EAP Working Group
INTERNET-DRAFT
Category: Standards Track
<draft-ietf-eap-keying-07.txt>
17 July 2005

Bernard Aboba
Dan Simon
Microsoft
J. Arkko
Ericsson
P. Eronen
Nokia
H. Levkowetz, Ed.
ipUnplugged

Extensible Authentication Protocol (EAP) Key Management Framework

Copyright Notice

Abstract

The Extensible Authentication Protocol (EAP), defined in [RFC3748], enables extensible network access authentication. This document provides a framework for the generation, transport and usage of keying material generated by EAP authentication algorithms, known as "methods". It also specifies the EAP key hierarchy.

Table of Contents

1. Introduction

The Extensible Authentication Protocol (EAP), defined in [RFC3748], was designed to enable extensible authentication for network access in situations in which the IP protocol is not available. Originally developed for use with PPP [RFC1661], it has subsequently also been applied to IEEE 802 wired networks [IEEE-802.1X].

This document provides a framework for the generation, transport and usage of keying material generated by EAP authentication algorithms, known as "methods". In EAP keying material is generated by EAP methods. Part of this keying material may be used by EAP methods themselves and part of this material may be exported. The exported keying material may be transported by AAA protocols or transformed by Secure Association Protocols into session keys which are used by lower layer ciphersuites. This document describes each of these elements and provides a system-level security analysis. It also specifies the EAP key hierarchy.

1.1. Requirements Language

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].

1.2. Terminology

This document frequently uses the following terms:

authenticator
: The end of the link initiating EAP authentication. The term Authenticator is used in [IEEE-802.1X], and authenticator has the same meaning in this document.

peer
: The end of the link that responds to the authenticator. In [IEEE-802.1X], this end is known as the Supplicant.

Supplicant
: The end of the link that responds to the authenticator in [IEEE-802.1X]. In this document, this end of the link is called the peer.

backend authentication 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. This terminology is also used in [IEEE-802.1X].

AAA
: Authentication, Authorization and Accounting. AAA protocols with EAP support include RADIUS [RFC3579] and Diameter [I-D.ietf-aaa-eap]. In this document, the terms "AAA server" and "backend authentication server" are used interchangeably.

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.

security association
: A set of policies and cryptographic state used to protect information. Elements of a security association may include cryptographic keys, negotiated ciphersuites and other parameters, counters, sequence spaces, authorization attributes, etc.

Long Term Credential
: EAP methods frequently make use of long term secrets in order to enable authentication between the peer and server. In the case of a method based on pre-shared key authentication, the long term

credential is the pre-shared key. In the case of a public-key based method, the long term credential is the corresponding private key.

Master Session Key (MSK)
: Keying material that is derived between the EAP peer and server and exported by the EAP method. The MSK is at least 64 octets in length.

Extended Master Session Key (EMSK)
: Additional keying material derived between the peer and server that is exported by the EAP method. The EMSK is at least 64 octets in length, and is never shared with a third party.

Initialization Vector (IV)
: A quantity of at least 64 octets, suitable for use in an initialization vector field, that is derived between the peer and EAP server. Since the IV is a known value in methods such as EAP-TLS [RFC2716], it cannot be used by itself for computation of any quantity that needs to remain secret. As a result, its use has been deprecated and EAP methods are not required to generate it. However, when it is generated it MUST be unpredictable.

Pairwise Master Key (PMK)
: The AAA-Key is divided into two halves, the "Peer to Authenticator Encryption Key" (Enc-RECV-Key) and "Authenticator to Peer

Encryption Key" (Enc-SEND-Key) (reception is defined from the point of view of the authenticator). Within [IEEE-802.11i] Octets 0-31 of the AAA-Key (Enc-RECV-Key) are known as the Pairwise Master Key (PMK). In [IEEE-802.11i] the TKIP and AES CCMP ciphersuites derive their Transient Session Keys (TSKs) solely from the PMK, whereas the WEP ciphersuite as noted in [RFC3580], derives its TSKs from both halves of the AAA-Key.

Transient EAP Keys (TEKs)
: Session keys which are used to establish a protected channel between the EAP peer and server during the EAP authentication exchange. The TEKs are appropriate for use with the ciphersuite negotiated between EAP peer and server for use in protecting the EAP conversation. Note that the ciphersuite used to set up the protected channel between the EAP peer and server during EAP

authentication is unrelated to the ciphersuite used to subsequently protect data sent between the EAP peer and authenticator. An example TEK key hierarchy is described in Appendix C.

Transient Session Keys (TSKs)
: Session keys used to protect data exchanged between a port of the peer and a port of the authenticator after the EAP authentication has successfully completed. TSKs are appropriate for the lower layer ciphersuite negotiated between the ports of the EAP peer and authenticator. Examples of TSK derivation are provided in Appendix D.

AAA-Key
: A key derived by the peer and EAP server, used by the peer and authenticator in the derivation of Transient Session Keys (TSKs). Where a backend authentication server is present, the AAA-Key is transported from the backend authentication server to the authenticator, wrapped within the AAA-Token; it is therefore known by the peer, authenticator and backend authentication server. Despite the name, the AAA-Key is computed regardless of whether a backend authentication server is present. AAA-Key derivation is discussed in Section 2.5; in existing implementations the MSK is used as the AAA-Key.

AAA-Token
: Where a backend server is present, the AAA-Key and one or more attributes is transported between the backend authentication server and the authenticator within a package known as the AAA-Token. The format and wrapping of the AAA-Token, which is intended to be accessible only to the backend authentication server and authenticator, is defined by the AAA protocol. Examples include RADIUS [RFC2548] and Diameter [I-D.ietf-aaa-eap].

1.3. Overview

EAP, defined in [RFC3748] is a two-party protocol spoken between the EAP peer and server. Within EAP, keying material is generated by EAP methods. Part of this keying material may be used by EAP methods themselves and part of this material may be exported. In addition to export of keying material, EAP methods may also export associated parameters, and may import and export Channel Bindings from the lower

layer.

As illustrated in Figure 1, the EAP method key derivation has at the root the long term credential utilized by the selected EAP method. If authentication is based on a pre-shared key, the parties store the EAP method to be used and the pre-shared key. The EAP server also stores the peer's identity and/or other information necessary to decide whether access to some service should be granted. The peer stores information necessary to choose which secret to use for which service.

If authentication is based on proof of possession of the private key corresponding to the public key contained within a certificate, the parties store the EAP method to be used and the trust anchors used to validate the certificates. The EAP server also stores the peer's identity and/or other information necessary to decide whether access to some service should be granted. The peer stores information necessary to choose which certificate to use for which service.

Based on the long term credential established between the peer and the server, EAP methods derive two types of keys:

[1] Keys calculated locally by the EAP method but not exported by the EAP method, such as the TEKs.
[2] Keying material exported by the EAP method: MSK, EMSK, IV.

As noted in [RFC3748] Section 7.10, EAP methods generating keys are required to calculate and export the MSK and EMSK, which must be at least 64 octets in length. EAP methods also may export the IV; however, the use of the IV is deprecated.

+-+ ---+

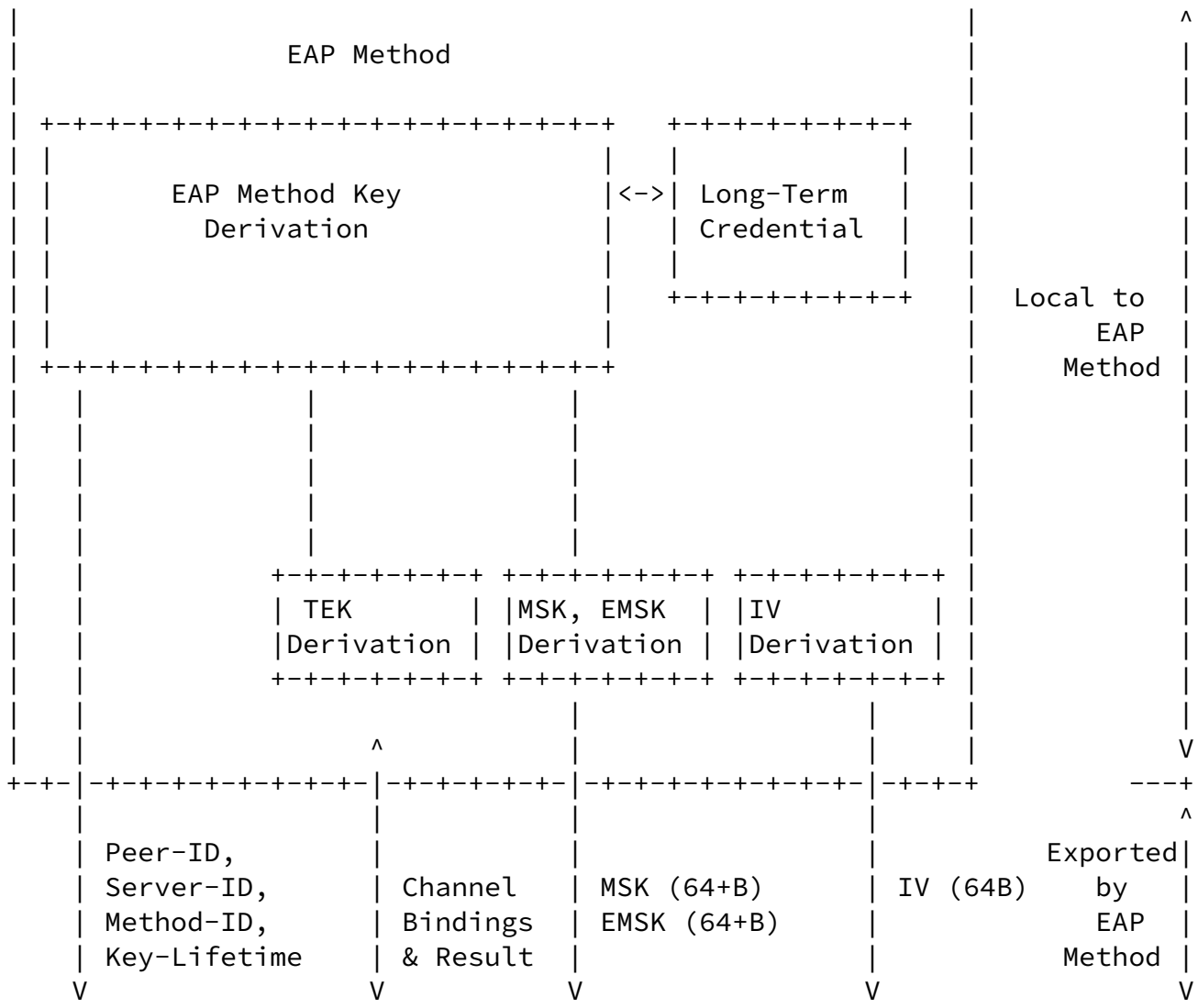


Figure 1: EAP Parameter Import/Export

EAP methods also MAY export method-specific peer and server identifiers (peer-ID and server-ID), a method-specific EAP conversation identifier known as the Method-ID, and the lifetime of the exported keys, known the Key-Lifetime. EAP methods MAY also support the import and export of Channel Bindings. New EAP method specifications MUST define the Peer-ID, Server-ID and Method-ID. The combination of the Peer-ID and Server-ID uniquely specifies the endpoints of the EAP method exchange.

