setup failure.

Nir, et al.	Expires December 25, 2011	[Page 14]
-------------	---------------------------	-----------

Internet-Draft	TLS with EAP Extension (TEE)	June 2011
----------------	------------------------------	-----------

[7.](#) IANA Considerations

IANA is asked to assign an extension type value from the "ExtensionType Values" registry for the tee_supported extension.

IANA is asked to assign two handshake message types from the "TLS HandshakeType Registry", one for "EapMsg" and one for "EapFinished".

8. Acknowledgments

The authors would like to thank Josh Howlett for his comments.

The TLS Inner Application Extension work ([TLS/IA]) has inspired the authors to create this simplified work. TLS/IA provides a somewhat different approach to integrating non-certificate credentials into the TLS protocol, in addition to several other features available from the RADIUS namespace.

The authors would also like to thank the various contributors to [[RFC5996](#)] whose work inspired this one.

[9.](#) Changes from Previous Versions

[9.1.](#) Changes in version -02

- o Added discussion of alternative designs.

[9.2.](#) Changes in version -01

- o Changed the construction of the Finished message
- o Replaced MS-CHAPv2 with GSK in examples.
- o Added open issues section.
- o Added reference to [[Compound-Authentication](#)]
- o Fixed reference to MITM attack

[9.3.](#) Changes from the protocol model draft

- o Added diagram for EapMsg
- o Added discussion of EAP applicability
- o Added discussion of mutually-authenticated EAP methods vs other methods in the security considerations.
- o Added operational considerations.
- o Other minor nits.

[10.](#) References

[10.1.](#) Normative References

- [EAP] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", [RFC 3748](#), June 2004.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), August 2008.
- [TLS-EXT] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", [RFC 4366](#), April 2006.

[10.2.](#) Informative References

- [Compound-Authentication] Puthenkulam, J., Lortz, V., Palekar, A., and D. Simon, "The Compound Authentication Binding Problem", [draft-puthenkulam-eap-binding-04](#) (work in progress), October 2003.
- [Dia-EAP] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", [RFC 4072](#), August 2005.
- [Diameter] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", [RFC 3588](#), September 2003.

- [I.D.Webauth-phishing] Hartman, S., "Requirements for Web Authentication Resistant to Phishing", [draft-hartman-webauth-phishing-09](#) (work in progress), August 2008.
- [MITM] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols", IACR ePrint Archive , October 2002.
- [RAD-EAP] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", [RFC 3579](#), September 2003.
- [RADIUS] Rigney, C., Willens, S., Rubens, A., and W. Simpson,

Nir, et al. Expires December 25, 2011 [Page 18]

Internet-Draft TLS with EAP Extension (TEE) June 2011

- "Remote Authentication Dial In User Service (RADIUS)", [RFC 2865](#), June 2000.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", [RFC 5247](#), August 2008.
- [RFC5433] Clancy, T. and H. Tschofenig, "EAP Generalized Pre-Shared Key (EAP-GPSK)", [RFC 5433](#), February 2009.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol: IKEv2", [RFC 5996](#), September 2010.
- [TLS-PSK] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", [RFC 4279](#), December 2005.
- [TLS/IA] Funk, P., Blake-Wilson, S., Smith, H., Tschofenig, N., and T. Hardjono, "TLS Inner Application Extension (TLS/IA)", [draft-funk-tls-inner-application-extension-03](#) (work in progress), June 2006.

Nir, et al.

Expires December 25, 2011

[Page 19]

Internet-Draft

TLS with EAP Extension (TEE)

June 2011

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 67897
Israel

Email: ynir@checkpoint.com

Yaron Sheffer
Independent

Email: aronf.ietf@gmail.com

Hannes Tschofenig
Nokia Siemens Networks

Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Peter Gutmann
University of Auckland
Department of Computer Science
New Zealand

Email: pgut001@cs.auckland.ac.nz