Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 2 September 2023

Chen, Ed.
L. Su
China Mobile
H. Wang
Huawei International Pte. Ltd.
1 March 2023

Use Identity as Raw Public Key in EAP-TLS
draft-chen-emu-eap-tls-ibs-05

Abstract

This document specifies the use of identity as a raw public key in EAP-TLS, the procedure of EAP-TIBS is consistent with EAP-TLS's interactive process, identity-based signature is extended to support EAP-TLS's signature algorithms to avoid X.509 certificates, this authentication method can avoid the overhead of receiving and processing certificate chains.

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 2023.

Copyright Notice

Table of Contents

1. Introduction

The Extensible Authentication Protocol(EAP) defined in [RFC3748] can provide support for multiple authentication methods. Transport Layer Security(TLS) provides for mutual authentication, integrity-protected ciphersuite negotiation, and exchange between two endpoints. The EAP-TLS defined in [RFC5216] which combines EAP and TLS that apply EAP method to load TLS procedures.

Traditionally, TLS client and server public keys are obtained in PKIX containers in-band as part of the TLS handshake procedure and are validated using trust anchors based on a PKIX certification authority (CA). But there is another method, Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) are defined in [RFC7250], the document defines two TLS extensions client_certificate_type and server_certificate_type, which can be used as part of an extended TLS handshake when raw public keys are used. [RFC9190] reads certificates can be of any type supported by TLS including raw public keys. In [RFC7250] it assuming that an out-of-band mechanism is used to bind the public key to the entity presenting the key.

Digital signatures provide the functions of Sender reliability and Message integrity. A chain of trust for such signatures is usually provided by certificates, but in low-bandwidth and resource-constrained environments, the use of certificates might be undesirable. In comparison with the original certificate, the raw public key is fairly small. This document describes a signature algorithm using identity as a raw public key in EAP-TLS, instead of transmitting a full certificate in the EAP-TLS message, only public keys are exchanged between client and server, also known as EAP-TIBS.

With the existing raw public key scheme, a public key and identity mapping table is required at server. This table usually established with offline method and may require additional efforts for establishment and maintenance, especially when the number of devices are huge. On the other hand, with IBS signature algorithm, it not only can take the advantage of raw public key, but also eliminates the efforts for the mapping table establishment and maintenance at the server side. Instead, a small table for CRL is enough for exclude revoked identity from accessing the network. A number of IBE and IBS algorithms have been standardized, such as ECCSI defined in [RFC6507].

IBC was first proposed by Adi Shamir in 1984. For an IBC system, a Key Management System (KMS) is required to generate keys for devices. The KMS choose its KMS Secret Authentication Key(KSAK) as the root of trust. A public parameter, KMS Public Authentication Key (KPAK) is derived from this secrete key and is used by others in verifying the signature. The signatures are generated by an entity with private keys obtained from the KMS. KMS is a trusted third party, users or devices can obtain private key using their identities from KMS. In IBS the private key is also known as Secret Signing Key(SSK). A sender can sign a message using SSK. The receiver can verify the signature with sender’s identity and the KPAK.

This method has great advantages in internal management.

2. Terminology

The following terms are used:

* IBC: Identity-Based Cryptograph, it is an asymmetric public key cryptosystem.

* IBS: Identity-based Signature, such as ECCSI.

* PKI: Public Key Infrastructure, an infrastructure built with a public-key mechanism.

* Authenticator: The entity initiating EAP authentication.

* Peer: The entity that responds to the authenticator.

* Backend authenticator server: A backend authentication server is an entity that provides an authentication service to an authenticator. When used, this server typically executes EAP methods for the authenticator.

* 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.

3. EAP TIBS Use Cases

3.1. IoT use case

Used for authentication of Internet of Things devices: due to the limited processing power of IoT devices, resources for secure identity authentication are limited, especially passive, long life cycle devices, however, the traditional certificate authentication based on PKI X509, because of the complexity of certificate processing and certificate chain authentication, not very suitable for the Internet of Things scenario. Internet of Things devices really need a more lightweight authentication method, and EAP-TIBS as one of the candidates.

3.2. Non CA use case

Used for systems that do not support CA certificates: an internal system with network security boundaries that can self-operate the Key Management System(KMS) secret key distribution center, EAP-TIBS can be used between internal subsystems.

4. Structure of the Raw Public Key Extension

To support the negotiation of using raw public key between client and server, a new certificate structure is defined in [RFC7250]. It is used by the client and server in the hello messages to indicate the types of certificates supported by each side. When RawPublicKey type is selected for authentication, SubjectPublicKeyInfo which is a data structure is used to carry the raw public key and its cryptographic algorithm.

The SubjectPublicKeyInfo structure is defined in Section 4.1 of [RFC5280] and not only contains the raw keys, such as the public exponent and the modulus of an RSA public key, but also an algorithm identifier. The algorithm identifier can also include parameters. The structure of SubjectPublicKeyInfo is shown in Figure 1:

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
          algorithm               AlgorithmIdentifier,
          subjectPublicKey        BIT STRING  }

     AlgorithmIdentifier   ::=  SEQUENCE  {
          algorithm               OBJECT IDENTIFIER,
          parameters              ANY DEFINED BY algorithm OPTIONAL }
```

Figure 1: SubjectPublicKeyInfo ASN.1 Structure

The algorithms identifiers are Object Identifier(OIDs), AlgorithmIdentifier is also data structure with two fields, OID represent the cryptographic algorithm used with raw public key, such as ECCSI, parameters are the necessary parameters associated with the algorithm.

In the case of IBS algorithm, the User's identity is the raw public key which can be represented by "subjectPublicKey", when ECCSI is used as the Identity-based signature algorithm, then "algorithm" is for ECCSI, and "parameters" is the parameters needed in ECCSI.

So far, IBS has the following four algorithms, the following table is the corresponding table of Key type and OID.

Key Type	Document	OID
ISO/IEC 14888-3 IBS-1	ISO/IEC 14888-3: IBS-1 mechanism	1.0.14888.3.0.7
ISO/IEC 14888-3 IBS-2	ISO/IEC 14888-3: IBS-2 mechanism	1.0.14888.3.0.8
ISO/IEC 14888-3 ChineseIBS(SM9)	ISO/IEC 14888-3: ChineseIBS mechanism	1.2.156.10197.1.302.1
Elliptic Curve-Based Signatureless For Identitiy-based Encryption (ECCSI)	Section 5.2 in RFC 6507	1.3.6.1.5.5.7.6.29

Table 1: Algorithm Object Identifiers

In the draft draft-wang-tls-raw-public-key-with-ibc, there extend signature scheme with IBS algorithm which indicated in the client's "signature_algorithms" extension. The SignatureScheme data structure also keep pace with the section 4.

5. EAP-TIBS for TLS1.2

5.1. EAP-TIBS Handshake

This section describes EAP-TIBS in the case of TLS1.2, as described in [RFC7250], the document intrudoces the use of raw public keys in TLS/DTLS, the basic raw public key TLS exchange will appear as follows, Figure 2 shows the client_certificate_type and server_certificate_type extensions added to the client and server hello messages. An extension type MUST NOT appear in the ServerHello unless the same extension type appeared in the corresponding ClientHello, defined in [RFC5246].

The server_certificate_type extension in the client hello indicates the types of certificates the client is able to process when provided by the server in a subsequent certificate payload.

The client_certificate_type and server_certificate_type extensions sent in the client hello each carry a list of supported certificate types, sorted by client preference. When the client supports only one certificate type, it is a list containing a single element. Many types of certificates can be used, such as RawPublicKey, X.509 and OpenPGP.

This section describes EAP-TLS extend using raw public keys, the procedures is as follows, In the discussion, we will use the term "EAP server" to denote the ultimate endpoint conversing with the peer.

Figure 2: EAP-TIBS authentication procedure for TLS1.2

5.2. EAP-TIBS example

In this example, both the TLS client and server use ECCSI for authentication, and they are restricted in that they can only process ECCSI signature algorithm. As a result, the TLS client sets both the server_certificate_type and the client_certificate_type extensions to be raw public key; in addition, the client sets the signature algorithm in the client hello message to be eccsi_sha256.

```
Authenticating Peer                EAP server
   -------------------              -------------
                                     <- EAP-Request/
                                     Identity
   EAP-Response/
   Identity (MyID) ->
                                     <- EAP-Request/
                                     EAP-Type=EAP-TLS
                                     (TLS Start)
   EAP-Response/
   EAP-Type=EAP-TLS
   (TLS client_hello
   signature_algorithm = (eccsi_sha256)
    server_certificate_type = (RawPublicKey,...)
    client_certificate_type = (RawPublicKey,...))->