Peer-ID

As described in [RFC3748] Section 7.3, the identity provided in the EAP-Response/Identity, may be different from the peer identity authenticated by the EAP method. Where the EAP method authenticates the peer identity, that identity is exported by the

method as the Peer-ID. A suitable EAP peer name may not always be available. Where an EAP method does not define a method-specific peer identity, the Peer-ID is the null string. The Peer-ID for existing EAP methods is defined in Appendix E.

Server-ID

Where the EAP method authenticates the server identity, that identity is exported by the method as the Server-ID. A suitable EAP server name may not always be available. Where an EAP method does not define a method-specific peer identity, the Server-ID is the null string. The Server-ID for existing EAP methods is defined in Appendix E.

Method-ID

EAP method specifications deriving keys MUST specify a temporally unique method identifier known as the Method-ID. The EAP Method-ID uniquely identifies an EAP session of a given Type between an EAP peer and server. The Method-ID is typically constructed from nonces or counters used within the EAP method exchange. The Method-ID for existing EAP methods is defined in Appendix E.

Session-ID

The Session-ID uniquely identifies an EAP session between an EAP peer (as identified by the Peer-ID) and server (as identified by the Server-ID). The EAP Session-ID consists of the concatenation of the Expanded EAP Type Code (including the Type, Vendor-ID and Vendor-Type fields defined in [RFC3748] Section 5.7) and the Method-ID. The inclusion of the Expanded Type Code in the EAP Session-Id ensures that each EAP method has a distinct Session-ID space. Since an EAP session is not bound to a particular authenticator or specific ports on the peer and authenticator, the authenticator port or identity are not included in the Session-Id.

Key-Lifetime

While EAP itself does not support key lifetime negotiation, it is possible to specify methods that do. However, systems that rely on such negotiation for exported keys would only function with these methods. As a result, it is NOT RECOMMENDED to use this approach as the sole way to determine key lifetimes.

Channel Bindings

Channel Bindings include lower layer parameters that are verified for consistency between the EAP peer and server. In order to

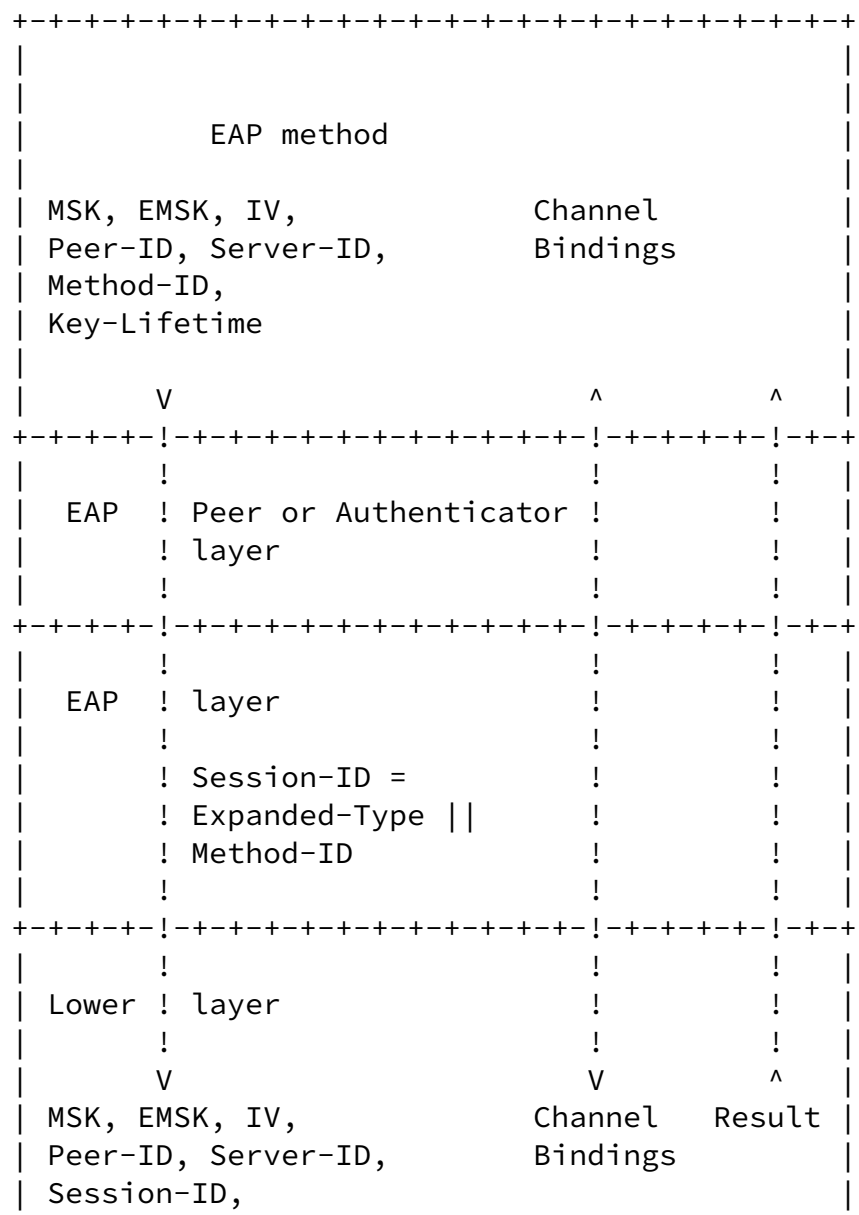
avoid introducing media dependencies, EAP methods MUST treat Channel Bindings as opaque octets. Typically the EAP method imports Channel Bindings from the lower layer on the peer, and transmits them securely to the EAP server, which exports them to the lower layer. However, transport may occur from EAP server to peer, or may be bi-directional. On the side of the exchange (peer or server) where Channel Bindings are verified, the lower layer passes the result of the verification (TRUE or FALSE) up to the EAP method.

1.3.1. Layering

As illustrated in Figure 2, keying material and parameters exported by EAP methods are passed down to the EAP peer or authenticator layers, which passes them to the EAP layer. Keying material and related parameters (including Channel Bindings) MUST NOT be cached by the EAP peer or authenticator layers, or the EAP layer.

Based on the Method-ID exported by the EAP method, the EAP layer forms the EAP Session-ID by concatenating the EAP Expanded Type with the Method-ID. Together with the MSK, EMSK, IV, Peer-ID, Server-ID, and Key-Lifetime, the EAP layer passes the Session-ID down to the lower layer.

The Method-ID is exported by EAP methods rather than the Session-ID so as to prevent EAP methods from writing into each other's Session-ID space. Lower layers MAY cache keying material and related parameters, including Channel Bindings. Lower Layer behavior is discussed in more detail in Section 2.


EAP method
MSK, EMSK, IV,
Peer-ID, Server-ID,
Method-ID,
Key-Lifetime
Channel
Bindings
EAP
Peer or Authenticator
layer
EAP
layer
Session-ID =
Expanded-Type ||
Method-ID
Lower
layer
MSK, EMSK, IV,
Peer-ID, Server-ID,
Session-ID,
Channel
Bindings
Result

```
   | Key-Lifetime                                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Flow of EAP parameters

1.3.2. Key Naming

Each key created within the EAP key management framework has a name (the identifier by which the key can be identified), as well as a scope (the parties to whom the key is available). The scope of exported parameters is defined by the EAP peer name (if securely exchanged within the method) and the EAP server name (also only if securely exchanged). Where a peer or server name is missing the null string is used.

MSK Name
: This key is created between the EAP peer and EAP server, and can be referred to using the string "MSK:", concatenated with the EAP Session-ID.

EMSK Name
: The EMSK can be referred to using the string "EMSK:", concatenated with the EAP Session-ID.

IV Name
: Use of the IV is deprecated. However, if necessary the IV can be referred to using the string "IV:" concatenated with the EAP Session-ID.

PMK Name
: This document does not specify a naming scheme for the PMK. The PMK is only identified by the key from which it is derived.

  Note: IEEE 802.11i names the PMKID for the purposes of being able to refer to it in the Secure Association protocol; this naming is based on a hash of the PMK itself as well as some other parameters (see Section 8.5.1.2 [IEEE-802.11i]).

TEK Name
: The TEKs may or may not be named. Their naming is specified in the

EAP method.

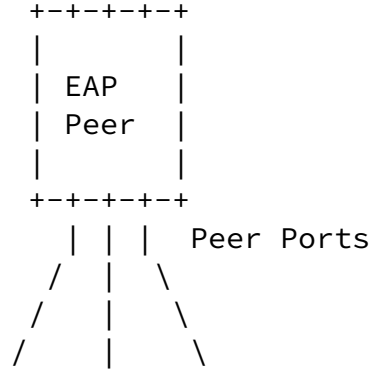
1.3.3. EAP and AAA

EAP is typically deployed in order to support extensible network access authentication in situations where a peer desires network access via one or more authenticators. Since both the peer and authenticator may have more than one physical or logical port, a given peer may simultaneously access the network via multiple authenticators, or via multiple physical or logical ports on a given authenticator. Similarly, an authenticator may offer network access to multiple peers, each via a separate physical or logical port. The situation is illustrated in Figure 3.

Where authenticators are deployed standalone, the EAP conversation occurs between the peer and authenticator, and the authenticator must locally implement an EAP method acceptable to the peer. However, one of the advantages of EAP is that it enables deployment of new authentication methods without requiring development of new code on the authenticator. While the authenticator may implement some EAP methods locally and use those methods to authenticate local users, it may at the same time act as a pass-through for other users and methods, forwarding EAP packets back and forth between the backend

authentication server and the peer.

This is accomplished by encapsulating EAP packets within the Authentication, Authorization and Accounting (AAA) protocol, spoken between the authenticator and backend authentication server. AAA protocols supporting EAP include RADIUS [RFC3579] and Diameter [I-D.ietf-aaa-eap].



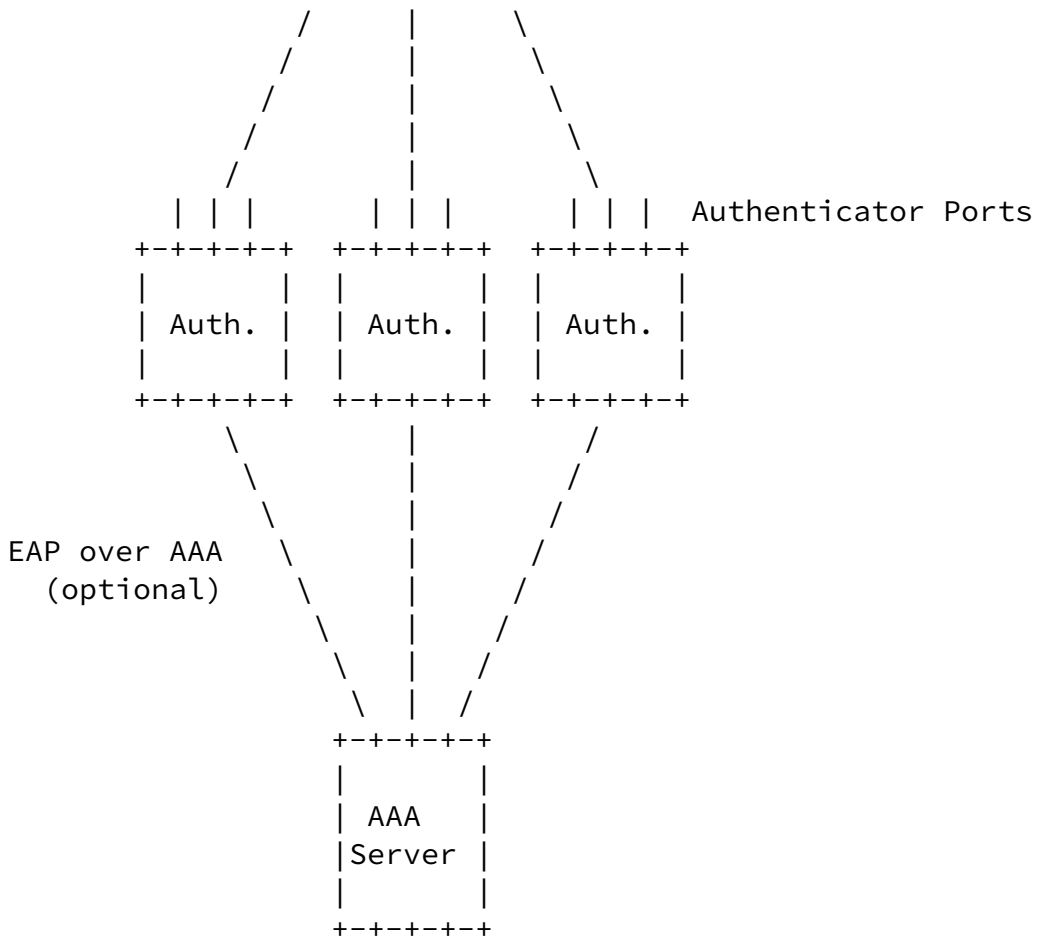


Figure 3: Relationship between peer, authenticator and backend server

1.4. EAP Invariants

Certain basic characteristics, known as "EAP Invariants", hold true for EAP implementations on all media:

Mode independence
Media independence
Method independence
Ciphersuite independence

1.4.1. Mode Independence

EAP as defined in [RFC3748] is a two party protocol spoken between the EAP peer and server. A fundamental property of EAP is that at the EAP method layer, the conversation between the EAP peer and server is unaffected by whether the EAP authenticator is operating in "pass-through" mode.

EAP methods operate identically in all aspects, including key derivation and parameter import/export, regardless of whether the authenticator is operating as a pass-through or not.

The successful completion of an EAP method that supports key derivation results in the export of keying material on the EAP peer and server. Even though the EAP peer or server may import Channel-Bindings that may include the identity of the EAP authenticator, this information is treated as opaque octets. As a result, within EAP the only relevant identities are the Peer-ID and Server-ID. Channel Bindings are only interpreted by the lower layer.

Within EAP, the primary function of the AAA protocol is to maintain the principle of Mode Independence, so that as far as the EAP peer is concerned, its conversation with the EAP authenticator, and all consequences of that conversation, are identical, regardless of the authenticator mode of operation.

1.4.2. Media Independence

One of the goals of EAP is to allow EAP methods to function on any lower layer meeting the criteria outlined in [RFC3748], Section 3.1. For example, as described in [RFC3748], EAP authentication can be run over PPP [RFC1661], IEEE 802 wired networks [IEEE-802.1X], and IEEE 802.11 wireless LANs [IEEE-802.11i].

In order to maintain media independence, it is necessary for EAP to avoid consideration of media-specific elements. For example, EAP methods cannot be assumed to have knowledge of the lower layer over which they are transported, and cannot be restricted to identifiers

associated with a particular usage environment (e.g. MAC addresses).

Note that media independence may be retained within EAP methods that support Channel-Bindings or method-specific identification. An EAP

method need not be aware of the content of an identifier in order to use it. This enables an EAP method to use media-specific identifiers such as MAC addresses without compromising media independence. Channel-Bindings are treated as opaque octets by EAP methods, so that handling them does not require media-specific knowledge.

1.4.3. Method Independence

By enabling pass-through, authenticators can support any method implemented on the peer and server, not just locally implemented methods. This allows the authenticator to avoid implementing code for each EAP method required by peers. In fact, since a pass-through authenticator is not required to implement any EAP methods at all, it cannot be assumed to support any EAP method-specific code.

As a result, as noted in [RFC3748], authenticators must by default be capable of supporting any EAP method. This is useful where there is no single EAP method that is both mandatory-to-implement and offers acceptable security for the media in use. For example, the [RFC3748] mandatory-to-implement EAP method (MD5-Challenge) does not provide dictionary attack resistance, mutual authentication or key derivation, and as a result is not appropriate for use in wireless LAN authentication [RFC4017]. However, despite this it is possible for the peer and authenticator to interoperate as long as a suitable EAP method is supported on the EAP server.

1.4.4. Ciphersuite Independence

Ciphersuite Independence is a consequence of the principles of Mode Independence and Media Independence.

While EAP methods may negotiate the ciphersuite used in protection of the EAP conversation, the ciphersuite used for the protection of the data exchanged after EAP authentication has completed is negotiated between the peer and authenticator within the lower layer, outside of EAP. Since the ciphersuites used to protect data depend on the lower layer, requiring EAP methods have knowledge of lower layer ciphersuites would compromise the principle of Media Indepence.

Since ciphersuite negotiation occurs in the lower layer, there is no need for ciphersuite negotiation within EAP, and EAP methods generate keying material that is ciphersuite-independent.

For example, within PPP, the ciphersuite is negotiated within the

Encryption Control Protocol (ECP) defined in [RFC1968], after EAP authentication is completed. Within [IEEE-802.11i], the AP ciphersuites are advertised in the Beacon and Probe Responses prior to EAP authentication, and are securely verified during a 4-way handshake exchange.

Advantages of ciphersuite-independence include:

Reduced update requirements
: If EAP methods were to specify how to derive transient session keys for each ciphersuite, they would need to be updated each time a new ciphersuite is developed. In addition, backend authentication servers might not be usable with all EAP-capable authenticators, since the backend authentication server would also need to be updated each time support for a new ciphersuite is added to the authenticator.

Reduced EAP method complexity
: Requiring each EAP method to include ciphersuite-specific code for transient session key derivation would increase method complexity and result in duplicated effort.

Simplified configuration
: The ciphersuite is negotiated between the peer and authenticator outside of EAP. Where the authenticator operates in "pass-through" mode, the EAP server is not a party to this negotiation, nor is it involved in the data flow between the EAP peer and authenticator. As a result, the EAP server may not have knowledge of the ciphersuites and negotiation policies implemented by the peer and authenticator, or be aware of the ciphersuite negotiated between them. For example, since ECP negotiation occurs after authentication, when run over PPP, the EAP peer and server may not anticipate the negotiated ciphersuite and therefore this information cannot be provided to the EAP method.

2. Lower Layer Operation

Where EAP key derivation is supported, the conversation typically takes place in three phases:

```
Phase 0: Discovery
Phase 1: Authentication
         1a: EAP authentication
         1b: AAA-Key Transport (optional)
Phase 2: Secure Association Establishment
         2a: Unicast Secure Association
         2b: Multicast Secure Association (optional)
```

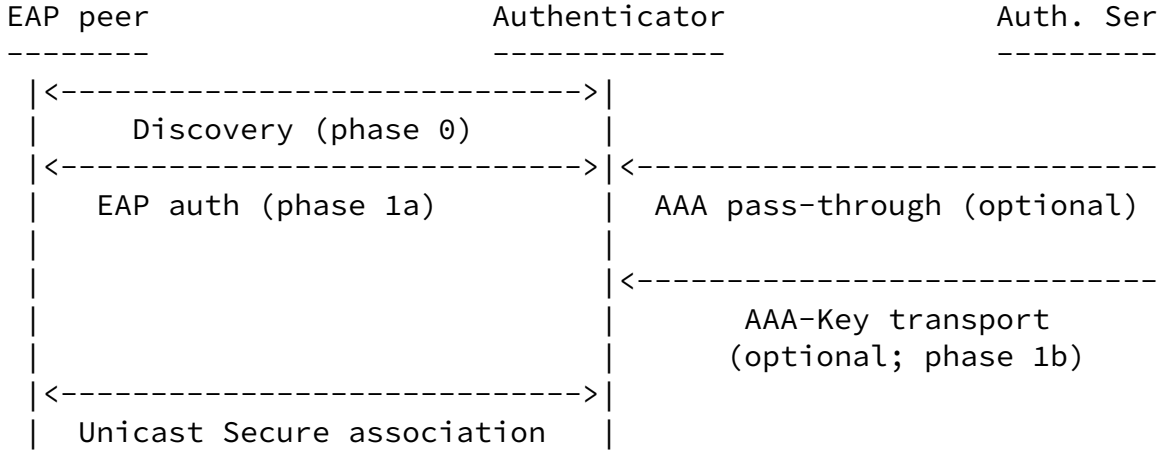
Of these phases, Phase 0, 1b and Phase 2 are handled by a lower layer. In the discovery phase (phase 0), peers locate authenticators and discover their capabilities. For example, a peer may locate an authenticator providing access to a particular network, or a peer may locate an authenticator behind a bridge with which it desires to establish a Secure Association.

The authentication phase (phase 1) may begin once the peer and authenticator discover each other. This phase always includes EAP authentication (phase 1a). Where the chosen EAP method supports key derivation, in phase 1a keying material is derived on both the peer and the EAP server. This keying material may be used for multiple purposes, including protection of the EAP conversation and subsequent data exchanges.

An additional step (phase 1b) is required in deployments which include a backend authentication server, in order to transport keying material (known as the AAA-Key) from the backend authentication server to the authenticator.

A Secure Association exchange (phase 2) then occurs between the peer and authenticator in order to manage the creation and deletion of unicast (phase 2a) and multicast (phase 2b) security associations between the peer and authenticator.

The conversation phases and relationship between the parties is shown in Figure 4.



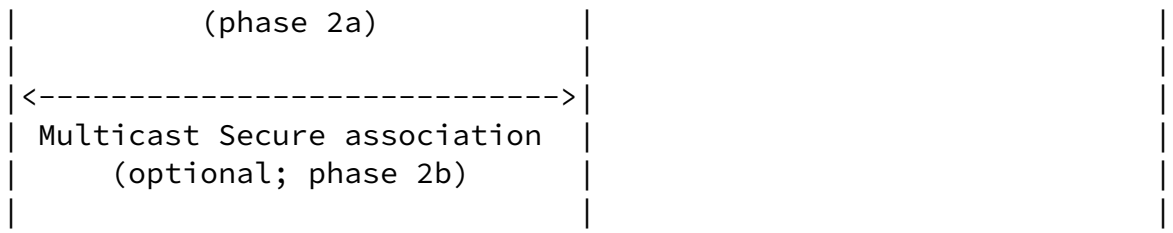


Figure 4: Conversation Overview

2.1. Discovery Phase

In the discovery phase (phase 0), the EAP peer and authenticator locate each other and discover each other's capabilities. Discovery can occur manually or automatically, depending on the lower layer over which EAP runs. Since authenticator discovery is handled outside of EAP, there is no need to provide this functionality within EAP.

For example, where EAP runs over PPP, the EAP peer might be configured with a phone book providing phone numbers of authenticators and associated capabilities such as supported rates, authentication protocols or ciphersuites. In contrast, PPPoE [RFC2516] provides support for a Discovery Stage to allow a peer to identify the Ethernet MAC address of one or more authenticators and establish a PPPoE SESSION_ID.

IEEE 802.11 [IEEE-802.11] also provides integrated discovery support utilizing Beacon and/or Probe Request/Response frames, allowing the peer (known as the station or STA) to determine the MAC address and capabilities of one or more authenticators (known as Access Point or APs).

2.2. Authentication Phase

Once the peer and authenticator discover each other, they exchange EAP packets. Typically, the peer desires access to the network, and the authenticators provide that access. In such a situation, access to the network can be provided by any authenticator attaching to the desired network, and the EAP peer is typically willing to send data traffic through any authenticator that can demonstrate that it is authorized to provide access to the desired network.

An EAP authenticator may handle the authentication locally, or it may act as a pass-through to a backend authentication server. In the latter case the EAP exchange occurs between the EAP peer and a backend authenticator server, with the authenticator forwarding EAP packets between the two. The entity which terminates EAP authentication with the peer is known as the EAP server. Where pass-through is supported, the backend authentication server functions as the EAP server; where authentication occurs locally, the EAP server is the authenticator. Where a backend authentication server is present, at the successful completion of an authentication exchange, the AAA-Key is transported to the authenticator (phase 1b).

EAP may also be used when it is desired for two network devices (e.g. two switches or routers) to authenticate each other, or where two peers desire to authenticate each other and set up a secure

association suitable for protecting data traffic.

Some EAP methods exist which only support one-way authentication; however, EAP methods deriving keys are required to support mutual authentication. In either case, it can be assumed that the parties do not utilize the link to exchange data traffic unless their authentication requirements have been met. For example, a peer completing mutual authentication with an EAP server will not send data traffic over the link until the EAP server has authenticated successfully to the peer, and a Secure Association has been negotiated.

Since EAP is a peer-to-peer protocol, an independent and simultaneous authentication may take place in the reverse direction. Both peers may act as authenticators and authenticatees at the same time.

Successful completion of EAP authentication and key derivation by a peer and EAP server does not necessarily imply that the peer is committed to joining the network associated with an EAP server. Rather, this commitment is implied by the creation of a security association between the EAP peer and authenticator, as part of the Secure Association Protocol (phase 2). As a result, EAP may be used for "pre-authentication" in situations where it is necessary to pre-establish EAP security associations in order to decrease handoff or roaming latency.

2.3. Secure Association Phase

The Secure Association phase (phase 2), if it occurs, begins after the completion of EAP authentication (phase 1a) and key transport (phase 1b). A Secure Association Protocol used with EAP typically supports the following features:

[1] Generation of fresh transient session keys (TSKs). Where AAA-Key caching is supported, the EAP peer may initiate a new session using a AAA-Key that was used in a previous session. Were the TSKs to be derived from a portion of the AAA-Key, this would result in reuse of the session keys which could expose the underlying ciphersuite to attack.

As a result, where AAA-Key caching is supported, the Secure Association Protocol phase is REQUIRED, and MUST provide for freshness of the TSKs. This is typically handled via the exchange of nonces or counters, which are then mixed with the AAA-Key in order to generate fresh unicast (phase 2a) and possibly multicast (phase 2b) session keys. By not using the AAA-Key directly to protect data, the Secure Association Protocol protects against compromise of the AAA-Key.

[2] Entity Naming. A basic feature of a Secure Association Protocol is the explicit naming of the parties engaged in the exchange. Explicit identification of the parties is critical, since without this the parties engaged in the exchange are not identified and the scope of the transient session keys (TSKs) generated during the exchange is undefined. As illustrated in Figure 3, both the peer and NAS may have more than one physical or virtual port, so that port identifiers are NOT RECOMMENDED as a naming mechanism.

[3] Secure capabilities negotiation. This includes the secure negotiation of usage modes, session parameters (such as key lifetimes), ciphersuites and required filters, including confirmation of the capabilities discovered during phase 0. It is RECOMMENDED that the Secure Association Protocol support secure capabilities negotiation, in order to protect against spoofing during the discovery phase, and to ensure agreement between the peer and authenticator about how data is to be secured.

[4] Key management. EAP as defined in [RFC3748] supports key

derivation, but not key management. While EAP methods may derive keying material, EAP does provide for the management of exported or derived keys. For example, EAP does not support negotiation of the key lifetime of exported or derived keys, nor does it support rekey. Although EAP methods may support "fast reconnect" as defined in [RFC3748] Section 7.2.1, rekey of exported keys cannot occur without reauthentication. In order to provide method independence, key management of exported or derived keys SHOULD NOT be provided within EAP methods.

Since neither EAP nor EAP methods provide key management support, it is RECOMMENDED that key management facilities be provided within the Secure Association Protocol. This includes key lifetime management (such as via explicit key lifetime negotiation, or seamless rekey), as well synchronization of the installation and deletion of keys so as to enable recovery from partial or complete loss of key state by the peer or authenticator. Since key management requires a key naming scheme, Secure Association Protocols supporting key management support MUST also support key naming.

[5] Mutual proof of possession of the AAA-Key. The Secure Association Protocol MUST demonstrate mutual proof of posession of the AAA-Key, in order to show that both the peer and authenticator have been authenticated and authorized by the backend authentication server. Since mutual proof of possession is not the same as mutual authentication, the peer cannot verify authenticator assertions (including the authenticator identity) as a result of this exchange.

2.4. Lower Layer Key Hierarchy

From the keys exported by the EAP method, two other types of keys may be derived:

[3] Keys calculated from exported quantities: AAA-Key.
[4] Keys calculated by the Secure Association Protocol from the AAA-Key: TSKs.

In order to protect the EAP conversation, methods supporting key derivation typically negotiate a ciphersuite and derive Transient EAP Keys (TEKs) for use with that ciphersuite. The TEKs are stored

locally by the EAP method and are not exported.

As noted in [RFC3748] Section 7.10, EAP methods generating keys are required to calculate and export the MSK and EMSK, which must be at least 64 octets in length. EAP methods also may export the IV; however, the use of the IV is deprecated. On both the peer and EAP server, the exported MSK is utilized in order to calculate the AAA-Key. Where a backend authentication server is present, the AAA-Key is transported from the backend authentication server to the authenticator within the AAA-Token, using the AAA protocol.

Once EAP authentication completes and is successful, the peer and authenticator obtain the AAA-Key and the Secure Association Protocol is run between the peer and authenticator in order to securely negotiate the ciphersuite, derive fresh TSKs used to protect data, and provide mutual proof of possession of the AAA-Key.

When the authenticator acts as an endpoint of the EAP conversation rather than a pass-through, EAP methods are implemented on the authenticator as well as the peer. If the EAP method negotiated between the EAP peer and authenticator supports mutual authentication and key derivation, the EAP Master Session Key (MSK) and Extended Master Session Key (EMSK) are derived on the EAP peer and authenticator and exported by the EAP method. In this case, the MSK and EMSK are known only to the peer and authenticator and no other parties. The TEKs and TSKs also reside solely on the peer and authenticator. This is illustrated in Figure 6. As demonstrated in [I-D.ietf-roamops-cert], in this case it is still possible to support roaming between providers, using certificate-based authentication.

Where a backend authentication server is utilized, the situation is illustrated in Figure 7. Here the authenticator acts as a pass-through between the EAP peer and a backend authentication server. In this model, the authenticator delegates the access control decision to the backend authentication server, which acts as a Key Distribution Center (KDC). In this case, the authenticator