                                    <- EAP-Request/
                                    EAP-Type=EAP-TLS
                                    (TLS server_hello,
                                    {client_certificate_type = RawPublicKey}
                                    {server_certificate_type = RawPublicKey}
                                    {certificate = (1.3.6.1.5.5.7.6.29, hash
                                    value of ECCSIPublicParameters),
                                    serverID)}
                                    {certificate_request = (eccsi_sha256)}
                                    {server_hello_done}
                             )
   EAP-Response/
   EAP-Type=EAP-TLS
   ({certificate = ((1.3.6.1.5.5.7.6.29,
   hash value of ECCSIPublicParameters),
   ClientID)},
    {certificate_verify = (ECCSI-Sig-Value)},
    {finished}) ->
                                <- EAP-Request/
                                EAP-Type=EAP-TLS
                                (TLS finished)
   EAP-Response/
   EAP-Type=EAP-TLS ->
                                <- EAP-Success
```

Figure 3: EAP-TIBS example

6. EAP-TIBS for TLS1.3

6.1. EAP-TIBS Handshake

TLS1.3 defined in [RFC8446], as TLS 1.3 is not directly compatible with previous versions, all versions of TLS incorporate a versioning mechanism which allows clients and servers to interoperably negotiate a common version if one is supported by both peers. when make the discussion on EAP-TLS using raw public keys we also make a different with TLS1.2, this section is for EAP-TLS1.3 handshake using raw public keys and give example for EAP-TIBS.

This section describes EAP-TLS1.3 extend using raw public keys, the procedures is as follows, both client and server have the extension "key_share", the "key_share" extension contains the endpoint's cryptographic parameters. the "signature_algorithm" extension contains the signature algorithm and hash algorithms the client and server can support for the new signature algorithms specific to the IBS algorithms. When IBS is chosen as signature algorithm, the server need to indicated the required IBS signature algorithms int the signature_algorithm extension within the CertificateRequest.

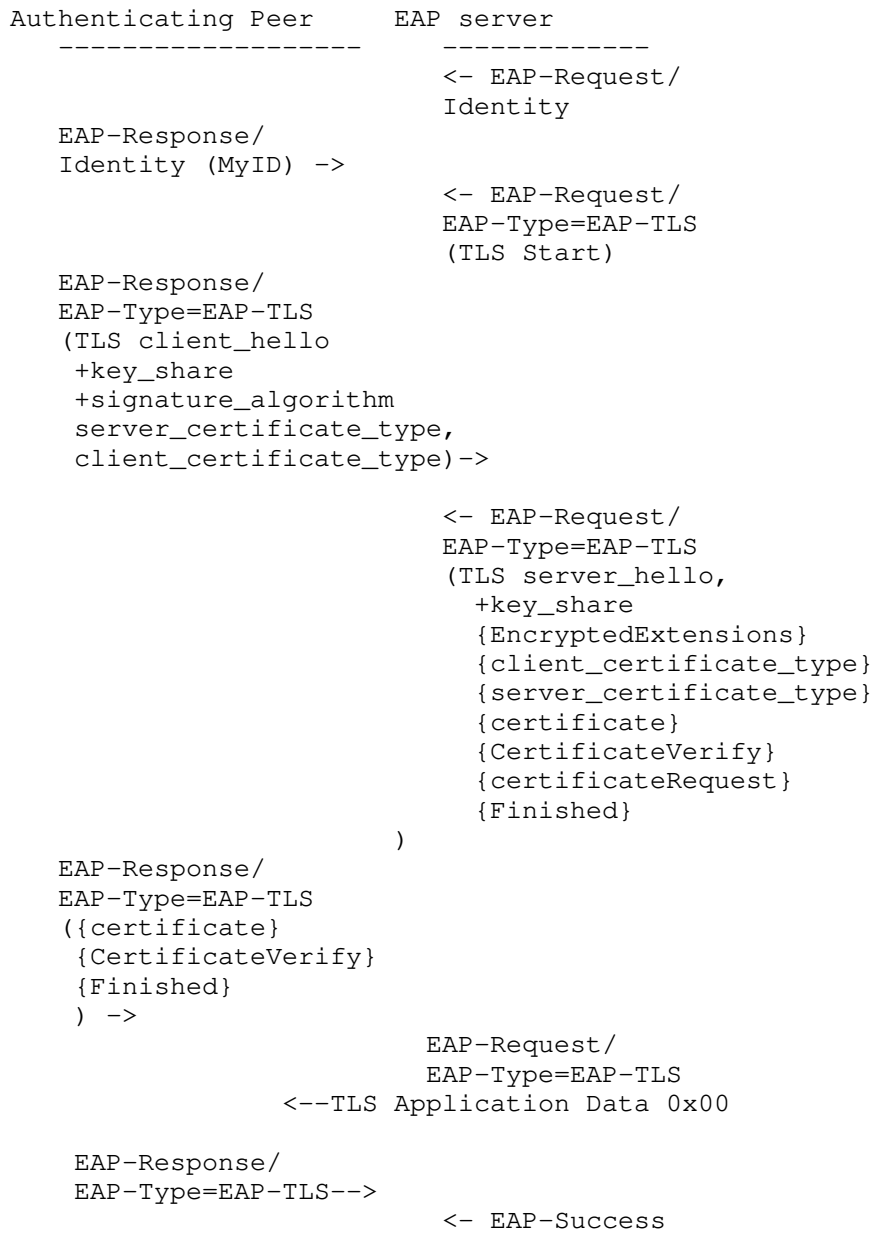


Figure 4: EAP-TIBS authentication procedure for TLS1.3

6.2. EAP-TIBS example

When the EAP server receives the client hello, it processes the message. Since it has an ECCSI raw public key from the KMS, it indicates that it agrees to use ECCSI and provides an ECCSI key by placing the SubjectPublicKeyInfo structure into the Certificate payload back to the client, including the OID, the identity of server, ServerID, which is the public key of server also, and hash value of KMS public parameters. The client_certificate_type indicates that the TLS server accepts raw public key. The TLS server demands client authentication, and therefore includes a certificate_request, which requires the client to use eccsi_sha256 for signature. A signature value based on the eccsi_sha256 algorithm is carried in the CertificateVerify. The client, which has an ECCSI key, returns its ECCSI public key in the Certificate payload to the server, which includes an OID for the ECCSI signature. The example of EAP-TLS1.3-IBS is as follows:

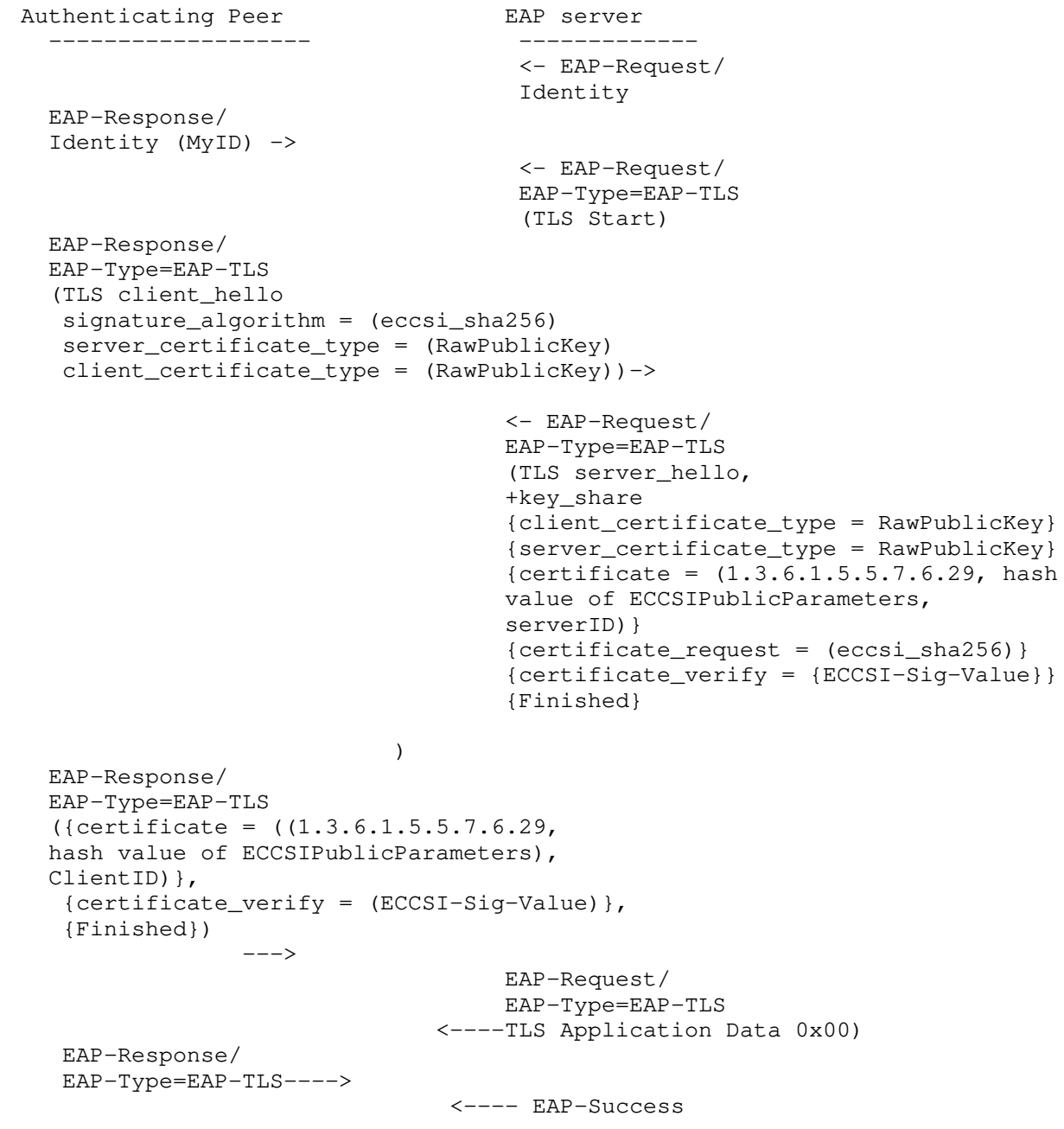


Figure 5: EAP-TLS1.3-IBS example

7. IANA Considerations

This document registers the following item in the "Method Types" registry under the "extensible Authentication Protocol(EAP) Registry" heading.

Value	Description
TBD	EAP-TIBS

8. Security Considerations

Although the identity authentication has been extended, the generation of session key still continues the EAP-TLS method.

9. Informative References

[RFC6507] Groves, M., "Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)", RFC 6507, DOI 10.17487/RFC6507, February 2012, <https://www.rfc-editor.org/rfc/rfc6507>.

[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/rfc/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/rfc/rfc5216>.

[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/rfc/rfc7250>.

[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/rfc/rfc5280>.

[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/rfc/rfc5246>.

[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/rfc/rfc9190>.

[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/rfc/rfc8446>.

Authors' Addresses

Meiling Chen (editor)
China Mobile
BeiJing
China
Email: chenmeiling@chinamobile.com

Li Su
China Mobile
BeiJing
China
Email: suli@chinamobile.com

Haiguang Wang
Huawei International Pte. Ltd.
Singapore
Email: wang.haiguang1@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 27 November 2022

O. Friel
Cisco
D. Harkins
Hewlett-Packard Enterprise

26 May 2022

Bootstrapped TLS Authentication
draft-friel-tls-eap-dpp-05

Abstract

This document defines a TLS extension that enables a server to prove to a client that it has knowledge of the public key of a key pair where the client has knowledge of the private key of the key pair. Unlike standard TLS key exchanges, the public key is never exchanged in TLS protocol messages. Proof of knowledge of the public key is used by the client to bootstrap trust in the server. The use case outlined in this document is to establish trust in an EAP 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 27 November 2022.

Copyright Notice

Table of Contents

1. Introduction

On-boarding of devices with no, or limited, user interface can be difficult. Typically, a credential is needed to access the network and network connectivity is needed to obtain a credential. This poses a catch-22.

If trust in the integrity of a device's public key can be obtained in an out-of-band fashion, a device can be authenticated and provisioned with a usable credential for network access. While this authentication can be strong, the device's authentication of the network is somewhat weaker. [duckling] presents a functional security model to address this asymmetry.

There are on-boarding protocols, such as [DPP], to address this use case but they have drawbacks. [DPP] for instance does not support wired network access. This document describes an on-boarding protocol, which we refer to as TLS Proof of Knowledge or TLS-POK.

1.1. Terminology

The following terminology is used throughout this document.

* BSK: Bootstrap Key which is an elliptic curve public private key pair.

* DPP: Device Provisioning Protocol

* EPSK: External Pre-Shared Key

* PSK: Pre-Shared Key

1.2. Bootstrap Key Pair

The mechanism for on-boarding of devices defined in this document relies on bootstrap key pairs. A client device has an associated elliptic curve (EC) bootstrap key pair (BSK). The BSK may be static and baked into device firmware at manufacturing time, or may be dynamic and generated at on-boarding time by the device. If the BSK public key, specifically the ASN.1 SEQUENCE SubjectPublicKeyInfo from [RFC5280], can be shared in a trustworthy manner with a TLS server, a form of "origin entity authentication" (the step from which all subsequent authentication proceeds) can be obtained.

The exact mechanism by which the server gains knowledge of the BSK public key is out of scope of this specification, but possible mechanisms include scanning a QR code to obtain a base64 encoding of the ASN.1-formatted public key or uploading of a Bill of Materials (BOM) which includes the public key. If the QR code is physically attached to the client device, or the BOM is associated with the device, the assumption is that the public key obtained in this bootstrapping method belongs to the client. In this model, physical possession of the device implies legitimate ownership.

The server may have knowledge of multiple BSK public keys corresponding to multiple devices, and existing TLS mechanisms are leveraged that enable the server to identity a specific bootstrap public key corresponding to a specific device.

Using the process defined herein, the client proves to the server that it has possession of the private analog to its public bootstrapping key. Provided that the mechanism in which the server obtained the BSK public key is trustworthy, a commensurate amount of authenticity of the resulting connection can be obtained. The server also proves that it knows the client's public key which, if the client does not gratuitously expose its public key, can be used to obtain a modicum of correctness, that the client is connecting to the correct network (see [duckling]).

1.3. Alignment with Wi-Fi Alliance Device Provisioning Profile

The definition of the BSK public key aligns with that given in [DPP]. This, for example, enables the QR code format as defined in [DPP] to be reused for TLS-POK. Therefore, a device that supports both wired LAN and Wi-Fi LAN connections can have a single QR code printed on its label, and the bootstrap key can be used for DPP if the device bootstraps against a Wi-Fi network, or TLS-POK if the device bootstraps against a wired network. Similarly, a common bootstrap public key format could be imported in a BOM into a server that handles devices connecting over both wired and Wi-Fi networks.

Any bootstrapping method defined for, or used by, [DPP] is compatible with TLS-POK.

2. Bootstrapping in TLS 1.3

Bootstrapping in TLS 1.3 leverages Certificate-Based Authentication with an External Pre-Shared Key [RFC8773]. The External PSK (EPSK) is derived from the BSK public key, and the EPSK is imported using [I-D.ietf-tls-external-psk-importer]. This BSK MUST be from a cryptosystem suitable for doing ECDSA.

The TLS PSK handshake gives the client proof that the server knows the BSK public key. Certificate based authentication of the client by the server is carried out using the BSK, giving the server proof that the client knows the BSK private key. This satisfies the proof of ownership requirements outlined in Section 1.

2.1. External PSK Derivation

An [I-D.ietf-tls-external-psk-importer] EPSK is made of of the tuple of (Base Key, External Identity, Hash). The EPSK is derived from the BSK public key using [RFC5869] with the hash algorithm from the ciphersuite:

```
epsk   = HKDF-Expand(HKDF-Extract(<>, bskey),
                     "tls13-imported-bsk", L)
epskid = HKDF-Expand(HKDF-Extract(<>, bskey),
                     "tls13-bspsk-identity", L)
```

where:

- epsk is the EPSK Base Key
- epskid is the EPSK External Identity
- <> is a NULL salt
- bskey is the DER-encoded ASN.1 subjectPublicKeyInfo representation of the BSK public key
- L is the length of the digest of the underlying hash algorithm

The [I-D.ietf-tls-external-psk-importer] ImportedIdentity structure is defined as:

```
struct {
   opaque external_identity<1...2^16-1>;
   opaque context<0..2^16-1>;
   uint16 target_protocol;
   uint16 target_kdf;
} ImportedIdentity;
```

and is created using the following values:

```
external_identity = epskid
context = "tls13-bsk"
target_protocol = TLS1.3(0x0304)
target_kdf = HKDF_SHA256(0x0001)
```

The EPSK and ImportedIdentity are used in the TLS handshake as specified in [I-D.ietf-tls-external-psk-importer].

A performance versus storage tradeoff a server can choose is to precompute the identity of every bootstrapped key with every hash algorithm that it uses in TLS and use that to quickly lookup the bootstrap key and generate the PSK. Servers that choose not to employ this optimization will have to do a runtime check with every bootstrap key it holds against the identity the client provides.

2.2. Changes to TLS 1.3 Handshake

The client includes the "tls_cert_with_extern_psk" extension in the ClientHello, per [RFC8773]. The client identifies the BSK by inserting the serialized content of ImportedIdentity into the PskIdentity.identity in the PSK extension, per [I-D.ietf-tls-external-psk-importer]. The server looks up the client's EPSK key in its database using the mechanisms documented in

[I-D.ietf-tls-external-psk-importer]. If no match is found, the server SHALL terminate the TLS handshake with an alert.

If the server found the matching BSK, it includes the "tls_cert_with_extern_psk" extension in the ServerHello message, and the corresponding EPSK identity in the "pre_shared_key" extension. When these extensions have been successfully negotiated, the TLS 1.3 key schedule SHALL include both the EPSK in the Early Secret derivation and an (EC)DHE shared secret value in the Handshake Secret derivation.

After successful negotiation of these extensions, the full TLS 1.3 handshake is performed with the additional caveat that the client authenticates with a raw public key (its BSK) per [RFC7250]. The BSK is always an elliptic curve key pair, therefore the ClientCertTypeExtension SHALL always indicate RawPublicKey and the type of the client's Certificate SHALL be ECDSA and contain the client's BSK public key as a DER-encoded ASN.1 subjectPublicKeyInfo SEQUENCE.

When the server processes the client's Certificate it MUST ensure that it is identical to the BSK public key that it used to generate the EPSK and ImportedIdentity for this handshake.

When clients use the [duckling] form of authentication, they MAY forgo the checking of the server's certificate in the CertificateVerify and rely on the integrity of the bootstrapping method employed to distribute its key in order to validate trust in the authenticated TLS connection.

The handshake is shown in Figure 1.

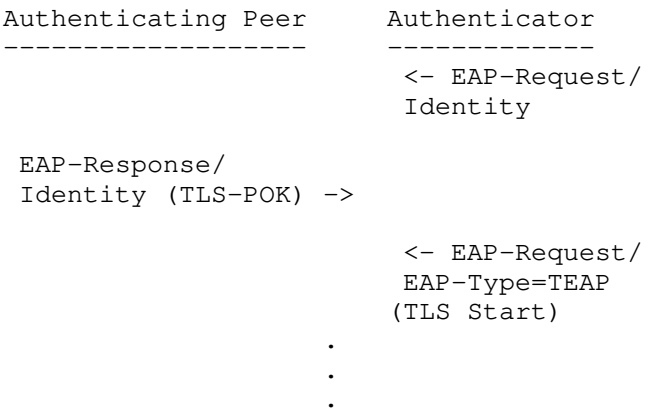
Figure 1: TLS 1.3 TLS-POK Handshake

3. Using TLS Bootstrapping in EAP

Enterprise deployments typically require an 802.1X/EAP-based authentication to obtain network access. Protocols like [RFC7030] can be used to enroll devices into a Certification Authority to allow them to authenticate using 802.1X/EAP. But this creates a Catch-22 where a certificate is needed for network access and network access is needed to obtain certificate.

Devices whose bootstrapping key can been obtained in an out-of-band fashion can perform an EAP-TLS-based exchange, for instance [RFC7170], and authenticate the TLS exchange using the bootstrapping extensions defined in Section 2. This network connectivity can then be used to perform an enrollment protocol (such as provided by [RFC7170]) to obtain a credential for subsequent network connectivity and certificate lifecycle maintenance.

Upon "link up", an Authenticator on an 802.1X-protected port will issue an EAP Identify request to the newly connected peer. For unprovisioned devices that desire to take advantage of TLS-POK, there is no initial realm in which to construct an NAI (see [RFC4282]) so the initial EAP Identity response SHOULD contain simply the name "TLS-POK" in order to indicate to the Authenticator that an EAP method that supports TLS-POK SHOULD be started.



4. Summary of Work

The protocol outlined here can be broadly broken up into 4 distinct areas:

* TLS extensions to transport the bootstrap public key identifier

* Use of the TLS 1.3 extension for certificate-based authentication with an external PSK

* The client's use of a raw public key in its certificate

* TEAP extensions to leverage the new TLS-POK handshake for trust establishment

This document captures all 4 areas.

5. IANA Considerations

None.

6. Security Considerations

Bootstrap and trust establishment by the TLS server is based on proof of knowledge of the client's bootstrap public key, a non-public datum. The TLS server obtains proof that the client knows its bootstrap public key and, in addition, also possesses its corresponding private analog.

Trust on the part of the client is based on validation of the server certificate and the TLS 1.3 handshake. In addition, the client assumes that knowledge of its public bootstrapping key is not widely disseminated and therefore any device that proves knowledge of its

bootstrapping key is the appropriate device from which to receive provisioning, for instance via [RFC7170]. [duckling] describes a security model for this type of "imprinting".

An attack on the bootstrapping method which substitutes the public key of a corrupted device for the public key of an honest device can result in the TLS sever on-boarding and trusting the corrupted device.

If an adversary has knowledge of the bootstrap public key, the adversary may be able to make the client bootstrap against the adversary's network. For example, if an adversary intercepts and scans QR labels on clients, and the adversary can force the client to connect to its server, then the adversary can complete the TLS-POK handshake with the client and the client will connect to the adversary's server. Since physical possession implies ownership, there is nothing to prevent a stolen device from being on-boarded.

7. References

7.1. Normative References

[I-D.ietf-tls-external-psk-importer]
Benjamin, D. and C. A. Wood, "Importing External PSKs for TLS", Work in Progress, Internet-Draft, draft-ietf-tls-external-psk-importer-08, 22 April 2022, <https://www.ietf.org/archive/id/draft-ietf-tls-external-psk-importer-08.txt>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[RFC8773] Housley, R., "TLS 1.3 Extension for Certificate-Based Authentication with an External Pre-Shared Key", RFC 8773, DOI 10.17487/RFC8773, March 2020, <https://www.rfc-editor.org/info/rfc8773>.

7.2. Informative References

[DPP] Wi-Fi Alliance, "Device Provisioning Profile", 2020.

[duckling] Stajano, F. and E. Rescorla, "The Ressurecting Ducking: Security Issues for Ad-Hoc Wireless Networks", 1999.

[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>.

[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>.

Authors' Addresses

Owen Friel
Cisco
Email: ofriel@cisco.com

Dan Harkins
Hewlett-Packard Enterprise
Email: daniel.harkins@hpe.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 7, 2022

T. Aura
Aalto University
M. Sethi
Ericsson
A. Peltonen
Aalto University
September 3, 2021

Nimble out-of-band authentication for EAP (EAP-NOOB) draft-ietf-emu-eap-noob-06

Abstract

The 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. The EAP method is intended for bootstrapping all kinds of Internet-of-Things (IoT) devices that have no pre-configured authentication credentials. The method makes use of a user-assisted one-directional OOB message between the peer device and authentication server to authenticate the in-band key exchange. The device must have a non-network input or output interface, such as a display, microphone, speaker, or blinking light, which can send or receive dynamically generated messages of tens of bytes in length.

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 7, 2022.

Table of Contents

1. Introduction

This document describes a method for registration, authentication and key derivation for network-connected smart 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, or they may be recycled devices after a hard reset. 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 user 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.

To summarize, devices may have the following characteristics:

- no pre-established relation with the intended server or user,

- no pre-provisioned device identifier or authentication credentials,
- input or output interface that may be capable of only one-directional out-of-band communication.

Many proprietary out-of-band (OOB) configuration methods exist 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 OOB channel. The device authentication relies on a user having physical access to the device, and 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 attacks.

The solution presented here is intended for devices that have either a non-network 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 pairing [BluetoothPairing], where it is essential to minimize the length of the manually transferred or compared codes. The OOB messages in this specification are dynamically generated. We thus do not support static printed registration codes. One reason for requiring dynamic OOB messages is that the receipt of the OOB message authorizes the server to take ownership of the device. Dynamic OOB messages are more secure than static printed codes, which could be leaked and later misused.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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. (We use the terms peer, device, and peer device interchangeably.)

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 two or more EAP conversations, called Exchanges in this specification. Each Exchange consists of several EAP request-response pairs. At least two separate EAP conversations are needed to give the human user time to deliver the OOB message between them.

The overall protocol starts with the Initial Exchange, which comprises four EAP request-response pairs. In the Initial Exchange, 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.

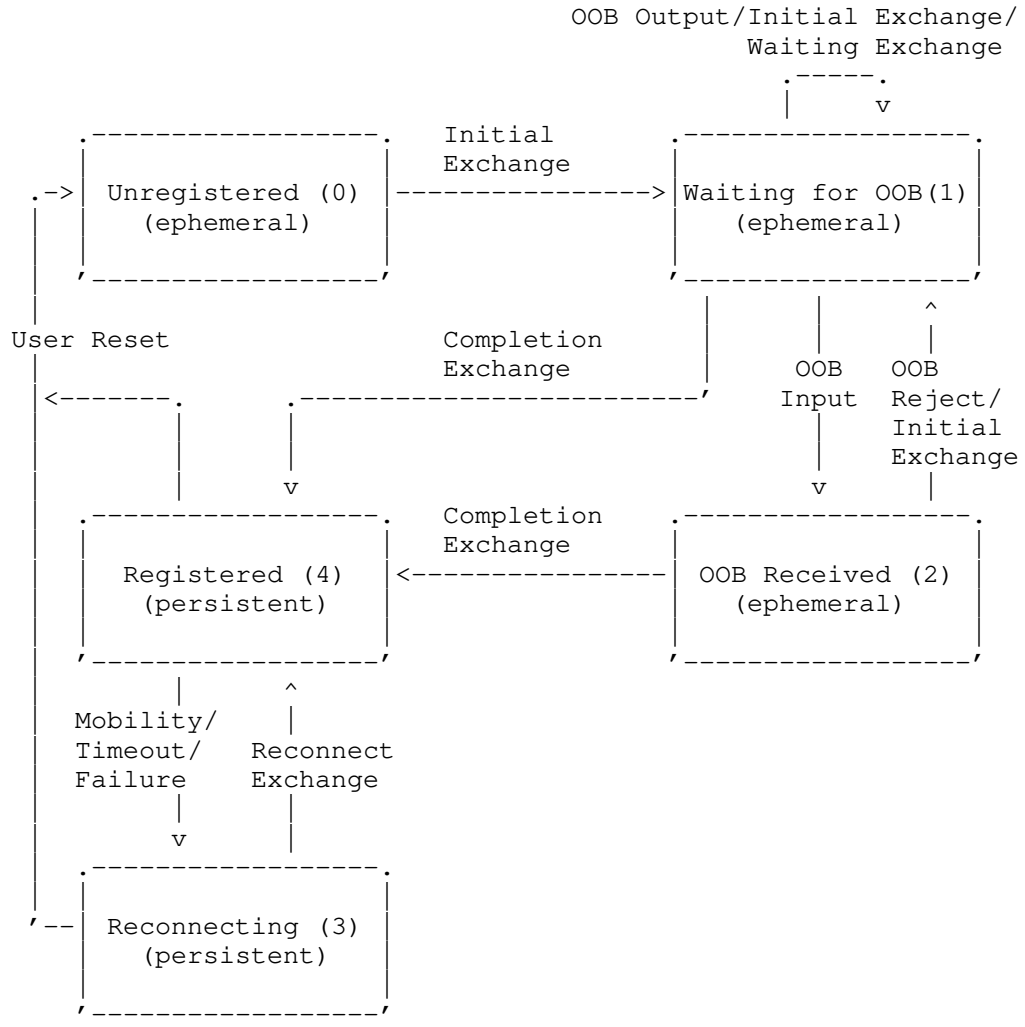


Figure 1: EAP-NOOB server-peer association state machine

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 a timeout or error may cause one or both endpoints to go back to the Unregistered state so that the Initial Exchange is repeated. 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 the ephemeral 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 MUST 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 also resets 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.arpa" 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 an Authentication, Authorization and Accounting (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. Moreover, EAP-NOOB supports peers with no input capability in the user interface (e.g., LED light bulbs). Since users cannot initiate the protocol in these devices, the devices 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. 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 fields present in each message. Each exchange comprises multiple EAP request-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-NOOB exchanges

All EAP-NOOB exchanges start with common handshake messages. The handshake begins 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 exchange begins 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=1) 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=1) 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 MUST determine the exchange type based on the combination of the peer and server states as follows (also summarized in Figure 11). If either the peer or 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 SHOULD indicate 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 the Registered (4) state.

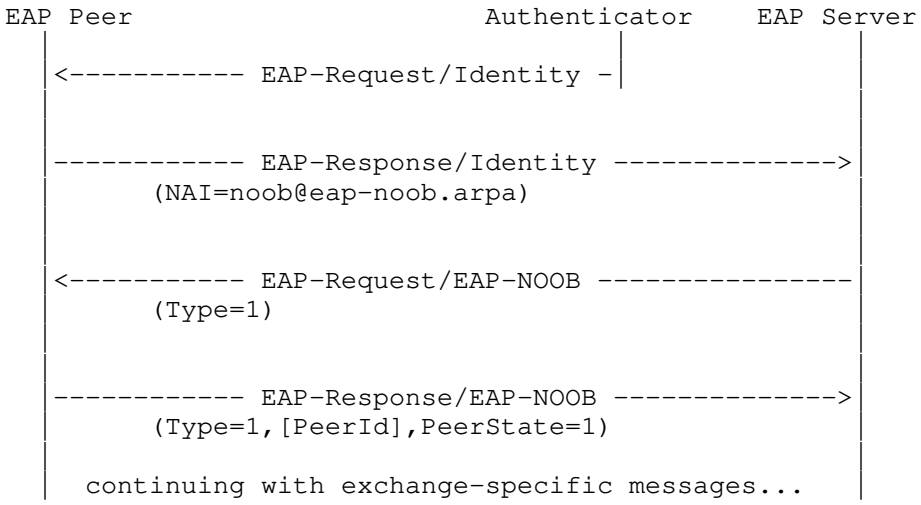


Figure 2: Common handshake in all EAP-NOOB exchanges