encapsulates EAP packet with a AAA protocol such as RADIUS [RFC3579] or Diameter [I-D.ietf-aaa-eap], and forwards packets to and from the backend authentication server, which acts as the EAP server. Since the authenticator acts as a pass-through, EAP methods reside only on the peer and EAP server As a result, the TEKs, MSK and EMSK are

derived on the peer and EAP server.

On completion of EAP authentication, EAP methods on the peer and EAP server export the Master Session Key (MSK) and Extended Master Session Key (EMSK). The peer and EAP server then calculate the AAA-Key from the MSK and EMSK, and the backend authentication server sends an Access-Accept to the authenticator, providing the AAA-Key within a protected package known as the AAA-Token.

The AAA-Key is then used by the peer and authenticator within the Secure Association Protocol to derive Transient Session Keys (TSKs) required for the negotiated ciphersuite. The TSKs are known only to the peer and authenticator.

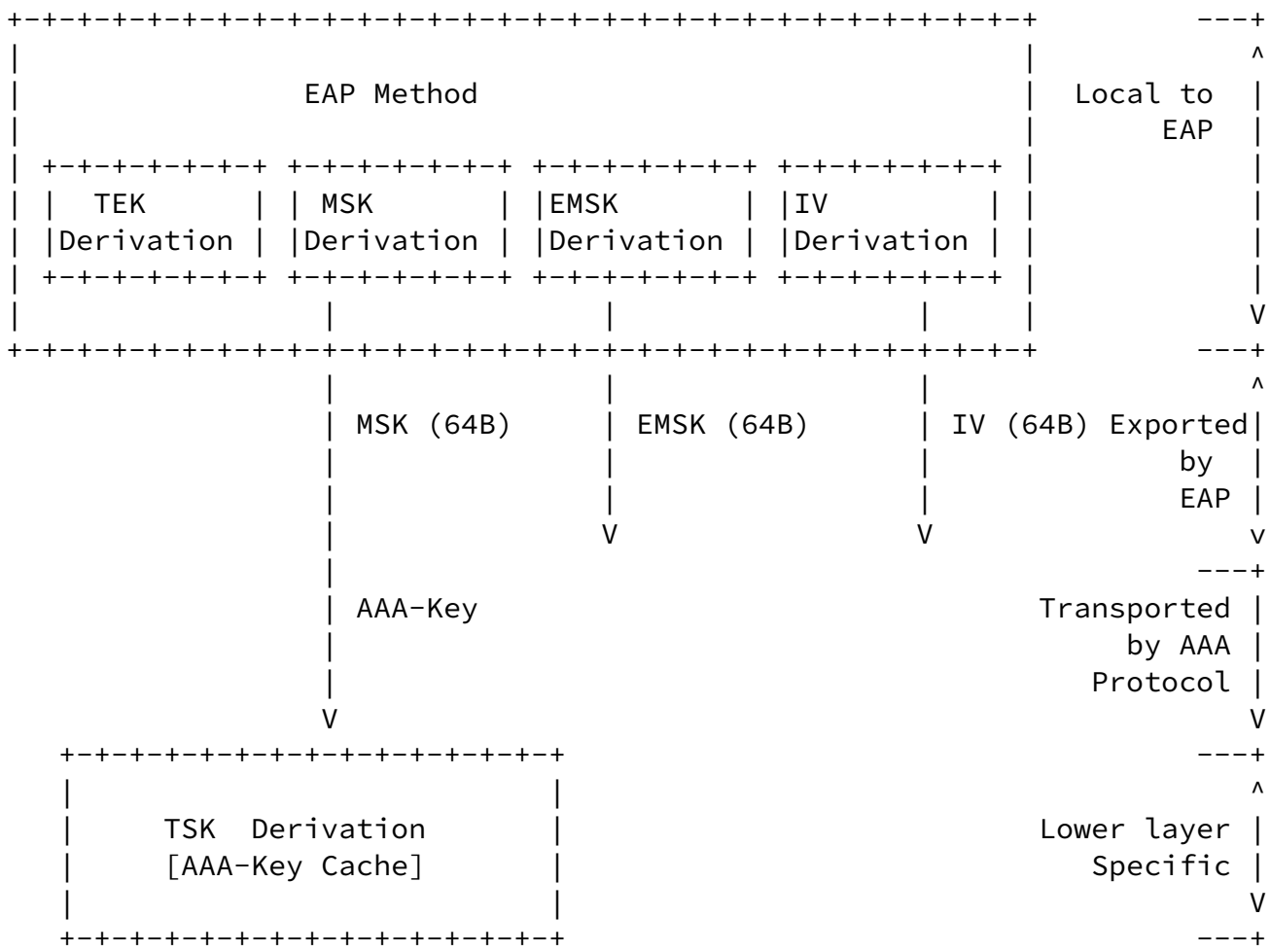


Figure 5: Complete Key Hierarchy

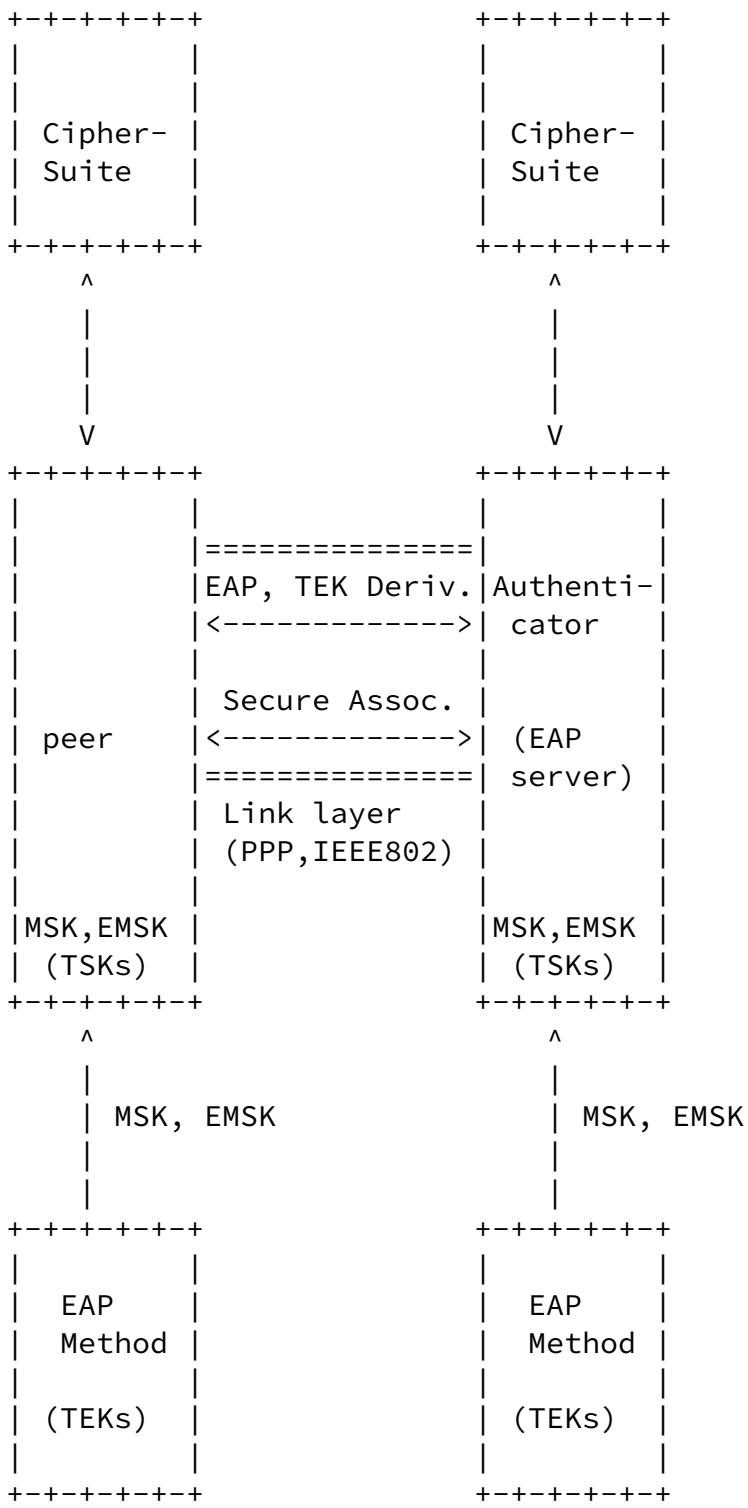


Figure 6: Relationship between EAP peer and authenticator (acting as an EAP server), where no backend authentication server is present.

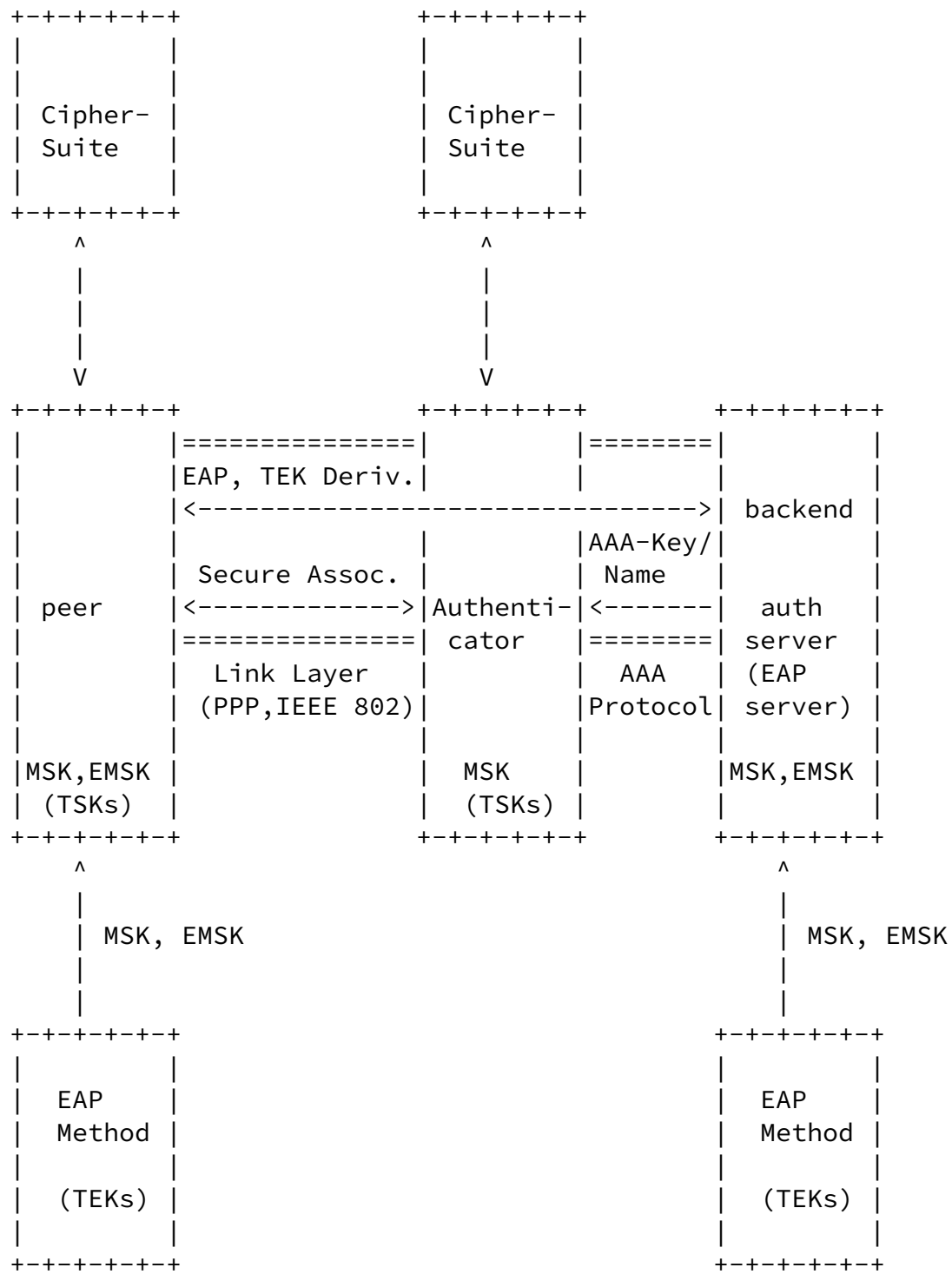


Figure 7: Pass-through relationship between EAP peer, authenticator

and backend authentication server.

2.5. AAA-Key Derivation and Naming

In existing usage, where a AAA-Key is generated as the result of a successful EAP authentication with the authenticator, the AAA-Key is

based on the MSK: AAA-Key = MSK(0,63).

In existing usage, the AAA-Key is always derived from the MSK so can be referred to using the MSK name.

The AAA-Key scope is provided by the concatenation of the EAP peer name (if securely provided to the authenticator), and the authenticator name (if securely provided to the peer).

For the purpose of identifying the authenticator to the peer, the value of the NAS-Identifier attribute is recommended. The authenticator may include the NAS-Identifier attribute to the AAA server in an Access-Request, and the authenticator may provide the NAS-Identifier to the EAP peer. Mechanisms for this include use of the EAP-Request/Identity (unsecured) or a lower layer mechanism (such as the 802.11 Beacon/Probe Response). Where the NAS-Identifier is provided by the authenticator to the peer a secure mechanism is RECOMMENDED.

For the purpose of identifying the peer to the authenticator, the EAP peer identifier provided within the EAP method is recommended. It cannot be assumed that the authenticator is aware of the EAP peer name used within the method. Therefore alternatives mechanisms need to be used to provide the EAP peer name to the authenticator. For example, the AAA server may include the EAP peer name in the User-Name attribute of the Access-Accept or the peer may provide the authenticator with its name via a lower layer mechanism.

Absent an explicit binding step within the Secure Association Protocol, the AAA-Key is not bound to a specific peer or authenticator port. As a result, the peer or authenticator port over which the EAP conversation takes place is not included in the AAA-Key scope.

2.5.1. TSKs

The TSKs are typically named. Their naming is specified in the Secure Association (phase 2) protocol, so that the correct set of transient session keys can be identified for processing a given packet. The scope of the TSKs is negotiated within the Secure Association Protocol.

TSK creation and deletion operations are typically supported so that establishment and re-establishment of TSKs can be synchronized between the parties.

In order to avoid confusion in the case where an EAP peer has more than one AAA-Key (phase 1b) applicable to establishment of a phase 2

security association, the secure Association protocol needs to utilize the AAA-Key name so that the appropriate phase 1b keying material can be identified for use in the Secure Association Protocol exchange.

3. Security Associations

During EAP authentication and subsequent exchanges, four types of security associations (SAs) are created:

[1] EAP method SA. This SA is between the peer and EAP server. It stores state that can be used for "fast reconnect" or other functionality in some EAP methods. Not all EAP methods create such an SA.

[2] EAP-Key SA. This is an SA between the peer and EAP server, which is used to store the keying material exported by the EAP method. Current EAP server implementations do not retain this SA after the EAP conversation completes.

[3] AAA SA(s). These SAs are between the authenticator and the backend authentication server. They permit the parties to mutually authenticate each other and protect the communications between them.

[4] Service SA(s). These SAs are between the peer and authenticator, and they are created as a result of phases 1-2 of the conversation (see Section 2).

Examples of security associations are provided in Appendix F.

3.1. EAP Method SA (peer - EAP server)

An EAP method may store some state on the peer and EAP server even after phase 1a has completed.

Typically, this is used for "fast reconnect": the peer and EAP server can confirm that they are still talking to the same party, perhaps using fewer round-trips or less computational power. In this case, the EAP method SA is essentially a cache for performance optimization, and either party may remove the SA from its cache at any point.

An EAP method may also keep state in order to support pseudonym-based identity protection. This is typically a cache as well (the information can be recreated if the original EAP method SA is lost), but may be stored for longer periods of time.

The EAP method SA is not restricted to a particular service or authenticator and is most useful when the peer accesses many different authenticators. An EAP method is responsible for specifying how the parties select if an existing EAP method SA should be used, and if so, which one. Where multiple backend authentication servers are used, EAP method SAs are not typically synchronized between them.

EAP method implementations should consider the appropriate lifetime for the EAP method SA. "Fast reconnect" assumes that the information required (primarily the keys in the EAP method SA) hasn't been compromised. In case the original authentication was carried out using, for instance, a smart card, it may be easier to compromise the EAP method SA (stored on the PC, for instance), so typically the EAP method SAs have a limited lifetime.

Contents:

- Implicitly, the EAP method this SA refers to
- Internal (non-exported) cryptographic state
- EAP method SA name

- SA lifetime

3.2. EAP-Key SA

This is an SA between the peer and EAP server, which is used to store the keying material exported by the EAP method. Current EAP server implementations do not retain this SA after the EAP conversation completes. As a result, all keys exported by the EAP method (including the MSK, EMSK and IV) on the AAA server are discarded and are not cached. Calculated keys (such as the AAA-Key) are also discarded and not cached.

3.3. AAA SA(s) (authenticator - backend authentication server)

In order for the authenticator and backend authentication server to authenticate each other, they need to store some information.

In case the authenticator and backend authentication server are colocated, and they communicate using local procedure calls or shared memory, this SA need not necessarily contain any information.

3.4. Service SA(s) (peer - authenticator)

The service SAs store information about the service being provided. These include the Root service SA and derived unicast and multicast service SAs.

The Root service SA is established as the result of the completion of EAP authentication (phase 1a) and AAA-Key derivation or transport (phase 1b). It includes:

- Service parameters (or at least those parameters that are still needed)
- On the authenticator, service authorization information received from the backend authentication server (or necessary parts of it)
- On the peer, usually locally configured service authorization information.
- The AAA-Key, if it can be needed again (to refresh and/or resynchronize other keys or for another reason)
- AAA-Key lifetime

Unicast and (optionally) multicast service SAs are derived from the Root service SA, via the Secure Association Protocol. In order for unicast and multicast service SAs and associated TSKs to be established, it is not necessary for EAP authentication (phase 1a) to be rerun each time. Instead, the Secure Association Protocol can be used to mutually prove possession of the AAA-Key and create associated unicast (phase 2a) and multicast (phase 2b) service SAs and TSKs, enabling the EAP exchange to be bypassed. Unicast and multicast service SAs include:

- Service parameters negotiated by the Secure Association Protocol.
- Endpoint identifiers.
- Transient Session Keys used to protect the communication.
- Transient Session Key lifetime.

One function of the Secure Association Protocol is to bind the the unicast and multicast service SAs and TSKs to endpoint identifiers. For example, within [IEEE802.11i], the 4-way handshake binds the TSKs to the MAC addresses of the endpoints; in [IKEv2], the TSKs are bound to the IP addresses of the endpoints and the negotiated SPI.

It is possible for more than one unicast or multicast service SA to be derived from a single Root service SA. However, a unicast or multicast service SA is always descended from only one Root service SA. Unicast or multicast service SAs descended from the same Root service SA may utilize the same security parameters (e.g. mode, ciphersuite, etc.) or they may utilize different parameters.

An EAP peer may be able to negotiate multiple service SAs with a given authenticator, or may be able to maintain one or more service SAs with multiple authenticators, depending on the properties of the media.

Except where explicitly specified by the Secure Association Protocol, it should not be assumed that the installation of new service SAs implies deletion of old service SAs. It is possible for multicast Root service SAs to between the same EAP peer and authenticator; during a re-key of a unicast or multicast service SA it is possible for two service SAs to exist during the period between when the new service SA and corresponding TSKs are calculated and when they are

installed.

Similarly, deletion or creation of a unicast or multicast service SA does not necessarily imply deletion or creation of related unicast or multicast service SAs, unless specified by the Secure Association protocol. For example, a unicast service SA may be rekeyed without implying a rekey of the multicast service SA.

The deletion of the Root service SA does not necessarily imply the deletion of the derived unicast and multicast service SAs and associated TSKs. Failure to mutually prove possession of the AAA-Key during the Secure Association Protocol exchange need not be grounds for deletion of the AAA-Key by both parties; the action to be taken is defined by the Secure Association Protocol.

3.4.1. Sharing service SAs

A single service may be provided by multiple logical or physical service elements. Each service is responsible for specifying how changing service elements is handled. Some approaches include:

Transparent sharing
: If the service parameters visible to the other party (either peer or authenticator) do not change, the service can be moved without requiring cooperation from the other party.

  Whether such a move should be supported or used depends on implementation and administrative considerations. For instance, an administrator may decide to configure a group of IKEv2/IPsec gateways in a cluster for high-availability purposes, if the implementation used supports this. The peer does not necessarily have any way of knowing when the change occurs.

No sharing
: If the service parameters require changing, some changes may require terminating the old service, and starting a new conversation from phase 0. This approach is used by all services for at least some parameters, and it doesn't require any protocol for transferring the service SA between the service elements.

  The service may support keeping the old service element active

while the new conversation takes phase, to decrease the time the service is not available.

Some sharing
: The service may allow changing some parameters by simply agreeing about the new values. This may involve a similar exchange as in phase 2, or perhaps a shorter conversation.

  This option usually requires some protocol for transferring the service SA between the elements. An administrator may decide not to enable this feature at all, and typically the sharing is restricted to some particular service elements (defined either by a service parameter, or simple administrative decision). If the old and new service element do not support such "context transfer", this approach falls back to the previous option (no transfer).

  Services supporting this feature should also consider what changes require new authorization from the backend authentication server (see Section 5.2).

  Note that these considerations are not limited to service parameters related to the authenticator--they apply to peer parameters as well.

4. Key Management

EAP supports key derivation, but not key management. As a result, key management functionality needs to be provided by the Secure Association Protocol. This includes:

[a] Generation of fresh transient session keys (TSKs). Where AAA-Key caching is supported, the EAP peer may initiate a new session using a AAA-Key that was used in a previous session. Were the TSKs to be derived from a portion of the AAA-Key, this would result in reuse of the session keys which could expose the underlying ciphersuite to attack. As a result, where AAA-Key caching is supported, the Secure Association Protocol phase is REQUIRED, and MUST provide for freshness of the TSKs.

[b] Key lifetime determination. EAP does not support negotiation of key lifetimes, nor does it support rekey without reauthentication. As a result, the Secure Association Protocol may handle rekey and determination of the key lifetime. Where key caching is supported, secure negotiation of key lifetimes is RECOMMENDED. Lower layers that support rekey, but not key caching, may not require key lifetime negotiation. To take an example from IKE, the difference between IKEv1 and IKEv2 is that in IKEv1 SA lifetimes were negotiated. In IKEv2, each end of the SA is responsible for

enforcing its own lifetime policy on the SA and rekeying the SA when necessary.

[c] Key resynchronization. It is possible for the peer or authenticator to reboot or reclaim resources, clearing portions or all of the key cache. Therefore, key lifetime negotiation cannot guarantee that the key cache will remain synchronized, and the peer may not be able to determine before attempting to use a AAA-Key whether it exists within the authenticator cache. It is therefore RECOMMENDED for the Secure Association Protocol to provide a mechanism for key state resynchronization. Since in this situation one or more of the parties initially do not possess a key with which to protect the resynchronization exchange, securing this mechanism may be difficult.

[d] Key selection. Where key caching is supported, it may be possible for the EAP peer and authenticator to share more than one key of a given type. As a result, the Secure Association Protocol needs to support key selection, using the EAP Key Naming scheme described in this document.

[e] Key scope determination. Since the Discovery phase is handled out-of-band, EAP does not provide a mechanism by which the peer can determine the authenticator identity. As a result, where the authenticator has multiple ports and AAA-Key caching is supported, the EAP peer may not be able to determine the scope of validity of a AAA-Key. Similarly, where the EAP peer has multiple ports, the authenticator may not be able to determine whether a peer has authorization to use a particular AAA-Key. To allow key scope determination, the lower layer SHOULD provide a mechanism by which the peer can determine the scope of the AAA-Key cache on each authenticator, and by which the authenticator can determine the scope of the AAA-Key cache on a peer.

4.1. Key Caching

In existing implementations, key caching may be supported on the EAP peer and authenticator but not on the backend server. Where explicitly supported by the lower layer, the EAP peer and authenticator MAY cache the AAA-Key and/or TSKs. The structure of the key cache on the peer and authenticator is defined by the lower layer. Unless specified by the lower layer, the EAP peer and authenticator MUST assume that peers and authenticators do not cache

the AAA-Key or TSKs.

In existing AAA server implementations, all keys exported by EAP methods (including the MSK, EMSK and IV) and calculated keys (e.g. AAA-Key) are not cached and are lost after EAP authentication

completes:

[1] In order to avoid key reuse, on the EAP server, transported keys are deleted once they are sent. An EAP server MUST NOT retain keys that it has previously sent to the authenticator. For example, an EAP server that has transported a AAA-Key based on the MSK MUST delete the MSK, and no keys may be derived from the MSK from that point forward by the server.

[2] Keys which are not transported, such as the EMSK, are also deleted by existing implementations.

4.2. Parent-Child Relationships

When keying material exported by EAP methods expires, all keying material derived from the exported keying material expires, including the AAA-Key and TSKs.

When an EAP reauthentication takes place, new keying material is derived and exported by the EAP method, which eventually results in replacement of calculated keys, including the AAA-Key and TSKs.

As a result, while the lifetime of calculated keys can be less than or equal that of the exported keys they are derived from, it cannot be greater. For example, TSK rekey may occur prior to EAP reauthentication.

Failure to mutually prove possession of the AAA-Key during the Secure Association Protocol exchange need not be grounds for deletion of the AAA-Key by both parties; rate-limiting Secure Association Protocol exchanges could be used to prevent a brute force attack.

4.3. Local Key Lifetimes

The Transient EAP Keys (TEKs) are session keys used to protect the EAP conversation. The TEKs are internal to the EAP method and are

not exported. TEKs are typically created during an EAP conversation, used until the end of the conversation and then discarded. However, methods may rekey TEKs during a conversation.

When using TEKs within an EAP conversation or across conversations, it is necessary to ensure that replay protection and key separation requirements are fulfilled. For instance, if a replay counter is used, TEK rekey MUST occur prior to wrapping of the counter. Similarly, TSKs MUST remain cryptographically separate from TEKs despite TEK rekeying or caching. This prevents TEK compromise from leading directly to compromise of the TSKs and vice versa.

EAP methods may cache local keying material which may persist for multiple EAP conversations when fast reconnect is used [RFC 3748]. For example, EAP methods based on TLS (such as EAP-TLS [RFC2716]) derive and cache the TLS Master Secret, typically for substantial time periods. The lifetime of other local keying material calculated within the EAP method is defined by the method. Note that in general, when using fast reconnect, there is no guarantee to that the original long-term credentials are still in the possession of the peer. For instance, a card hold holding the private key for EAP-TLS may have been removed. EAP servers SHOULD also verify that the long-term credentials are still valid, such as by checking that certificate used in the original authentication has not yet expired.

4.4. Exported and Calculated Key Lifetimes

All EAP methods generating keys are required to generate the MSK and EMSK, and may optionally generate the IV. However, EAP, defined in [RFC3748], does not support the negotiation of lifetimes for exported keying material such as the MSK, EMSK and IV.

Several mechanisms exist for managing key lifetimes:

[a] AAA attributes. AAA protocols such as RADIUS [RFC2865] and Diameter [I-D.ietf-aaa-eap] support the Session-Timeout attribute. The Session-Timeout value represents the maximum lifetime of the exported keys, and all keys calculated from it, on the authenticator. Since existing AAA servers do not cache keys exported by EAP methods, or keys calculated from exported keys, the value of the Session-Timeout attribute has no bearing on the key

lifetime within the AAA server.

On the authenticator, where EAP is used for authentication, the Session-Timeout value represents the maximum session time prior to re-authentication, as described in [RFC3580]. Where EAP is used for pre-authentication, the session may not start until some future time, or may never occur. Nevertheless, the Session-Timeout value represents the time after which the AAA-Key, and all keys calculated from it, will have expired on the authenticator. If the session subsequently starts, re-authentication will be initiated once the Session-Time has expired. If the session never started, or started and ended, the AAA-Key and all keys calculated from it will be expired by the authenticator prior to the future time indicated by Session-Timeout.

Since the TSK lifetime is often determined by authenticator resources, the AAA server has no insight into the TSK derivation process, and by the principle of ciphersuite independence, it is not appropriate for the AAA server to manage any aspect of the TSK

derivation process, including the TSK lifetime.

[b] Lower layer mechanisms. While AAA attributes can communicate the maximum exported key lifetime, this only serves to synchronize the key lifetime between the backend authentication server and the authenticator. Lower layer mechanisms such as the Secure Association Protocol can then be used to enable the lifetime of exported and calculated keys to be negotiated between the peer and authenticator.

Where TSKs are established as the result of a Secure Association Protocol exchange, it is RECOMMENDED that the Secure Association Protocol include support for TSK resynchronization. Where the TSK is taken from the AAA-Key, there is no need to manage the TSK lifetime as a separate parameter, since the TSK lifetime and AAA-Key lifetime are identical.

[c] System defaults. Where the EAP method does not support the negotiation of the exported key lifetime, and a key lifetime negotiation mechanism is not provided by the lower lower, there may be no way for the peer to learn the exported key lifetime. In this case it is RECOMMENDED that the peer assume a default value of the