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=2) from the server contains a newly allocated PeerId for the peer and an optional NewNAI for assigning a new NAI to the peer. The server allocates a new PeerId in the Initial Exchange regardless of any old PeerId received in the previous response (Type=1). 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=2) with the selected protocol version (Verp), the received PeerId, the selected cryptosuite (Cryptosuitep), an indicator of the OOB channel direction(s) selected by the peer (Dirp), and a PeerInfo object. In the second EAP-NOOB request and response (Type=3), 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.

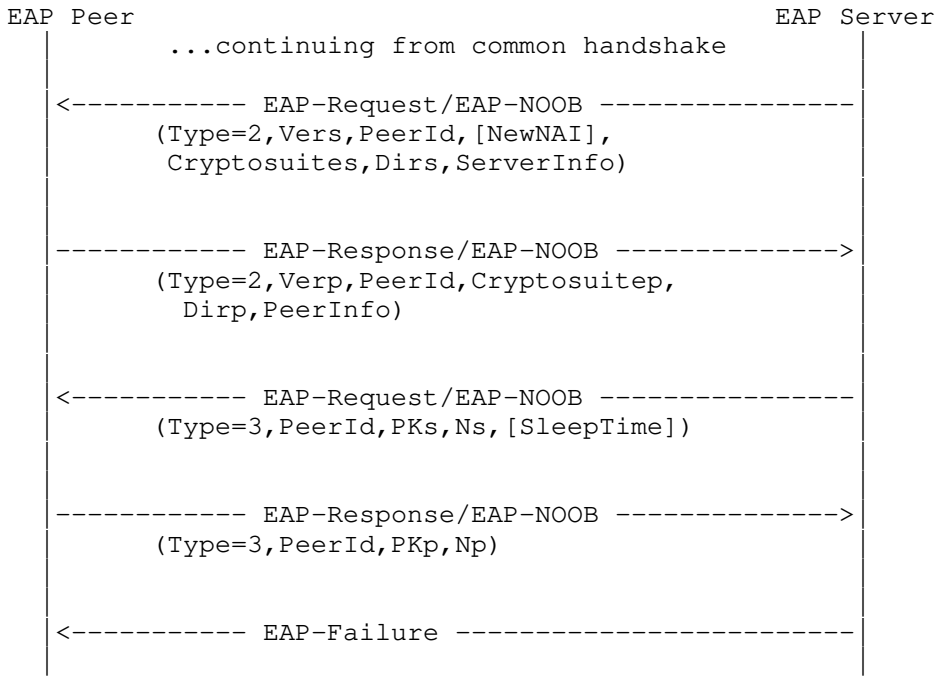


Figure 3: Initial Exchange

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 as 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.

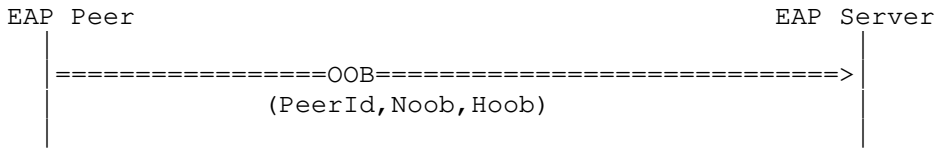


Figure 4: OOB Step, from peer to EAP server

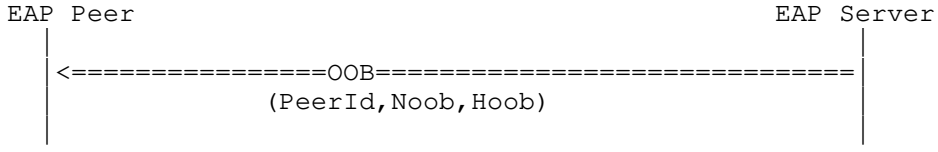


Figure 5: OOB Step, from EAP server to peer

The OOB receiver MUST compare the received value of the fingerprint Hoob (see Section 3.3.2) with a value that it computed locally for the PeerID received. This integrity check ensures that the endpoints agree on contents of the Initial Exchange. 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 in the association state machine, 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 an 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 instead 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 NoobInterval = NoobTimeout / 2, in which case the receiver of the OOB will at any given time accept either of the two latest Noob values. 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 at least one of them.

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

3.2.4. Completion Exchange

After the Initial Exchange, if the OOB channel directions selected by the peer include the peer-to-server direction, 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. If the OOB channel directions selected by the peer include the server-to-peer direction 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 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=6) takes place. In the request, the server sends the NoobId value (see Section 3.3.2), 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 this case, two request-response pairs (Type=5 and Type=6) are needed. The purpose of the first request-response pair (Type=5) 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 request-response pair (Type=6) 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 called 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.

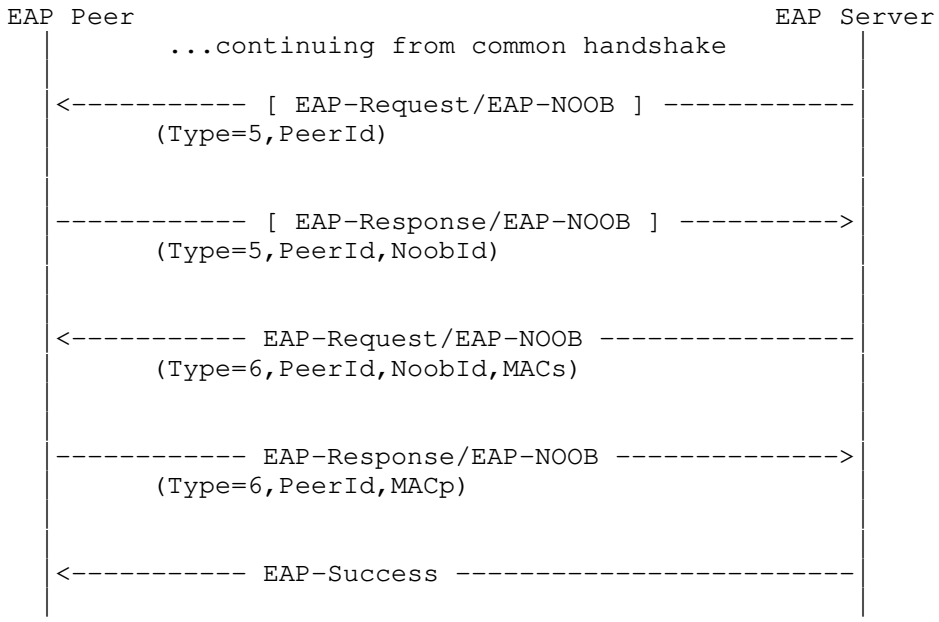


Figure 6: Completion Exchange

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 always ends 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 SHOULD 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 MUST 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 message authentication code. 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.

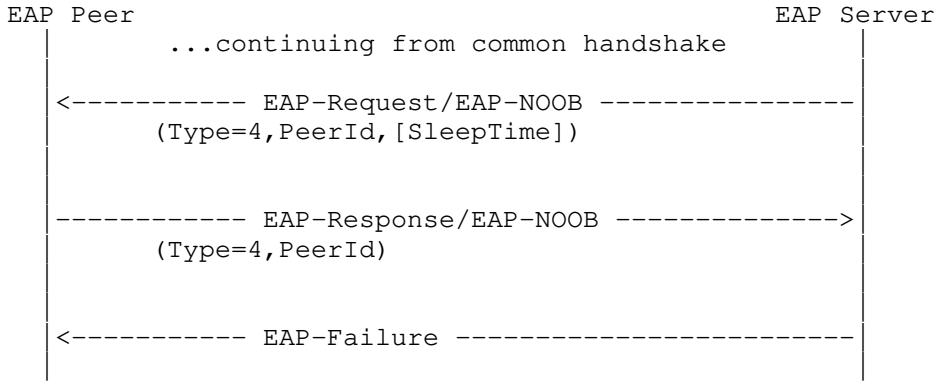


Figure 7: Waiting Exchange

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 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 ASCII string.

Another way to generate the identifiers is to choose a random 22-character alphanumeric ASCII 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. The peer MUST copy the PeerId byte-by-byte from the message where it was allocated, and the server MUST perform a byte-by-byte comparison between the received and the previously allocated PeerID. The peer sets the PeerId value in response type 1 as follows. As stated in Section 3.2.1, when the peer is in the Unregistered (0) state, it SHOULD omit the PeerId from response type 1. 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 NAI for the peer is "noob@eap-noob.arpa". The peer implementation MAY allow the user or application to configure a different NAI, which overrides the default NAI. Furthermore, the server MAY assign a new NAI to the peer in the Initial Exchange or Reconnect Exchange, in the NewNAI field of request types 2 and 7, to override any previous NAI value. 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 NewNAI in the latest Initial Exchange, the peer MUST use the configured NAI, or if it does not exist, the default NAI. When the peer is in one of the states 1..2 and the server sent a NewNAI in the latest Initial Exchange, the peer MUST use this server-assigned NAI. When the peer moves to the Registered (4) state after the Completion Exchange, it writes to the persistent EAP-NOOB association the same NAI value that it used in the Completion Exchange. When the peer is in the Reconnecting (3) or Registered (4) state, it MUST use the NAI from its persistent EAP-NOOB association. When the server sends NewNAI in the Reconnect Exchange, the peer writes its value to the persistent EAP-NOOB association when it moves from the Reconnecting (3) state to the Registered (4) state. All the NAI values MUST follow the syntax specified in [RFC7542].

The purpose of the server-assigned NAI is to enable more flexible routing of the EAP sessions over the AAA infrastructure, including roaming scenarios (see Appendix C). Moreover, some Authenticators or AAA servers use the realm part of the assigned NAI to determine peer-specific connection parameters, such as isolating the peer to a specific VLAN. The user or application configured NAI, on the other hand, 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 server-assigned NAI 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 NAI 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.
NAI, NewNAI	Peer NAI and server-assigned new peer NAI as defined in Section 3.3.1.
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.
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.
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]. The 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. Mandatory to implement cryptosuites and the registration procedure for new cryptosuites is specified in Section 5.1. Example values are "[1]" and "1", respectively.
Dirs, Dirp	An integer indicating 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) [RFC3986] 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 the peer MUST NOT start a new execution of the EAP-NOOB method with the authenticator, unless the peer receives the OOB message or the sending is triggered by an application-specific user action. 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]. The 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 5.3.
ErrorInfo	Textual error message for logging and debugging purposes. A 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 the 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 H, which is 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 function HMAC, which is the HMAC message authentication code [RFC2104] based on the cryptographic hash function H 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,Cryptos
uitep,Dirp,NAI,PeerInfo,0,PKs,Ns,PKp,Np,Noob).

NoobId = H("NoobId",Noob).

MACs = HMAC(Kms; 2,Vers,Verp,PeerId,Cryptosuites,Dirs,ServerInfo,C
ryptosuitep,Dirp,NAI,PeerInfo,0,PKs,Ns,PKp,Np,Noob).

MACp = HMAC(Kmp; 1,Vers,Verp,PeerId,Cryptosuites,Dirs,ServerInfo,C
ryptosuitep,Dirp,NAI,PeerInfo,0,PKs,Ns,PKp,Np,Noob).

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

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

The inputs denoted with "" above are not present, and the values in brackets [] are optional. Both kinds of missing input values are represented by empty strings "" in the HMAC input (JSON array). The NAI included in the inputs is the NAI value that will be in the persistent EAP-NOOB association if the Completion Exchange or Reconnect Exchange succeeds. In the Completion Exchange, the NAI is the NewNAI value assigned by the server in the preceding Initial Exchange, or if no NewNAI was sent, the NAI used by the client in the Initial Exchange. In the Reconnect Exchange, the NAI is the NewNAI value assigned by the server in the same Reconnect Exchange, or if no NewNAI was sent, the unchanged NAI from the persistent EAP-NOOB association. 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 as JSON strings 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 Main Session Key (MSK) and Extended Main Session Key (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 are no longer accepted by the other endpoint. Changes 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 the 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 ASCII characters)
Verp	Negotiated protocol version	integer
Cryptosuitep	Negotiated cryptosuite	integer
CryptosuitepPrev (at peer only)	Previous cryptosuite	integer
NAI	NAI assigned by server, configured by user, or the default NAI "noob@eap-noob.arpa"	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=7) 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 CryptosuitePrev from the persistent EAP-NOOB association are only used to support the negotiation as described above; all actual cryptographic operations use the newly negotiated cryptosuite. The request and response (Type=7) 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 value without changing the cryptosuite. In the second request and response (Type=8), 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=9), 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=9) 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=9) 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=9). 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=9) 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 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=9), 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 NewNAI, 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

NewNAI was sent, each side updates NAI in the persistent EAP-NOOB association when moving to the Registered (4) state.

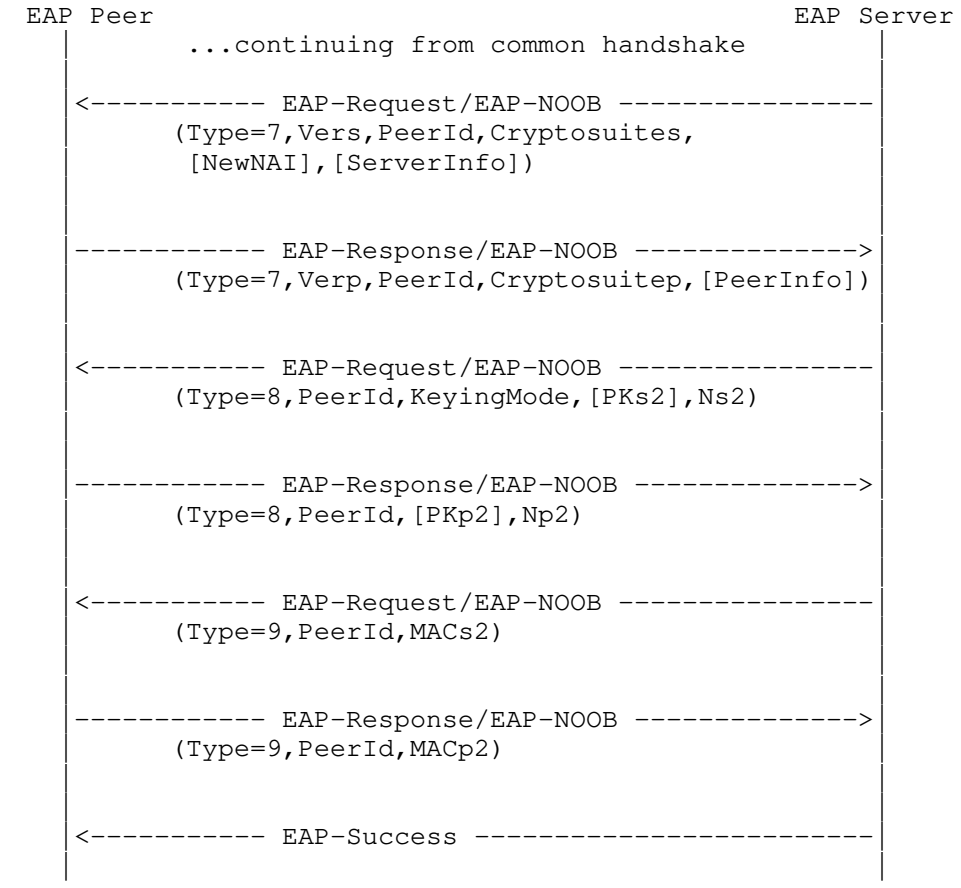


Figure 8: Reconnect Exchange

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 the 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(2e, 0s, 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 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 one-step key derivation function (KDF) defined in section 5.8.2.1 of the NIST specification. The auxiliary function H is a hash function, and it 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 four 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 (FixedInfo), for which we use the concatenation format defined in section 5.8.2.1.1 of the NIST specification [NIST-DH]. FixedInfo 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 a 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.

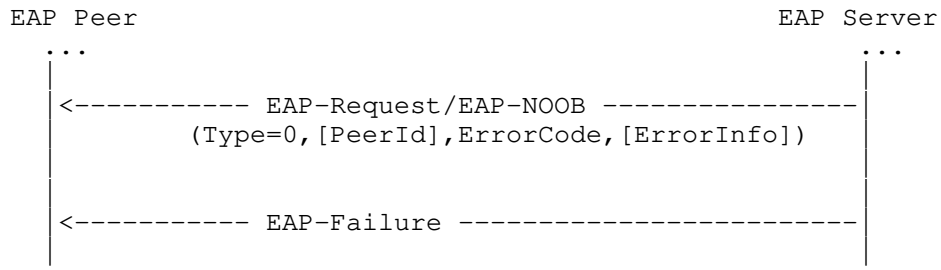


Figure 9: Error notification from server to peer

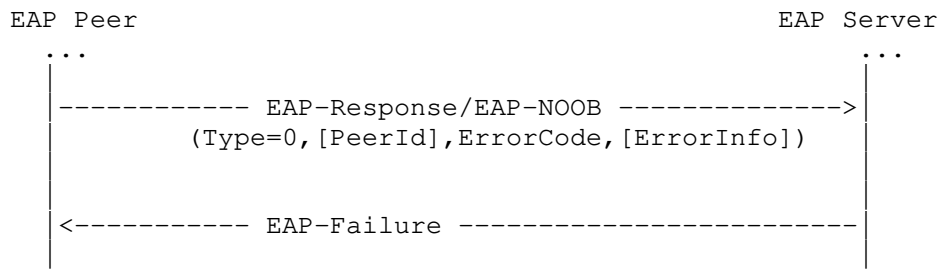


Figure 10: Error notification from peer to server

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 1005 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, peer devices that are unable to alert a user SHOULD continue to try to reconnect infrequently (e.g., approximately every 3600 seconds).

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 an 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 as 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 an invalid server URL.

4. ServerInfo and PeerInfo contents

The ServerInfo and PeerInfo fields in the Initial Exchange and Reconnect Exchange enable the server and peer, respectively, to 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 (see Section 7.7). This section describes the optional initial data fields for ServerInfo and PeerInfo registered by this specification. Further specifications may request new application-specific ServerInfo and PeerInfo data fields from IANA (see Section 5.4 and Section 5.5).

Data Field	Description
Type	Type-tag string that can be used by the peer as a hint for how to interpret the ServerInfo contents.
ServerName	String that may be used to aid human identification of the server.
ServerURL	Prefix string when the OOB message is formatted as a URL, as suggested in Appendix D.
SSIDList	List of IEEE 802.11 wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix C. JSON array of ASCII encoded SSID strings.
Base64SSIDList	List of IEEE 802.11 wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix C. JSON array of SSIDs, each of which is base64url encoded without padding. Peers SHOULD send at most one of the fields SSIDList and Base64SSIDList in PeerInfo, and the server SHOULD ignore SSIDList if Base64SSIDList is included.

Table 6: ServerInfo data fields

Data Field	Description
Type	Type-tag string that can be used by the server as a hint for how to interpret the PeerInfo contents.
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 be in upper or lower case and MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.
SSID	IEEE 802.11 network SSID for channel binding. The SSID is an ASCII string.
Base64SSID	IEEE 802.11 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] for channel binding. The string MAY be in upper or lower case and MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.

Table 7: PeerInfo data fields

5. 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 in the Method Types sub-registry of the Extensible Authentication Protocol (EAP) registry (requested value = 56).

This memo also requires IANA to create and maintain a new registry entitled "Nimble out-of-band authentication for EAP Parameters" in the Extensible Authentication Protocol (EAP) registry. IANA is also requested to create and maintain sub-registries defined in the following subsections.

5.1. Cryptosuites

IANA is requested to create and maintain a new sub-registry entitled "EAP-NOOB Cryptosuites" in the "Nimble out-of-band authentication for EAP Parameters" registry. 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 initial values for this registry are:

Cryptosuite	Algorithms
1	ECDHE curve Curve25519 [RFC7748], public-key format [RFC7517], hash function SHA-256 [RFC6234]. The JWK encoding of Curve25519 public key is defined in [RFC8037]. For clarity: the "crv" parameter is "X25519", the "kty" parameter is "OKP", and the public-key encoding contains only an x-coordinate.
2	ECDHE curve NIST P-256 [FIPS186-4], public-key format [RFC7517], hash function SHA-256 [RFC6234]. The JWK encoding of NIST P-256 public key is defined in [RFC7518]. For clarity: the "crv" parameter is "P-256", the "kty" parameter is "EC", and the public-key encoding has both an x and y coordinates as defined in Section 6.2.1 of [RFC7518].

Table 8: EAP-NOOB cryptosuites

EAP-NOOB implementations MUST support Cryptosuite 1. Support for Cryptosuite 2 is RECOMMENDED. An example of Cryptosuite 1 public-key encoded as a JWK object is given below (line breaks are for readability only).

"jwk":{"kty":"OKP","crv":"X25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrfx-FG-IK08"}

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

5.2. Message Types

IANA is requested to create and maintain a new sub-registry entitled "EAP-NOOB Message Types" in the "Nimble out-of-band authentication for EAP Parameters" registry. EAP-NOOB request and response pairs are identified by an integer Message Type. The initial values for this registry are:

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

Table 9: EAP-NOOB Message Types

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

5.3. Error codes

IANA is requested to create and maintain a new sub-registry entitled "EAP-NOOB Error codes" in the "Nimble out-of-band authentication for EAP Parameters" registry. Cryptosuites are identified by an integer. The initial values for this registry are:

Error code	Purpose
1001	Invalid NAI
1002	Invalid message structure
1003	Invalid data
1004	Unexpected message type
1005	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 10: EAP-NOOB error codes

Assignment of new error codes MUST be done through IANA with "Specification Required" as defined in [RFC8126], except for the range 6001-6999. This range is reserved for "Private Use" and "Experimental Use", both locally and on the open Internet.

5.4. ServerInfo data fields

IANA is requested to create and maintain a new sub-registry entitled "EAP-NOOB ServerInfo data fields" in the "Nimble out-of-band authentication for EAP Parameters" registry. The initial values for this registry are:

Data Field	Specification
Type	This RFC Section 4
ServerName	This RFC Section 4
ServerURL	This RFC Section 4
SSIDList	This RFC Section 4
Base64SSIDList	This RFC Section 4

Table 11: ServerInfo data fields

Assignment of new values for new ServerInfo data fields MUST be done through IANA with "Specification Required" as defined in [RFC8126].

5.5. PeerInfo data fields

IANA is requested to create and maintain a new sub-registry entitled "EAP-NOOB PeerInfo data fields" in the "Nimble out-of-band authentication for EAP Parameters" registry. The initial values for this registry are:

Data Field	Specification
Type	This RFC Section 4
PeerName	This RFC Section 4
Manufacturer	This RFC Section 4
Model	This RFC Section 4
SerialNumber	This RFC Section 4
MACAddress	This RFC Section 4
SSID	This RFC Section 4
Base64SSID	This RFC Section 4
BSSID	This RFC Section 4

Table 12: PeerInfo data fields

Assignment of new values for new PeerInfo data fields MUST be done through IANA with "Specification Required" as defined in [RFC8126].

5.6. Domain name reservation

The special-use domain "eap-noob.arpa" should be registered in the .arpa registry (<https://www.iana.org/domains/arpa>). No A, AAAA, or PTR records are requested.

5.7. Guidance for Designated Experts

Experts SHOULD be conservative in the allocation of new Cryptosuites. Experts MUST ascertain that the requested values match the current Crypto Forum Research Group (CFRG) guidance on cryptographic algorithm security. Experts MUST ensure that any new Cryptosuites fully specify the encoding of the ECDHE public key and should include details such as the value of "kty" (key type) parameter when JWK [RFC7517] encoding is used.

Experts SHOULD be conservative in the allocation of new Message Types. Experts SHOULD ascertain that a well-defined specification for the new Message Type is permanently and publicly available.

Experts SHOULD be conservative in the allocation of new Error codes since the 6001-6999 range is already allocated for private and experimental use.

Experts MAY be liberal in the allocation of new ServerInfo and PeerInfo data fields. Experts MUST ensure that the Data Field requested has a unique name that is not easily confused with existing registrations. For example, requests for a new PeerInfo data field "ssid" should be rejected even though it is unique because it can be confused with the existing registration of "SSID". Experts MUST ensure that a suitable Description for the data field is available.

6. Implementation Status

Note to RFC Editor: Please remove this entire section and the reference to RFC7942 before publication.

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.

6.1. Implementation with wpa_supplicant and hostapd

- Responsible Organization: Aalto University
- Location: <https://github.com/tuomaura/eap-noob>
- Coverage: This implementation includes all the features described in the current specification. The implementation supports two-dimensional QR codes and NFC as example out-of-band (OOB) channels.
- Level of Maturity: Alpha
- Version compatibility: Version 08 of the individual draft implemented
- Licensing: BSD

- Contact Information: Tuomas Aura, tuomas.aura@aalto.fi

6.2. Implementation on Contiki

- Responsible Organization: University of Murcia and Aalto University
- Location: <https://github.com/eduingles/coap-eap-noob>
- Coverage: This implementation includes all the features described in the current specification. The implementation uses a blinking LED light as the out-of-band (OOB) channel.
- Level of Maturity: Alpha
- Version compatibility: Version 06 of the draft implemented
- Licensing: BSD
- Contact Information: Eduardo Ingles, eduardo.ingles@um.es

6.3. Implementation with wpa_supplicant and hostapd

- Responsible Organization: Ericsson
- Location: <https://github.com/Vogeltak/hostap>
- Coverage: This implementation is the most up-to-date one. The implementation only provides a minimal API interface for transferring OOB messages.
- Level of Maturity: Alpha
- Version compatibility: Version 02 of the working group adopted draft is implemented
- Licensing: BSD

6.4. Protocol modeling

The current EAP-NOOB specification has been modeled with the mCRL2 formal specification language [mcrl2]. The model [noob-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 an on-path 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 [noob-proverif] was used to verify security properties such as mutual authentication.

7. 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.

7.1. 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 attacks even in situations where the attacker is able to eavesdrop on the OOB channel and the nonce Noob is compromised. In some human-assisted OOB channels, such as human-perceptible audio or a user-typed URL, it may be easier to detect tampering than disclosure 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 attacks that may have happened 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., with a button press 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 on it from the OOB channel. Even if an attacker is able to eavesdrop on the secret nonce Noob, it nevertheless cannot perform a full impersonation attack on the in-band channel because a mismatching fingerprint would alert the OOB receiver, which would reject the OOB message. The attacker that eavesdropped on the secret nonce can impersonate the OOB receiver to the OOB sender. If it does, 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 credential 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 shared secret for each peer device, which makes the system more resilient against device compromise. Another advantage is that it is possible to revoke the security association for an individual device on the server side.

Forward secrecy during fast reconnect 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.

Another way in which some servers may control their computational load is to reuse the same ECDHE key for all peers over a short server-specific time window. In that case, forward secrecy will be achieved only after the server updates its ECDHE key, which may be a reasonable trade-off between security and performance. However, the server MUST NOT reuse the same ECDHE key with the same peer when rekeying with ECDHE (KeyingMode=2 or KeyingMode=3). Instead, it can simply not send an ECDHE key (KeyingMode=1).

The users delivering the OOB messages will often authenticate themselves to the EAP server, e.g., by logging into a secure web page or API. In this case, the server can 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.

7.2. Identifying correct endpoints

Potential weaknesses in EAP-NOOB arise from the fact that the user must identify physically the correct peer device. If the user mistakenly delivers the OOB message from the wrong peer device to the server, the server may create an association with the wrong peer. The reliance on the user in identifying the correct endpoints is an inherent property of user-assisted out-of-band authentication. To understand the potential consequences of the user mistake, we need to consider a few different scenarios. In the first scenario, there is no malicious party, and the user makes an accidental mistake between two out-of-the-box devices that are both ready to be registered to a server. If the user delivers the OOB message from the wrong device to the server, confusion may arise but usually no security issues. In the second scenario, an attacker intentionally tricks the user, for example, by substituting the original peer device with a compromised one. This is essentially a supply chain attack where the user accepts a compromised physical device.

There is also a third scenario, in which an opportunistic attacker tries to take advantage of the user's accidental mistake. For example, the user could play an audio or a blinking LED message to a device that is not expecting to receive it. In simple security bootstrapping solutions that transfer a master key to the device via the OOB channel, the device could misuse or leak the accidentally received master key. EAP-NOOB is not vulnerable to such opportunistic attackers because the OOB message has no value to anyone who did not take part in the corresponding Initial Exchange.

One mechanism that can mitigate user mistakes is certification of peer devices. A certificate or an attestation token (e.g.,[I-D.tschofenig-tls-cwt] and [I-D.ietf-rats-eat]) can convey to the server authentic identifiers and attributes, such as model and serial number, 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 for bootstrapping with EAP-NOOB.

Similarly, the attacker can try to trick the user into delivering the OOB message to the wrong server, so that the peer device becomes associated with the wrong server. If the EAP server is accessed through a web user interface, the attack is akin to phishing attacks where the user is tricked into accessing the wrong URL and wrong web page. OOB implementation with a dedicated app on a mobile device, which communicates with a server API at a pre-configured URL, can protect against such attacks.

After the device registration, an attacker could clone the device identity by copying the keys from the persistent EAP-NOOB association into another device. The attacker can be an outsider who gains access to the keys or the device owner who wants to have two devices matching the same registration. The cloning threats can be mitigated by creating the cryptographic keys and storing the persistent EAP-NOOB association on the peer device in a secure hardware component such as a trusted execution environment (TEE). Furthermore, remote attestation on the application level could provide assurance to the server that the device has not been cloned. Reconnect Exchange with a new cryptosuite (KeyingMode=3) will also disconnect all but the first clone that performs the update.

7.3. 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 press a reset button to return the device to the Unregistered state and to invoke a trusted UI. The problem with such trusted paths is that they are not standardized across devices.

Another potential consequence of a spoofed UI is the misbinding attack where the user tries to register a correct but compromised device, which tricks the user into registering another (uncompromised) device instead. For example, the 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 misbinding 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 on the server side can also prevent most misbinding attacks if the peer device sends its identifiers or attributes in the PeerInfo field and the server compares them with the expected values. The wrong but uncompromised device's PeerInfo will not match the expected values. Device certification by the manufacturer can further strengthen the asset tracking.

7.4. 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, which allocates many PeerId values and stores the ephemeral protocol state for them. The peer will typically only remember the latest ones. 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 on the server 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 attributes, such as the make and model or serial number. A compromised peer could lie in the PeerInfo which 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 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 (see Section 7.7).

7.5. 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 MUST check for downgrading attacks in the Reconnect Exchange as follows. 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. Determining the relative strength of the cryptosuites is out of scope of this specification and may be managed by implementations or by local policies at the peer and server.

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, an on-path 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 new applications. Currently, we expect most peer devices to support only one OOB direction, 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 the shared key material Kz whenever the cryptosuite is upgraded. We do not expect the cryptosuite upgrades to be frequent, but if an upgrade becomes necessary, it can be done without manual reset and reassociation of the peer devices.

7.6. Protected success and failure indications

Section 7.16 of [RFC3748] allows EAP methods to specify protected result indications because EAP-Success and EAP-Failure packets are neither acknowledged nor integrity protected. [RFC3748] notes that these indications may be explicit or implicit.

EAP-NOOB relies on implicit protected success indicators in the Completion and Reconnect exchange. Successful verification of MACs and MACs2 in the EAP-Request message from the server (message type 6 and message type 9, respectively) acts as an implicit protected success indication to the peer. Similarly, successful verification of MACp and MACp2 in the EAP-Response message from the peer (message type 6 and message type 9, respectively) act as an implicit protected success indication to the server.

EAP-NOOB failure messages are not protected. Protected failure result indications would not significantly improve availability since EAP-NOOB reacts to most malformed data by ending the current EAP conversation in EAP-Failure. However, since EAP-NOOB spans multiple conversations, failure in one conversation usually means no state change on the level of the EAP-NOOB state machine.

7.7. Channel Binding

EAP channel binding, defined in [RFC6677], means that the endpoints compare their perceptions of network properties, such as lower-layer identifiers, over the secure channel established by EAP authentication. Section 4.1 of [RFC6677] defines two approaches to channel binding. EAP-NOOB follows the first approach, in which the peer and server exchange plaintext information about the network over a channel that is integrity protected with keys derived during the EAP execution. More specifically, channel information is exchanged in the plaintext PeerInfo and ServerInfo objects and is later verified with message authentication codes (MACp, MACs, MACp2, MACs2). This allows policy-based comparison with locally perceived network properties on either side, as well as logging for debugging purposes. The peer MAY include in PeerInfo any data items that it wants to bind to the EAP-NOOB association and to the exported keys. These can be properties of the authenticator or the access link, such as the SSID and BSSID of the wireless network (see Table 6). As noted in Section 4.3 of [RFC6677], the scope of the channel binding varies between deployments. For example, the server may have less link-layer information available from roaming networks than from a local enterprise network, and it may be unable to verify all the network properties received in PeerInfo. There are also privacy considerations related to exchanging the ServerInfo and PeerInfo while roaming (see Section 7.10).

Channel binding to secure channels, defined in [RFC5056], binds authentication at a higher protocol layer to a secure channel at a lower layer. Like most EAP methods, EAP-NOOB exports the session keys MSK and EMSK, and an outer tunnel or a higher-layer protocol can bind its authentication to these keys. Additionally, EAP-NOOB exports the key AMSK, which may be used to bind application-layer authentication to the secure channel created by EAP-NOOB and to the session keys MSK and EMSK.

7.8. Denial of Service

While denial-of-service (DoS) attacks by on-link attackers cannot be fully prevented, the design goal in EAP-NOOB is to void long-lasting failure caused by an attacker who is present only temporarily or intermittently. The main defense mechanism is the persistent EAP-NOOB association, which is never deleted automatically due to in-band messages or error indications. Thus, the endpoints can always use the persistent association for reconnecting after the DoS attacker leaves the network. In this sense, the persistent association serves the same function in EAP-NOOB as a permanent master key or certificate in other authentication protocols. We discuss logical attacks against the updates of the persistent association in Section 7.9.

In addition to logical DoS attacks, it is necessary to consider resource exhaustion attacks against the EAP server. The number of persistent EAP-NOOB associations created in the server is limited by the need for a user to assist in delivering the OOB message. The users can be authenticated for the input or output of the OOB message at the EAP server, and any users who create excessive numbers of persistent associations can be held accountable and their associations can be deleted by the server administrator. What the attacker can do without user authentication is to perform the Initial Exchange repeatedly and create a large number of ephemeral associations (server in state 1, Waiting for OOB) without ever delivering the OOB message. Above in Section 7.4, it was suggested that the server should store the ephemeral states for at least a day. This may require off-loading the state storage from memory to disk during a DoS attack. However, if the server implementation is unable to keep up with a rate of Initial Exchanges performed by a DoS attacker and needs to drop some ephemeral states, no damage is caused to already created persistent associations, and the honest users can resume registering new peers when the DoS attacker leaves the network.

There are some trade-offs in the protocol design between polite back-off and giving way to DoS attackers. An on-link DoS attacker could spoof the SleepTime value in the Initial Exchange or Waiting Exchange

to cause denial of service against a specific peer device. There is an upper limit on the SleepTime (3600 seconds) to mitigate the spoofing threat. This means that, in the presence of an on-link DoS attacker who spoofs the SleepTime, it could take up to one hour after the delivery of the OOB message before the device performs the Completion Exchange and becomes functional. Similarly, the Unwanted peer error (error code 2001) could cause the peer to stop connecting to the network. If the peer device is able to alert the user about the error condition, it can safely stop connecting to the server and wait for the user to trigger a reconnection attempt, e.g., by resetting the device. As mentioned in Section 3.6.2, peer devices that are unable to alert the user should continue to retry the Initial Exchange infrequently to avoid a permanent DoS condition. We believe a maximum backoff time of 3600 seconds is reasonable for a new protocol because malfunctioning or misconfigured peer implementations are at least as great a concern as DoS attacks, and politely backing off within some reasonable limits will increase the acceptance of the protocol. The maximum backoff times could be updated to be shorter as the protocol implementations mature.

7.9. 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=6) 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=9) 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 method implementations 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 a state mismatch and persistent denial of service until the 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.

7.10. Privacy considerations

There are privacy considerations related to performing the Reconnect Exchange while roaming. The peer and server may send updated PeerInfo and ServerInfo fields in the Reconnect Exchange. This data is sent unencrypted between the peer and the EAP authenticator, such as a wireless access point. Thus, it can be observed by both outsiders and the access network. The PeerInfo field contains identifiers and other information about the peer device (see Table 6). While the information refers to the peer device and not directly to the user, it can leak information about the user to the access network and to outside observers. The ServerInfo, on the other hand, can leak information about the peer's affiliation with the home network. For this reason, the optional PeerInfo and ServerInfo in the Reconnect Exchange SHOULD be omitted unless some critical data has changed and it cannot be updated on the application layer.

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 add misbehaving peer devices to a deny list, as well as, how to avoid loss of synchronization between the server and the peer if messages are lost during the identifier update.

To enable stronger identity protection in later versions of EAP-NOOB, the optional server-assigned NAI (NewNAI) SHOULD have a constant username part. The RECOMMENDED username is "noob". The server MAY, however, send a different username in NewNAI to avoid username collisions within its realm or to conform to a local policy on usernames.

7.11. 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 13.

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
Confidentiality	no
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 13: EAP security claims

8. References

8.1. Normative references

[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.

[FIPS186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS 186-4 , July 2013.

[NIST-DH] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 3 , April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.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>.

[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>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

[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>.

[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>.

8.2. Informative references

[BluetoothPairing] Bluetooth, SIG, "Simple pairing whitepaper", Technical report , 2007.

[I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-10 (work in progress), June 2021.

[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, "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>.

[noob-mcrl2]
Peltonen, A., "mCRL2 model of EAP-NOOB", 2021, <https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/mcrl2>.

[noob-proverif]
Peltonen, A., "ProVerif model of EAP-NOOB", 2021, <https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/proverif >.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[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>.

[RFC6677] Hartman, S., Ed., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, DOI 10.17487/RFC6677, July 2012, <https://www.rfc-editor.org/info/rfc6677>.

[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	Initial Exchange	both 1 (0 on error)
3	-	no change, notify user
server state: Waiting for OOB (1)		
0	Initial Exchange	both 1 (0 on error)
1	Waiting Exchange	both 1 (no change on error)
2	Completion Exchange	both 4 (A)
3	-	no change, notify user
server state: OOB Received (2)		
0	Initial Exchange	both 1 (0 on error)
1	Completion Exchange	both 4 (B)
2	Completion Exchange	both 4 (A)
3	-	no change, notify user
server state: Reconnecting (3) or Registered (4)		
0..2	-	no change, notify user
3	Reconnect Exchange	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	Mobility/timeout/network failure	0
3..4	Mobility/timeout/network failure	3
3..4	Rekeying request	3
0..4	User resets association	0
0..4	Association state recovery	3

Figure 12: Local events on server and peer

Appendix B. Application-specific parameters

Table 14 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 implementers 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. When the OOB channel has error detection or correction, the RECOMMENDED value is 5.
NoobTimeout	How many seconds the sender of the OOB message remembers the sent Noob value. The RECOMMENDED value is 3600 seconds.
ServerInfoType	The value of the Type field and the other required fields in ServerInfo
PeerInfoType	The value of the Type field and the other required fields in PeerInfo

Table 14: OOB channel characteristics

Appendix C. 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 allowing the server to assign a NAI to the peer. After the NAI has been assigned, it 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 NAI of the home network. This is the user or application configured NAI mentioned in Section 3.3.1. In that case, the EAP-NOOB association can be created while roaming. The configured NAI 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 field called either SSIDList or Base64SSIDList. The list is formatted as explained in Table 6. 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 D. 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 dynamic 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 https or another server-authenticated scheme, so that there is a secure connection to the server and the on-path attacker cannot read or modify the OOB message.

The ServerInfo in this case includes a field called ServerURL of the following format with RECOMMENDED length of at most 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 ASCII string. The peer base64url encodes, without padding, the 16-byte values Noob and Hoob into 22-character ASCII 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://aaa.example.com/eapnoob?P=mcm5BSCDZ45cYPlAr1ghNw&N=rMinS0-F4E
fCU8D9ljxX_A&H=QvnMp4UGxuQVFaXPW_14UW

Appendix E. Version history

- Version 01:
  - Fixed Reconnection Exchange.
  - URL examples.
  - Message examples.
  - Improved state transition (event) tables.
- 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-response pair, not in NAI.

  * New subsection for the common handshake in all exchanges to avoid repetition.

- Version 07:

  * Updated example messages.

  * Added pointers to new implementation in Contiki.

- Version 08:

  * Editorial improvements and corrections.

- WG Version 00:

  * Editorial improvements and corrections.

  * Updated reference [NIST-DH] to Revision 3 (2018).

- WG Version 01:

  * Add NIST P-256 as Cryptosuite 2.

  * Renumber message types.

  * Very minor editorial fixes.

- WG Version 02:

  * Updated message examples with all KeyingModes.

  * Many editorial fixes and other updates based on the IoT directorate review of Dave Thaler.

  * Text on cloning attacks based on review by Hannes Tschofenig.

- WG Version 03:

  * Changed server-assigned Realm to server-assigned NAI to avoid username collisions. This issue was identified in a review by Alan Dekok.

  * Minor editorial improvements.

- WG Version 04:

  * Use of more inclusive language.

  * Minor changes to IANA registration policies.

  * Explain how relative strength of cryptosuites is determined.

  * Improvements based on AD review from Roman Danyliw and shepherd review from Joseph Salowey.

- WG Version 05:

  * Many text clarifications based on IESG evaluation.

  * Security considerations subsections on success indications, channel binding, and denial of service.

  * Privacy considerations gathered to a separate section.

- WG Version 06:

  * Remove leftover text in section on identifying correct endpoints.

  * Example messages removed.

Appendix F. Acknowledgments

Max Crone, Shiva Prasad TP, and Raghavendra MS implemented parts of this protocol with wpa_supplicant and hostapd. Eduardo Ingles (RFC editor: please add an accented e in Ingles) and Dan Garcia-Carrillo were involved in the implementation of this protocol on Contiki. Their inputs helped us in improving the specification.

The authors would 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, Alan Dekok, Darshak Thakore, Stefan Winter, Hannes Tschofenig, Daniel Migault, Roman Danyliw, Benjamin Kaduk, Francesca Palombini, Steve Hanna, Lars Eggert, and Eric Vyncke for their comments and reviews.

We would also like to express our sincere gratitude to Dave Thaler for his thorough review of the document.

Authors' Addresses

Tuomas Aura
Aalto University
Aalto 00076
Finland

EMail: tuomas.aura@aalto.fi

Mohit Sethi
Ericsson
Jorvas 02420
Finland

EMail: mohit@piuha.net

Aleksi Peltonen
Aalto University
Aalto 00076
Finland

EMail: aleksi.peltonen@aalto.fi

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.

Table of Contents

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).

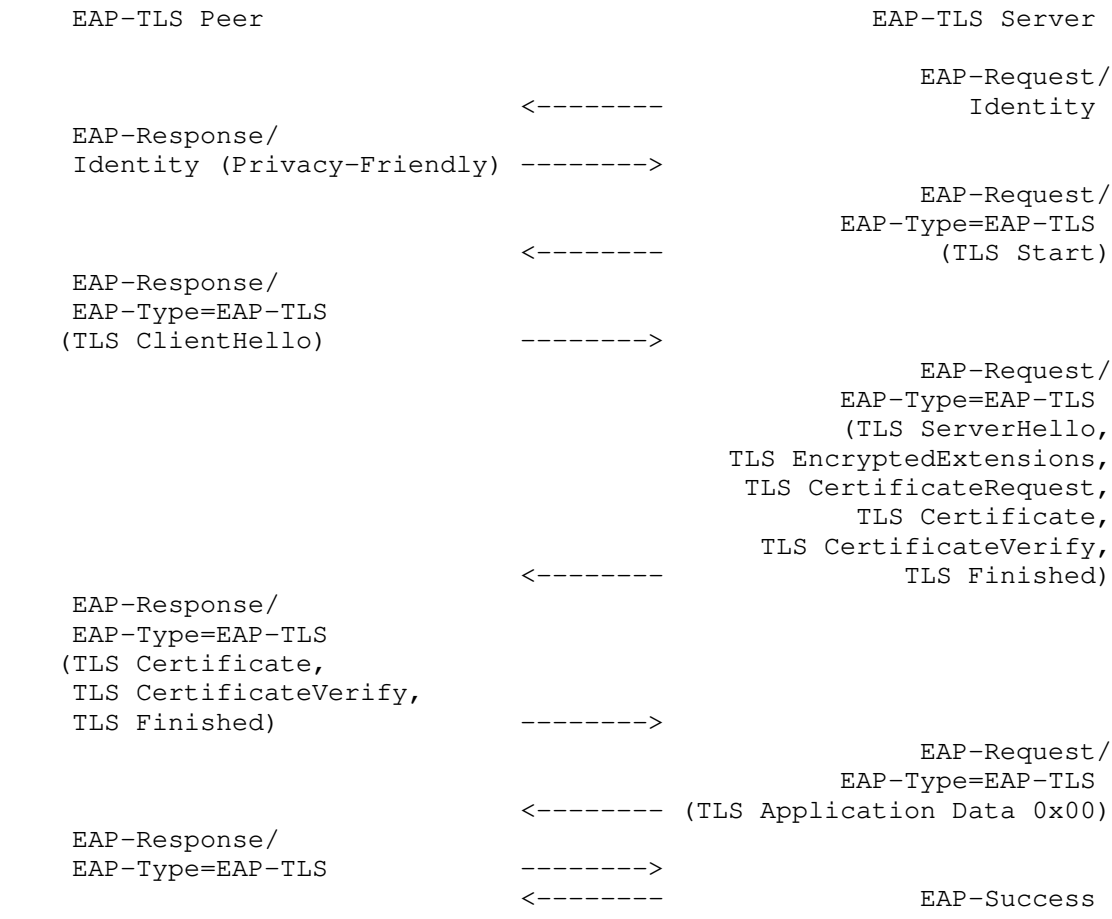


Figure 1: EAP-TLS mutual authentication

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.

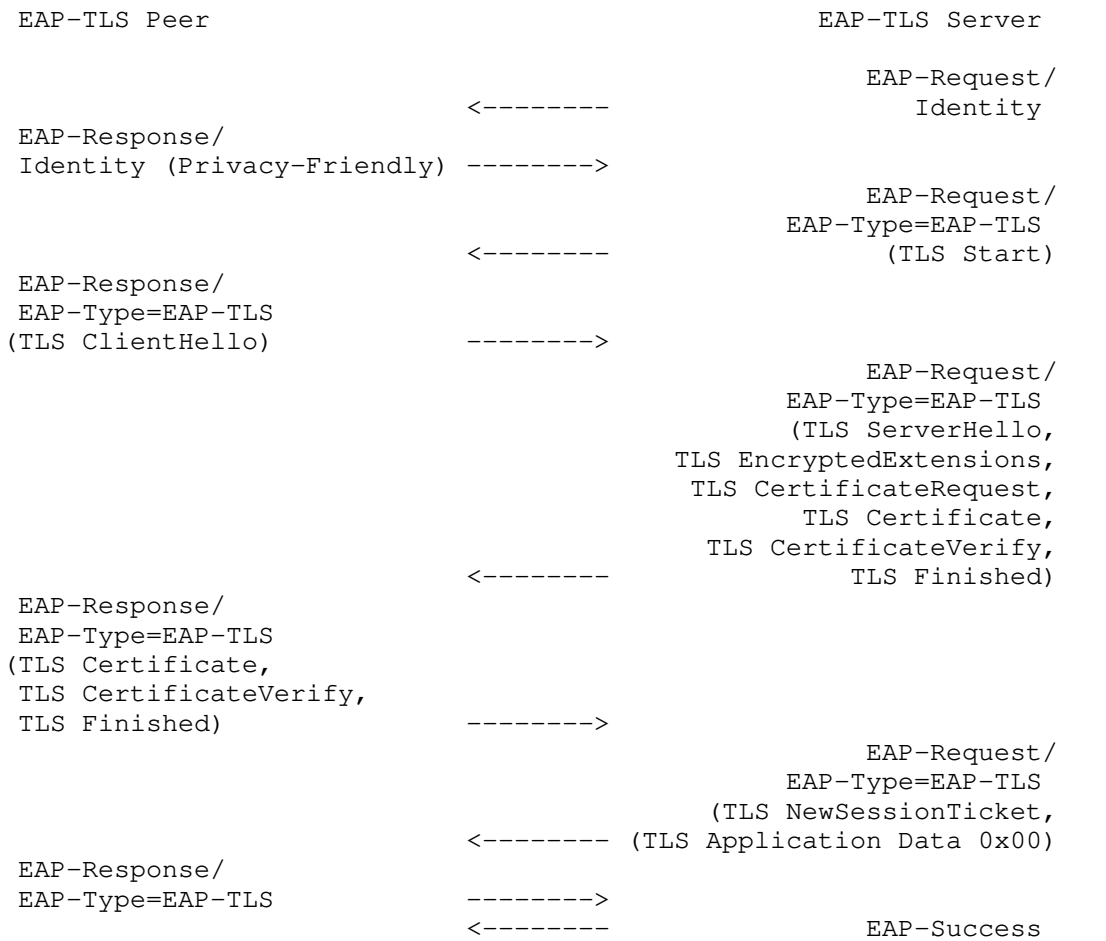


Figure 2: EAP-TLS ticket establishment

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.

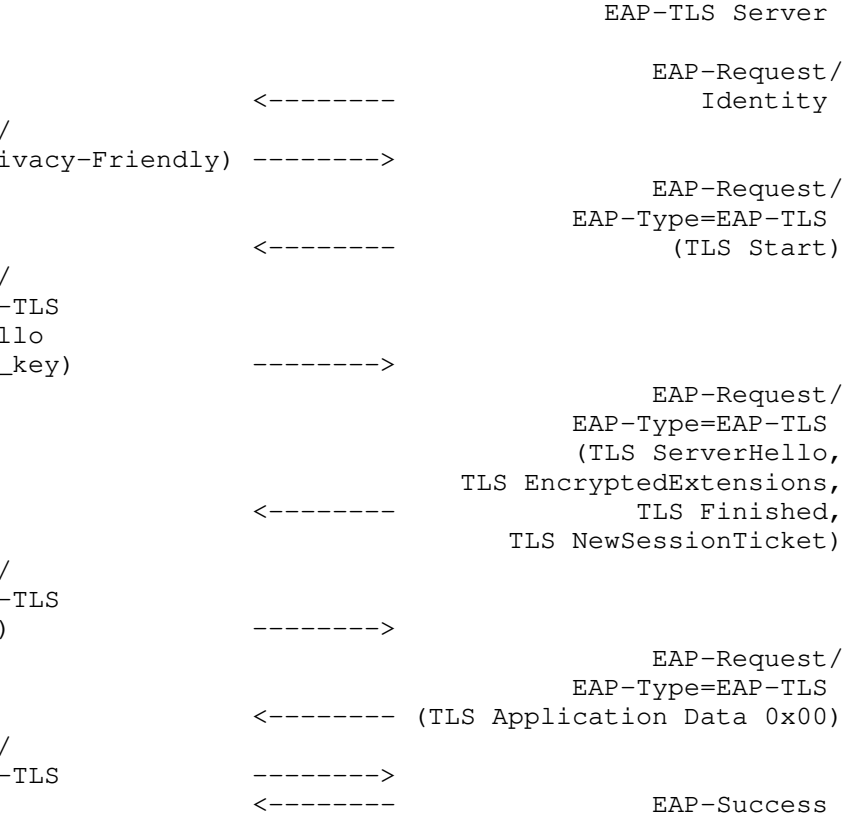


Figure 3: EAP-TLS resumption

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.

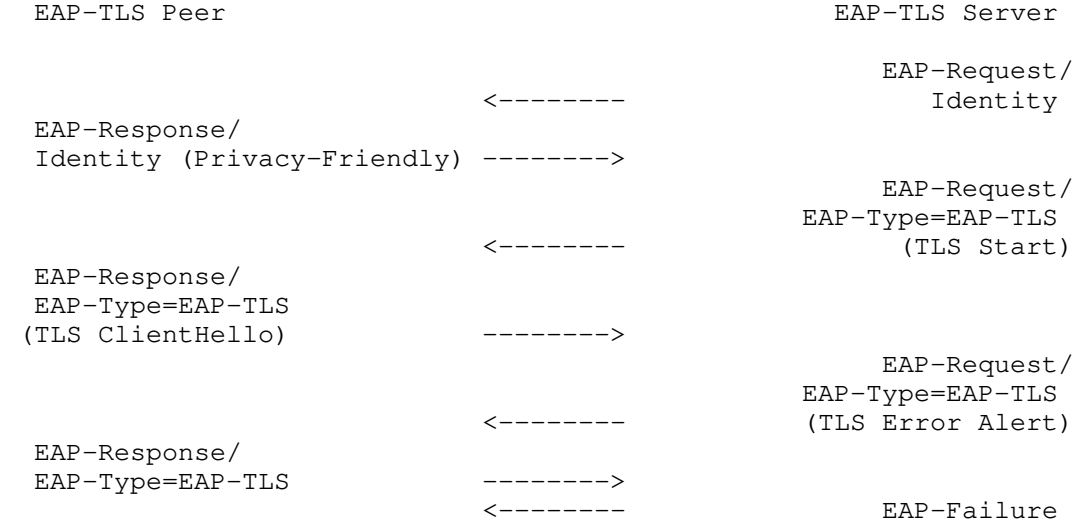


Figure 4: EAP-TLS server rejection of ClientHello

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

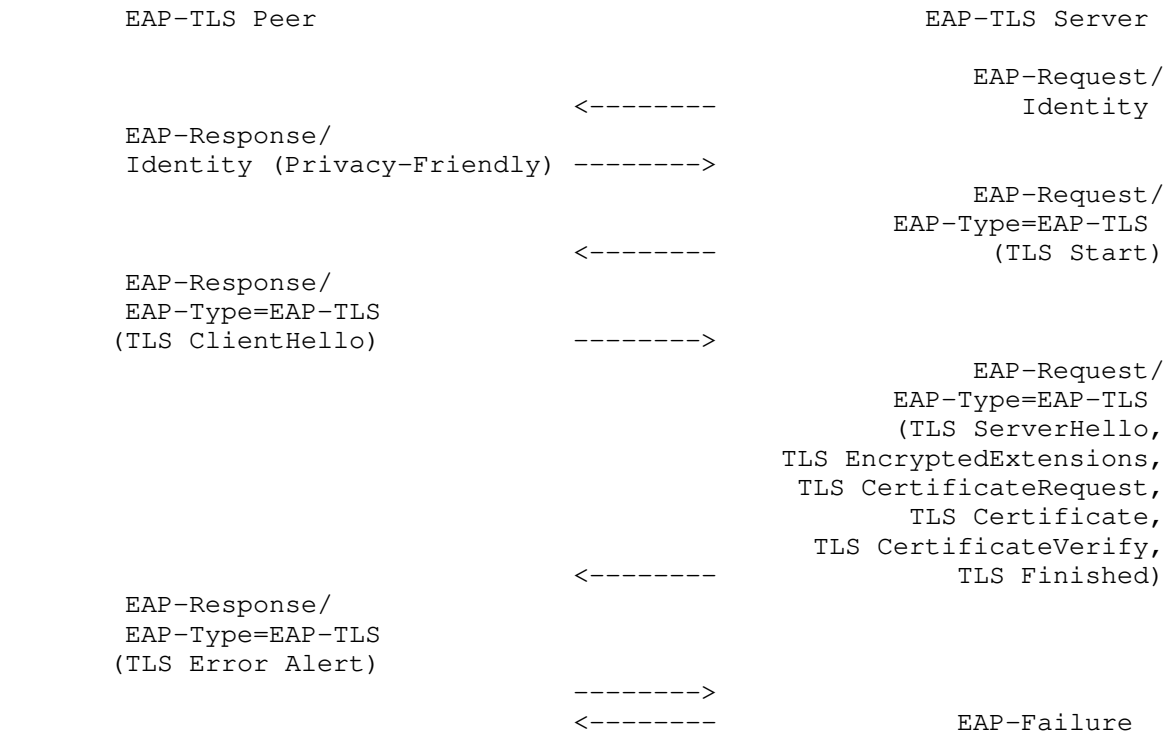


Figure 5: EAP-TLS unsuccessful EAP-TLS server authentication

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.

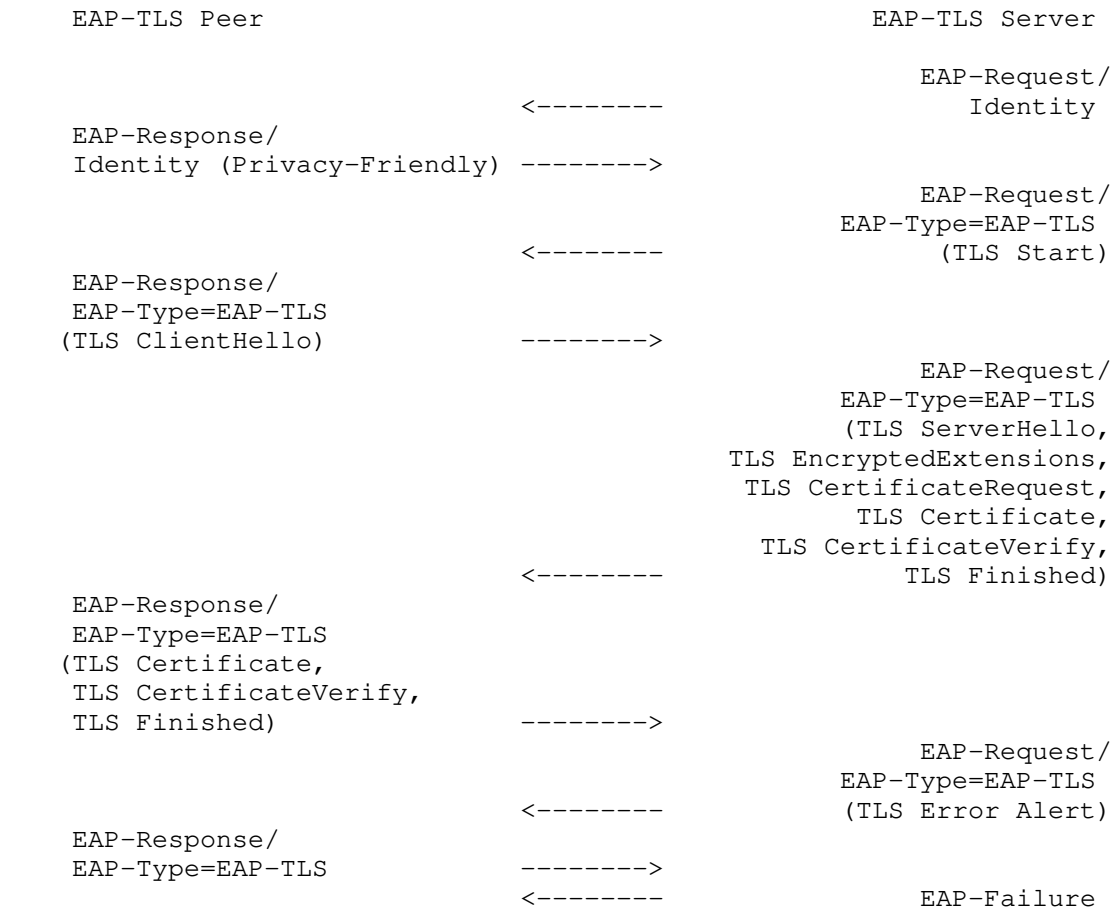


Figure 6: EAP-TLS unsuccessful client authentication

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]).

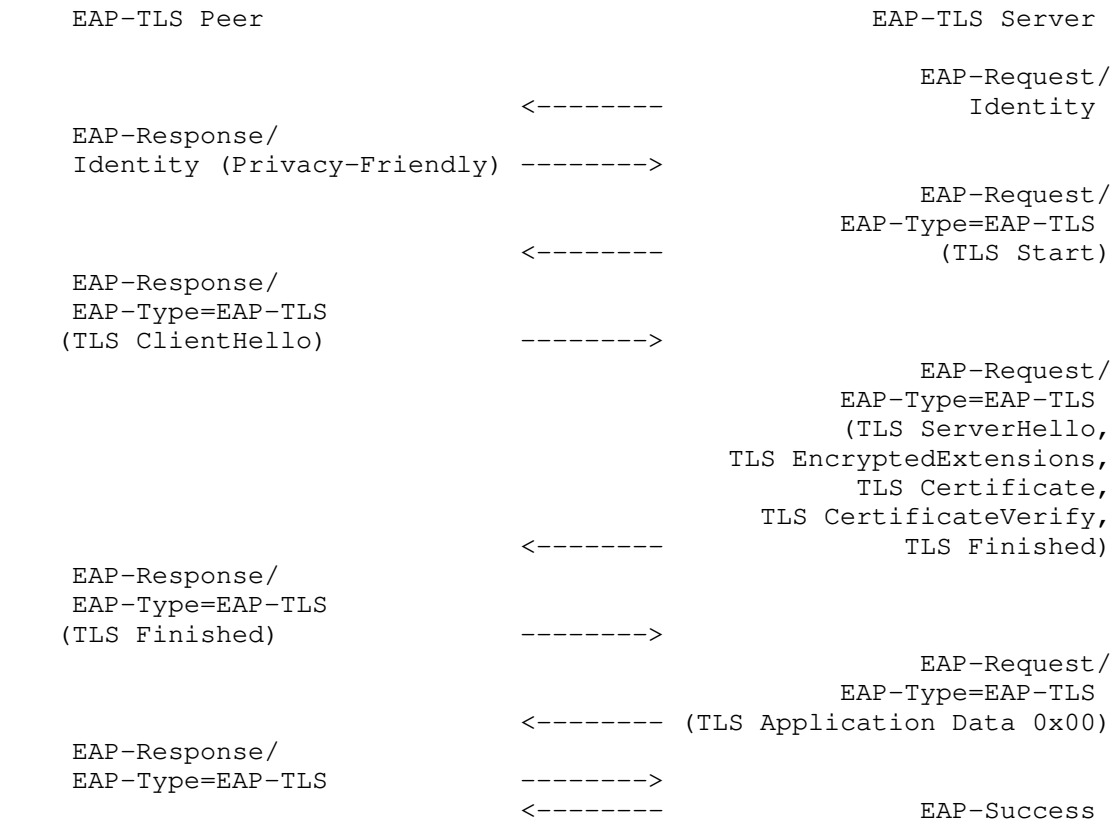


Figure 7: EAP-TLS without peer authentication

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.

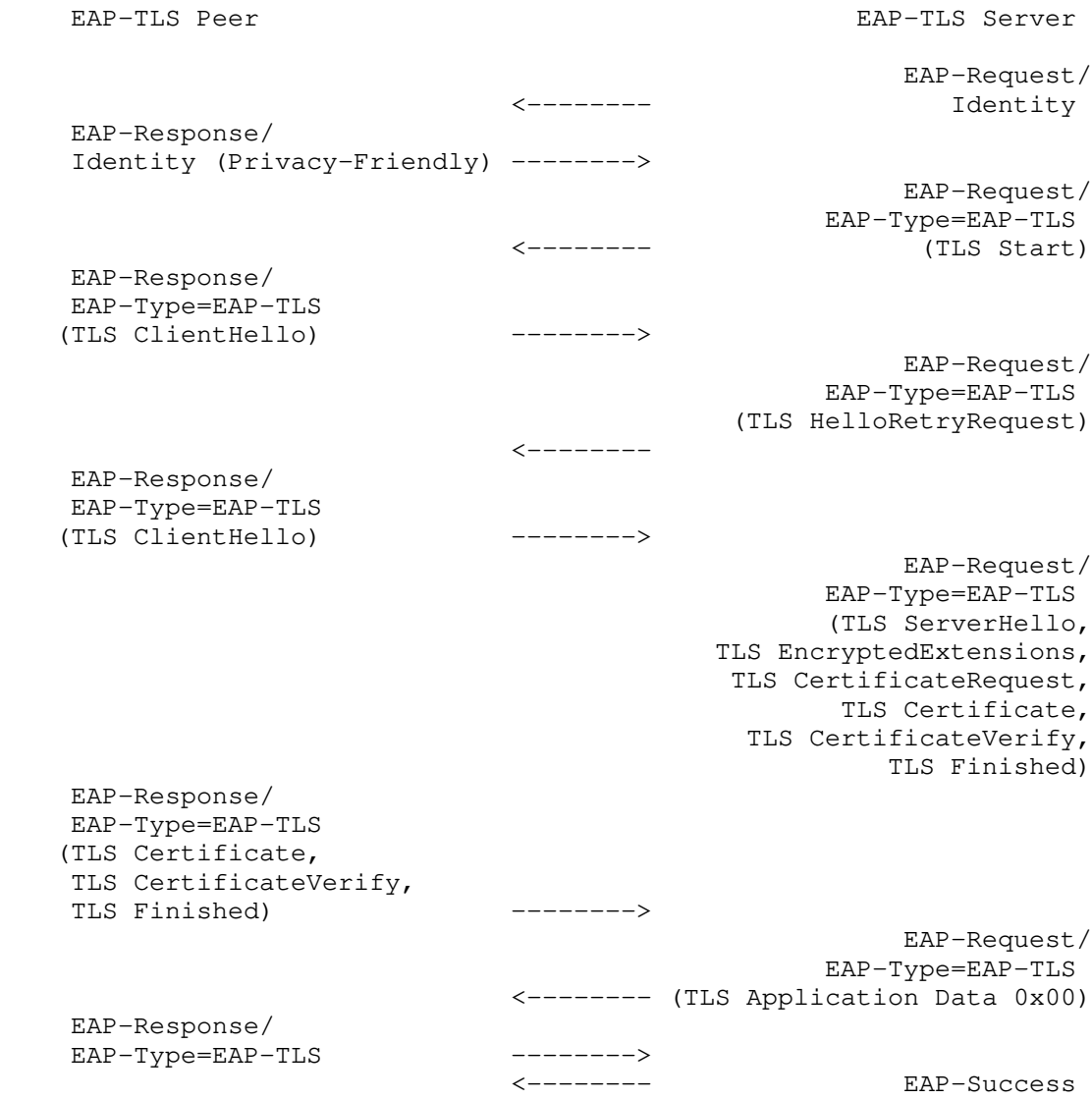


Figure 8: EAP-TLS with Hello Retry Request

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+RKPE3OTrYLn4AQc4AC2U=@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 (OSCP), 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.

[MulteFire]
MulteFire, "MulteFire 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. Kroeselberg, "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-eaptlscert]
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-eaptlscert-08, 20 November 2020, <https://www.ietf.org/archive/id/draft-ietf-emu-eaptlscert-08.txt>.

[I-D.ietf-tls-ticketrequests]
Pauly, T., Schinazi, D., and C. A. Wood, "TLS Ticket Requests", Work in Progress, Internet-Draft, draft-ietf-tls-ticketrequests-07, 3 December 2020, <https://www.ietf.org/archive/id/draft-ietf-tls-ticketrequests-07.txt>.

[I-D.ietf-emu-tls-eap-types]
DeKok, A., "TLS-based EAP types and TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-emu-tls-eap-types-03, 22 June 2021, <https://www.ietf.org/archive/id/draft-ietf-emu-tls-eap-types-03.txt>.

[I-D.ietf-tls-rfc8446bis]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-02, 23 August 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-rfc8446bis-02.txt>.

Appendix A. Updated References

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

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

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

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

Acknowledgments

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

Contributors

Alan DeKok, FreeRADIUS

Authors' Addresses

John Preuß Mattsson
Ericsson
SE-164 40 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Mohit Sethi
Ericsson
FI-02420 Jorvas
Finland

Email: mohit@piuha.net

Network Working Group
INTERNET-DRAFT
Updates: 4851, 5281, 7170
Category: Standards Track
Expires: August 16, 2023

DeKok, Alan
FreeRADIUS

16 February 2023

TLS-based EAP types and TLS 1.3
draft-ietf-emu-tls-eap-types-13.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 EAP-FAST (RFC 4851), EAP-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.

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 EAP-FAST [RFC4851], EAP-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 are addressed by 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 2.1.

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 result indication, but only during resumption. When other TLS-based EAP methods use full authentication, the result indication 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 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.

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.

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. These definitions include ones for the Master Session Key (MSK) and the Extended Master Session Key (EMSK):

```
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. The inclusion of the EAP type makes the derivation method-specific. There is no need to use different labels for different EAP types, as was done earlier.

These definitions apply in their entirety to EAP-TTLS [RFC5281] and PEAP as defined in [PEAP] and [MSPEAP]. Some definitions apply to EAP-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

TEAP previously used a Protected Access Credential (PAC), which is functionally equivalent to session tickets provided by TLS 1.3 which contain a pre-shared key (PSK) along with other data. As such, the use of a PAC is deprecated for TEAP in TLS 1.3. PAC provisioning as defined in [RFC7170] Section 3.8.1 is also no longer part of TEAP when TLS 1.3 is used.

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

```
IMSK[j] = TLS-Exporter("TEAPbindkey@ietf.org", secret, 32)
```

The secret is the EMSK or MSK from the j'th inner method. When an inner method does not provide an EMSK or MSK, IMSK[j] is 32 octets of zero.

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 EAP-FAST, but using the EAP-FAST Type.

The derivation of the Inner Method Session Keys (IMSK), Inner Method Compound Keys (IMCK), and Compound Session Keys (CMK) is given below.

```
session_key_seed = TLS-Exporter("EXPORTER: teap 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]
```

In these definitions, || denotes concatenation.

In TLS 1.3, the derivation of IMCK[j] uses both a different label, and a different order of concatenating fields, than was used by TEAP with TLS 1.2. Similarly, the session_key_seed in TLS 1.3 uses the Type as the context, where in TLS 1.2 the context was a zero-length field.

The outer MSK and EMSK are then derived from the final ("n"th) inner method, as follows:

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

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

The TEAP Compound MAC defined in [RFC7170] Section 5.3 remains the same, but the message authentication code (MAC) for TLS 1.3 is computed with the HMAC algorithm negotiated for HKDF in the key

schedule, as per section 7.1 of RFC 8446. That is, the MAC used is the MAC derived from the TLS handshake.

Compound-MAC = MAC(CMK[n], BUFFER)

Where we define CMK[n] as the CMK taken from the final ("n"th) inner 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. That is, the MAC used is the MAC derived from the TLS handshake.

The definition of BUFFER is unchanged from [RFC7170] Section 5.3.

2.2.1. Client Certificates

The use of client certificates is still permitted when using TEAP 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 [RFC7170] Section 7.6. If there is no Phase 2 data, then the EAP server MUST reject the session.

That is, while [RFC7170] 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 TEAP 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 TEAP.

[RFC7170] Section 7.4.1 suggest that client certificates should be sent in Phase 2 of the TEAP exchange, "since TLS client certificates are sent in the clear". While TLS 1.3 no longer sends client certificates in the clear, TEAP implementations need to distinguish identities for both User and Machine using the Identity-Type TLV (with values 1 and 2, respectively). When a client certificate is sent outside of the TLS tunnel, it MUST include Identity-Type as an outer TLV, in order to signal the type of identity which that client certificate is for.

2.3. EAP-FAST

For EAP-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 functionally equivalent to session tickets provided by TLS 1.3 which contain a pre-shared key (PSK) along with other data. As such, 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.

2.3.1. Client Certificates

The use of client certificates is still permitted when using EAP-FAST 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 [RFC4851] Section 7.4.1. If there is no Phase 2 data, then the EAP server MUST reject the session.

That is, while [RFC4851] implicitly permits the use of client certificates without proceeding to Phase 2, this practice is forbidden when EAP-FAST 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 EAP-FAST.

2.4. EAP-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-CHAPv2 (Section 11.2.4), and to be 9 octets for MS-CHAP (Section 11.2.3).

When PAP, CHAP, or MS-CHAPv1 are used as inner authentication methods, there is no opportunity for the EAP server to send a protected success indication, as is done in [RFC9190] Section 2.5. Instead, when TLS session tickets are disabled, the response from the EAP server MUST be either EAP-Success or EAP-Failure. These responses are unprotected, and can be forged by a skilled attacker.

Where TLS session tickets are enabled, the response from the EAP server may also continue TLS negotiation with a TLS NewSessionTicket message. Since this message is protected by TLS, it can serve as the protected success indication.

It is therefore RECOMMENDED that EAP servers always send a TLS NewSessionTicket message, even if resumption is not configured. When the EAP peer attempts to use the ticket, the EAP server can instead request a full authentication. As noted earlier, implementations SHOULD NOT send TLS NewSessionTicket messages until the "inner tunnel" authentication has completed, in order to take full advantage of the message as a protected success indication.

When resumption is not used, the TLS NewSessionTicket message is not available, and some authentication methods will not have a protected success indication. While we would like to always have a protected success indication, limitations of the underlying protocols, implementations, and deployment requirements make that impossible.

EAP peers MUST continue running their EAP state machine until they receive either an EAP-Success, or an EAP-Failure. Receiving a TLS NewSessionTicket message in response to inner method PAP, CHAP, or MS-CHAP authentication is normal, and MUST NOT be treated as a failure.

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 EAP-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 EAP-TTLS.

The use of client certificates is still permitted when using EAP-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 EAP method keying material. The pseudo-random function (PRF+) used in [PEAP-MPPE] 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.

Note that the above derivation uses SHA-1, which may be formally deprecated in the near future.

However, the pseudo-random function (PRF+) calculation uses a PEAP Tunnel Key which is defined in [PEAP-PRF] 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 the text in [PEAP-PRF] 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 here in Section 2.1, instead of using the TLS-PRF-128 derivation given in [PEAP-PRF]. The method defined in [PEAP-TK] MUST NOT be used.

2.5.1. Client Certificates

As with EAP-TTLS, [PEAP] permits the use of client certificates in addition to inner tunnel methods. The practice of using client certificates with no "inner method" is forbidden when PEAP 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 PEAP.

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 receipt of a Finished message 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 send application data along with the Finished message. It was then possible for implementations to assume that a receipt of a Finished message 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. TLS-based EAP methods such as EAP-TTLS, PEAP, and EAP-FAST each have method-specific application data which MUST be processed according to the EAP type.

TLS 1.3 in [RFC8446] Section 4.6.1 also permits NewSessionTicket messages to be sent after the server has received the client Finished message, which is a change from earlier TLS versions. This change can cause 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 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. In addition to prolonging the packet exchange, using a separate NewSessionTicket message can lead to non-interoperable implementations.

[RFC9190] Section 2.5 requires a protected result indication 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

For EAP-TLS, [RFC9190] Sections 2.1.3 and 2.1.7 recommend the use of anonymous Network Access Identifiers (NAIs) [RFC7542] in the EAP Response/Identity 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 for the outer identity as per [RFC7542] Section 2.4 has a few benefits. An NAI allows the EAP session to be routed in an AAA framework as described in [RFC7542] Section 3. Using an anonymous realm also ensures that user identifiers are kept private.

As for the inner identity, we define it generically as the identification information carried inside of the TLS tunnel. For PEAP, that identity may be an EAP Response/Identity. For EAP-TTLS, it may be the User-Name attribute. Vendor-specific EAP methods which use TLS will generally also have an inner identity. This identity is carried inside of the TLS tunnel, and is therefore both routed to the correct destination by the outer identity, and kept private by the use of TLS.

In other words, we can view the outer TLS layer of tunneled EAP methods as a secure transport layer which is responsible for getting the actual (inner) authentication credentials securely from the EAP peer to the EAP server. The EAP server then uses the inner identity and inner authentication data to identify and authenticate a particular user.

As the authentication data is routed to the correct destination, there is little reason for the inner identity to also contain a realm. We therefore have a few recommendations on the inner and outer identities, along with their relationship to each other.

The outer identity SHOULD use an anonymous NAI realm, which allows for both user privacy, and for the EAP session to be routed in an AAA framework as described in [RFC7542] Section 3. Where NAI realms are not used, packets will not be routable outside of the local organization.

The inner identity MUST NOT use an anonymous NAI realm. 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. Many organizations typically use only one realm for all user accounts.

However, there are situations where it is useful for an inner identity to contain a realm. For example, an organization may have multiple independent sub-organizations, each with a different and unique realm. These realms may be independent of one another, or the realms may be a subdomain (or subdomains) of the public outer realm.

In that case, an organization can configure one public "routing" realm, and multiple separate "inner" realms. This separation of realms also allows an organization to split users into logical groups by realm, where the "user" portion of the NAI may otherwise conflict. For example, "user@example.com" and "user@example.org" are different NAIs which can both be used as inner identities.

Using only one public realm both keeps internal information private, and also simplifies realm management for external entities by minimizing the number of realms which have to be tracked by them.

In most situations, routing identifiers should be associated with the authentication data that they are routing. For example, if a user has an inner identity of "user@example.com", then it generally makes little 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".

Where an EAP server receives an inner identity for a realm which it is not authoritative, it MUST reject the authentication. There is no reason for one organization to authentication users from a different (and independent) organization.

In addition, associating inner/outer identities from different organizations 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 for user authentication. The inner identity can then be fully-qualified: user name plus realm of the organization. This practice may result in successful authentications, but it has practical difficulties.

For example, an organization may host their own AAA servers, but use a "cloud" identity provider to hold user accounts. In that situation, the organizations could see try to 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. There is no reason to tie the configuration of user systems to public realm routing, that configuration more properly belongs in the network.

Both of these practices mean that changing "cloud" providers is difficult. When such a change happens, each individual EAP peer must be updated with a different outer identity which points to the new "cloud" provider. This process can be expensive, and some EAP peers 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. The requirements on identities, etc. remain unchanged from that document.

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

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, such as with 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

RFC Editor: Please remove this section before publication.

EAP-TTLS and PEAP are implemented and tested to be interoperable with wpa_supplicant 2.10 and Windows 11 as EAP peers, and FreeRADIUS 3.0.26 and Radiator as RADIUS / EAP 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 EAP-TTLS both work with Radiator and FreeRADIUS.

Implementors have demonstrated significant interest in getting PEAP and EAP-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.

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. Handling of TLS NewSessionTicket Messages

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, [RFC84346] Section 4.6.1 allows that "At any time after the server has received the client Finished message, it MAY send a NewSessionTicket message." This message is sent by the server before the inner authentication method has been run, and therefore before the user has been 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 message. The malicious client can then abort the authentication session, and use the obtained NewSessionTicket to "resume" the previous session. If the server allows the session to resume without verifying that the user had first been authenticated, the malicious client can then obtain network access without ever being authenticated network access without ever being authenticated.

As a result, EAP servers MUST NOT assume that a user has been authenticated simply because a TLS session is being resumed. Even if a session is being resumed, an EAP server MAY have policies which still force the inner authentication methods to be run. For example, the users password may have expired in the time interval between first authenticaction, and session resumption.

The guidelines given here therefore describe situations where an EAP server is permitted to allow session resumption, not where it is required to allow session resumption. An EAP server could simply refuse to issue session tickets, or could run the full inner authentication even if a session was resumed.

Where session tickets are used, the EAP server SHOULD track the successful completion of an inner authentication, and associate that status with any session tickets issued for that session. This requirement can be met in a number of different ways.

One way is for the EAP server to simply not send any TLS NewSessionTicket messages until the inner authentication has completed successfully. The EAP server then knows that the existence of a session ticket is proof that a user was authenticated, and the session can be resumed.

Another way is for the EAP server to simple discard or invalidate any session tickets until after the inner authentication has completed successfully. When the user is authenticated, a new TLS NewSessionTicket message can be sent to the client, and the new ticket cached and/or validated.

Another way is for the EAP server to associate the inner authentication status with each session ticket. When a session ticket is used, the authentication status is checked. When a session ticket shows that the inner authentication did not succeed, the EAP server MUST run the inner authentication method(s) in the resumed tunnel, and grant only access based on the success or failure of those inner methods/

However, the interaction between EAP implementations and any underlying TLS library may be complex, and the EAP server may not be

able to make the above guarantees. Where the EAP server is unable to determine the users authentication status from the session ticket, it MUST assume that inner authentication has not completed, and it MUST run the inner authentication method(s) successfully in the resumed tunnel before granting access.

This issue is not relevant for EAP-TLS, which only uses client certificates for authentication in the TLS handshake. It is only relevant for TLS-based EAP methods which do not use the TLS layer to authenticate

6.2. Protected Success and Failure indications

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

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

For full authentication, the other TLS-based EAP methods do not provide for protected success and failure indications as part of the outer TLS exchange. That is, the protected result indication 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 indications as part of the authentication exchange performed inside of the TLS tunnel. These result indications are therefore protected, as they cannot be modified or forged.

However, some inner methods do not provide for success or failure indications. For example, the use of EAP-TTLS with inner PAP, CHAP, or MS-CHAP. Those methods send authentication credentials to the EAP 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 result indications. PAP can often be replaced with EAP-GTC, CHAP with EAP-MD5, and MS-CHAPv1 with MS-CHAPv2 or EAP-MSCHAPv2. All of the replacement methods provide for similar functionality, and have protected success and failure indication. The main cost to this change is additional round trips.

It is RECOMMENDED that implementations deprecate inner tunnel methods which do not provide protected success and failure indications when

TLS session tickets cannot be used. Implementations SHOULD use EAP-GTC instead of PAP, and EAP-MD5 instead of CHAP. Implementations SHOULD use MS-CHAPv2 or EAP-MSCHAPv2 instead of MS-CHAPv1. New TLS-based EAP methods MUST provide protected success and failure indications 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 indications, 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 succeeded, then implementations SHOULD cause authentication to succeed for the entire session. There MAY be additional protocol exchanges which could still cause failure, so we cannot mandate sending success on successful authentication.

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 as "Value", with this specification as "Reference". For all of these labels the "DTLS-OK" field should be "N", and the "Recommended" field should be "Y".

These labels are used only for TEAP.

* EXPORTER: teap 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

[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", RFC 9190, 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-10.txt, October 2004.

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

[PEAP-PRF]
"PEAP Intermediate PEAP MAC Key (IPMK) and Compound MAC Key (CMK)" https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/0de54161-0bd3-424a-9b1a-854b4040a6df

[PEAP-TK]
"PEAP Tunnel Key (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-TTLS,v0)", 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