exported key lifetime; 8 hours is recommended. Similarly, the lifetime of calculated keys can also be managed as a system parameter on the authenticator.

[d] Method specific negotiation within EAP. While EAP itself does not support lifetime negotiation, it would be possible to specify methods that do. However, systems that rely on such negotiation for exported keys would only function with these methods. As a result, it is NOT RECOMMENDED to use this approach as the sole way to determine key lifetimes.

4.5. Key cache synchronization

Issues arise when attempting to synchronize the key cache on the peer and authenticator. Lifetime negotiation alone cannot guarantee key cache synchronization.

One problem is that the AAA protocol cannot guarantee synchronization of key lifetimes between the peer and authenticator. Where the Secure Association Protocol is not run immediately after EAP authentication, the exported and calculated key lifetimes will not be known by the peer during the hiatus. Where EAP pre-authentication occurs, this can leave the peer uncertain whether a subsequent attempt to use the exported keys will prove successful.

However, even where the Secure Association Protocol is run

immediately after EAP, it is still possible for the authenticator to reclaim resources if the created key state is not immediately utilized.

The lower layer may utilize Discovery mechanisms to assist in this. For example, the authenticator manages the AAA-Key cache by deleting the oldest AAA-Key first (LIFO), the relative creation time of the last AAA-Key to be deleted could be advertised with the Discovery phase, enabling the peer to determine whether a given AAA-Key had been expired from the authenticator key cache prematurely.

4.6. Key Scope

As described in Section 2.5, in existing applications the AAA-Key is derived from the MSK by the EAP peer and server, and is used as the

root of the ciphersuite-specific key hierarchy. Where a backend authentication server is present, the AAA-Key is transported from the EAP server to the authenticator; where it is not present, the AAA-Key is calculated on the authenticator.

Regardless of how many sessions are initiated using it, the AAA-Key scope is between the EAP peer that calculates it, and the authenticator that either calculates it (where no backend authenticator is present) or receives it from the server (where a backend authenticator server is present).

It should be understood that an authenticator or peer:

[a] may contain multiple physical ports;
[b] may advertise itself as multiple "virtual" authenticators or peers;
[c] may utilize multiple CPUs;
[d] may support clustering services for load balancing or failover.

As illustrated in Figure 1, an EAP peer with multiple ports may be attached to one or more authenticators, each with multiple ports. Where the peer and authenticator identify themselves using a port identifier such as a link layer address, it may not be obvious to the peer which authenticator ports are associated with which authenticators. Similarly, it may not be obvious to the authenticator which peer ports are associated with which peers. As a result, the peer and authenticator may not be able to determine the scope of the AAA-Key.

When a single physical authenticator advertises itself as multiple "virtual authenticators", the EAP peer and authenticator also may not be able to agree on the scope of the AAA-Key, creating a security vulnerability. For example, the peer may assume that the "virtual

authenticators" are distinct and do not share a key cache, whereas, depending on the architecture of the physical AP, a shared key cache may or may not be implemented.

Where the AAA-Key is shared between "virtual authenticators" an attacker acting as a peer could authenticate with the "Guest" "virtual authenticator" and derive a AAA-Key. If the virtual authenticators share a key cache, then the peer can utilize the AAA-

Key derived for the "Guest" network to obtain access to the "Corporate Intranet" virtual authenticator.

Several measures are recommended to address these issues:

[a] Authenticators are REQUIRED to cache associated authorizations along with the AAA-Key and apply authorizations consistently. This ensures that an attacker cannot obtain elevated privileges even where the AAA-Key cache is shared between "virtual authenticators".

[b] It is RECOMMENDED that physical authenticators maintain separate AAA-Key caches for each "virtual authenticator".

[c] It is RECOMMENDED that each "virtual authenticator" identify itself distinctly to the AAA server, such as by utilizing a distinct NAS-identifier attribute. This enables the AAA server to utilize a separate credential to authenticate each "virtual authenticator".

[d] It is RECOMMENDED that Secure Association Protocols identify peers and authenticators unambiguously, without incorporating implicit assumptions about peer and authenticator architectures. Using port-specific MAC addresses as identifiers is NOT RECOMMENDED where peers and authenticators may support multiple ports.

[e] The AAA server and authenticator MAY implement additional attributes in order to further restrict the AAA-Key scope. For example, in 802.11, the AAA server may provide the authenticator with a list of authorized Called or Calling-Station-Ids and/or SSIDs for which the AAA-Key is valid.

[f] Where the AAA server provides attributes restricting the key scope, it is RECOMMENDED that restrictions be securely communicated by the authenticator to the peer. This can be accomplished using the Secure Association Protocol, but also can be accomplished via the EAP method or the lower layer.

4.7. Key Strength

In order to guard against brute force attacks, EAP methods deriving keys need to be capable of generating keys with an appropriate

effective symmetric key strength. In order to ensure that key

generation is not the weakest link, it is RECOMMENDED that EAP methods utilizing public key cryptography choose a public key that has a cryptographic strength meeting the symmetric key strength requirement.

As noted in [RFC3766] Section 5, this results in the following required RSA or DH module and DSA subgroup size in bits, for a given level of attack resistance in bits:

Attack Resistance (bits)	RSA or DH Modulus size (bits)	DSA subgroup size (bits)
70	947	128
80	1228	145
90	1553	153
100	1926	184
150	4575	279
200	8719	373
250	14596	475

4.8. Key Wrap

As described in [RFC3579] Section 4.3, known problems exist in the key wrap specified in [RFC2548]. Where the same RADIUS shared secret is used by a PAP authenticator and an EAP authenticator, there is a vulnerability to known plaintext attack. Since RADIUS uses the shared secret for multiple purposes, including per-packet authentication, attribute hiding, considerable information is exposed about the shared secret with each packet. This exposes the shared secret to dictionary attacks. MD5 is used both to compute the RADIUS Response Authenticator and the Message-Authenticator attribute, and some concerns exist relating to the security of this hash [MD5Attack].

As discussed in [RFC3579] Section 4.3, the security vulnerabilities of RADIUS are extensive, and therefore development of an alternative key wrap technique based on the RADIUS shared secret would not substantially improve security. As a result, [RFC3759] Section 4.2 recommends running RADIUS over IPsec. The same approach is taken in Diameter EAP [I-D.ietf-aaa-eap], which defines cleartext key attributes, to be protected by IPsec or TLS.

Where an untrusted AAA intermediary is present (such as a RADIUS proxy or a Diameter agent), and data object security is not used, the AAA-Key may be recovered by an attacker in control of the untrusted intermediary. Possession of the AAA-Key enables decryption of data traffic sent between the peer and a specific authenticator. However,

as long as a AAA-Key or keys derived from it is only utilized by a single authenticator, compromise of the AAA-Key does not enable an attacker to impersonate the peer to another authenticator. Vulnerability to an untrusted AAA intermediary can be mitigated by implementation of redirect functionality, as described in [RFC3588] and [I-D.ietf-aaa-eap].

5. Handoff Vulnerabilities

With EAP, a number of mechanisms are be utilized in order to reduce the latency of handoff between authenticators. One such mechanism is EAP pre-authentication, in which EAP is utilized to pre-establish a AAA-Key on an authenticator prior to arrival of the peer. Another such mechanism is AAA-Key caching, in which an EAP peer can re-attach to an authenticator without having to re-authenticate using EAP. Yet another mechanism is context transfer, such as is defined in [IEEE-802.11F] and [CTP]. These mechanisms introduce new security vulnerabilities, as discussed in the sections that follow.

5.1. Authorization

In a typical network access scenario (dial-in, wireless LAN, etc.) access control mechanisms are typically applied. These mechanisms include user authentication as well as authorization for the offered service.

As a part of the authentication process, the AAA network determines the user's authorization profile. The user authorizations are transmitted by the backend authentication server to the EAP authenticator (also known as the Network Access Server or authenticator) included with the AAA-Token, which also contains the AAA-Key, in Phase 1b of the EAP conversation. Typically, the profile is determined based on the user identity, but a certificate presented by the user may also provide authorization information.

The backend authentication server is responsible for making a user authorization decision, answering the following questions:

[a] Is this a legitimate user for this particular network?

[b] Is this user allowed the type of access he or she is requesting?

[c] Are there any specific parameters (mandatory tunneling, bandwidth, filters, and so on) that the access network should be aware of for this user?

[d] Is this user within the subscription rules regarding time of day?

[e] Is this user within his limits for concurrent sessions?

[f] Are there any fraud, credit limit, or other concerns that indicate that access should be denied?

While the authorization decision is in principle simple, the process is complicated by the distributed nature of AAA decision making. Where brokering entities or proxies are involved, all of the AAA devices in the chain from the authenticator to the home AAA server are involved in the decision. For instance, a broker can disallow access even if the home AAA server would allow it, or a proxy can add authorizations (e.g., bandwidth limits).

Decisions can be based on static policy definitions and profiles as well as dynamic state (e.g. time of day or limits on the number of concurrent sessions). In addition to the Accept/Reject decision made by the AAA chain, parameters or constraints can be communicated to the authenticator.

The criteria for Accept/Reject decisions or the reasons for choosing particular authorizations are typically not communicated to the authenticator, only the final result. As a result, the authenticator has no way to know what the decision was based on. Was a set of authorization parameters sent because this service is always provided to the user, or was the decision based on the time/day and the capabilities of the requesting authenticator device?

5.2. Correctness

When the AAA exchange is bypassed via use of techniques such as AAA-Key caching, this creates challenges in ensuring that authorization is properly handled. These include:

[a] Consistent application of session time limits. Bypassing AAA should not automatically increase the available session time, allowing a user to endlessly extend their network access by changing the point of attachment.

[b] Avoidance of privilege elevation. Bypassing AAA should not result

in a user being granted access to services which they are not entitled to.

[c] Consideration of dynamic state. In situations in which dynamic state is involved in the access decision (day/time, simultaneous session limit) it should be possible to take this state into account either before or after access is granted. Note that consideration of network-wide state such as simultaneous session limits can typically only be taken into account by the backend

authentication server.

[d] Encoding of restrictions. Since a authenticator may not be aware of the criteria considered by a backend authentication server when allowing access, in order to ensure consistent authorization during a fast handoff it may be necessary to explicitly encode the restrictions within the authorizations provided in the AAA-Token.

[e] State validity. The introduction of fast handoff should not render the authentication server incapable of keeping track of network-wide state.

A handoff mechanism capable of addressing these concerns is said to be "correct". One condition for correctness is as follows: For a handoff to be "correct" it MUST establish on the new device the same context as would have been created had the new device completed a AAA conversation with the authentication server.

A properly designed handoff scheme will only succeed if it is "correct" in this way. If a successful handoff would establish "incorrect" state, it is preferable for it to fail, in order to avoid creation of incorrect context.

Some backend authentication server and authenticator configurations are incapable of meeting this definition of "correctness". For example, if the old and new device differ in their capabilities, it may be difficult to meet this definition of correctness in a handoff mechanism that bypasses AAA. Backend authentication servers often perform conditional evaluation, in which the authorizations returned in an Access-Accept message are contingent on the authenticator or on dynamic state such as the time of day or number of simultaneous sessions. For example, in a heterogeneous deployment, the backend

authentication server might return different authorizations depending on the authenticator making the request, in order to make sure that the requested service is consistent with the authenticator capabilities.

If differences between the new and old device would result in the backend authentication server sending a different set of messages to the new device than were sent to the old device, then if the handoff mechanism bypasses AAA, then the handoff cannot be carried out correctly.

For example, if some authenticator devices within a deployment support dynamic VLANs while others do not, then attributes present in the Access-Request (such as the authenticator-IP-Address, authenticator-Identifier, Vendor-Identifier, etc.) could be examined to determine when VLAN attributes will be returned, as described in

[RFC3580]. VLAN support is defined in [IEEE-802.1Q]. If a handoff bypassing the backend authentication server were to occur between a authenticator supporting dynamic VLANs and another authenticator which does not, then a guest user with access restricted to a guest VLAN could be given unrestricted access to the network.

Similarly, in a network where access is restricted based on the day and time, Service Set Identifier (SSID), Calling-Station-Id or other factors, unless the restrictions are encoded within the authorizations, or a partial AAA conversation is included, then a handoff could result in the user bypassing the restrictions.

In practice, these considerations limit the situations in which fast handoff mechanisms bypassing AAA can be expected to be successful. Where the deployed devices implement the same set of services, it may be possible to do successful handoffs within such mechanisms. However, where the supported services differ between devices, the handoff may not succeed. For example, [RFC2865] section 1.1 states:

> "A authenticator that does not implement a given service MUST NOT implement the RADIUS attributes for that service. For example, a authenticator that is unable to offer ARAP service MUST NOT implement the RADIUS attributes for ARAP. A authenticator MUST treat a RADIUS access-accept authorizing an unavailable service as an access-reject instead."

Note that this behavior only applies to attributes that are known, but not implemented. For attributes that are unknown, [RFC2865] Section 5 states:

> "A RADIUS server MAY ignore Attributes with an unknown Type. A RADIUS client MAY ignore Attributes with an unknown Type."

In order to perform a correct handoff, if a new device is provided with RADIUS context for a known but unavailable service, then it MUST process this context the same way it would handle a RADIUS Access-Accept requesting an unavailable service. This MUST cause the handoff to fail. However, if a new device is provided with RADIUS context that indicates an unknown attribute, then this attribute MAY be ignored.

Although it may seem somewhat counter-intuitive, failure is indeed the "correct" result where a known but unsupported service is requested. Presumably a correctly configured backend authentication server would not request that a device carry out a service that it does not implement. This implies that if the new device were to complete a AAA conversation that it would be likely to receive different service instructions. In such a case, failure of the

handoff is the desired result. This will cause the new device to go back to the AAA server in order to receive the appropriate service definition.

In practice, this implies that handoff mechanisms which bypass AAA are most likely to be successful within a homogeneous device deployment within a single administrative domain. For example, it would not be advisable to carry out a fast handoff bypassing AAA between a authenticator providing confidentiality and another authenticator that does not support this service. The correct result of such a handoff would be a failure, since if the handoff were blindly carried out, then the user would be moved from a secure to an insecure channel without permission from the backend authentication server. Thus the definition of a "known but unsupported service" MUST encompass requests for unavailable security services. This includes vendor-specific attributes related to security, such as those described in [RFC2548].

6. Security Considerations

6.1. Security Terminology

"Cryptographic binding", "Cryptographic separation", "Key strength" and "Mutual authentication" are defined in [RFC3748] and are used with the same meaning here.

6.2. Threat Model

The EAP threat model is described in [RFC3748] Section 7.1. In order to address these threats, EAP relies on the security properties of EAP methods (known as "security claims", described in [RFC3784] Section 7.2.1). EAP method requirements for application such as Wireless LAN authentication are described in [RFC4017].

The RADIUS threat model is described in [RFC3579] Section 4.1, and responses to these threats are described in [RFC3579] Sections 4.2 and 4.3. Among other things, [RFC3579] Section 4.2 recommends the use of IPsec ESP with non-null transform to provide per-packet authentication and confidentiality, integrity and replay protection for RADIUS/EAP.

Given the existing documentation of EAP and AAA threat models and responses, there is no need to duplicate that material here. However, there are many other system-level threats no covered in these document which have not been described or analyzed elsewhere. These include:

[1] An attacker may try to modify or spoof Secure Association Protocol packets.

[2] An attacker compromising an authenticator may provide incorrect information to the EAP peer and/or server via out-of-band mechanisms (such as via a AAA or lower layer protocol). This includes impersonating another authenticator, or providing inconsistent information to the peer and EAP server.

[3] An attacker may attempt to perform downgrading attacks on the ciphersuite negotiation within the Secure Association Protocol in

order to ensure that a weaker ciphersuite is used to protect data.

Depending on the lower layer, these attacks may be carried out without requiring physical proximity.

In order to address these threats, [Housley] describes the mandatory system security properties:

Algorithm independence
: Wherever cryptographic algorithms are chosen, the algorithms must be negotiable, in order to provide resilient against compromise of a particular algorithm. Algorithm independence must be demonstrated within all aspects of the system, including within EAP, AAA and the Secure Association Protocol. However, for interoperability, at least one suite of algorithms MUST be implemented.

Strong, fresh session keys
: Session keys must be demonstrated to be strong and fresh in all circumstances, while at the same time retaining algorithm independence.

Replay protection
: All protocol exchanges must be replay protected. This includes exchanges within EAP, AAA, and the Secure Association Protocol.

Authentication
: All parties need to be authenticated. The confidentiality of the authenticator must be maintained. No plaintext passwords are allowed.

Authorization
: EAP peer and authenticator authorization must be performed.

Session keys
: Confidentiality of session keys must be maintained.

Ciphersuite negotiation
: The selection of the "best" ciphersuite must be securely confirmed.

Unique naming

Session keys must be uniquely named.

Domino effect
: Compromise of a single authenticator cannot compromise any other part of the system, including session keys and long-term secrets.

Key binding
: The key must be bound to the appropriate context.

6.3. Security Analysis

Figure 8 illustrates the relationship between the peer, authenticator and backend authentication server.

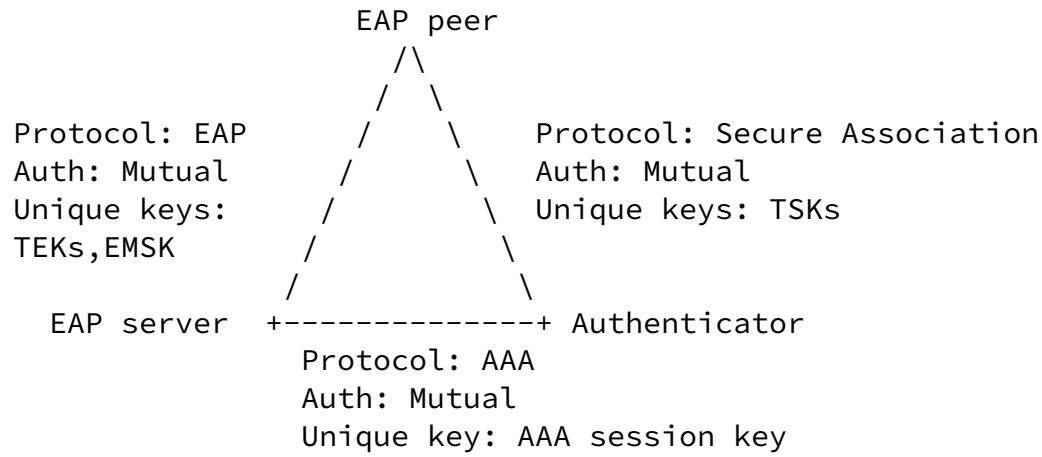


Figure 8: Relationship between peer, authenticator and auth. server

The peer and EAP server communicate using EAP [RFC3748]. The security properties of this communication are largely determined by the chosen EAP method. Method security claims are described in [RFC3748] Section 7.2. These include the key strength, protected ciphersuite negotiation, mutual authentication, integrity protection, replay protection, confidentiality, key derivation, key strength, dictionary attack resistance, fast reconnect, cryptographic binding, session independence, fragmentation and channel binding claims. At a minimum, methods claiming to support key derivation must also support mutual authentication. As noted in [RFC3748] Section 7.10:

> EAP Methods deriving keys MUST provide for mutual authentication between the EAP peer and the EAP Server.

Ciphersuite independence is also required:

> Keying material exported by EAP methods MUST be independent of the ciphersuite negotiated to protect data.

In terms of key strength and freshness, [RFC3748] Section 10 says:

> EAP methods SHOULD ensure the freshness of the MSK and EMSK even in cases where one party may not have a high quality random number generator.... In order to preserve algorithm independence, EAP methods deriving keys SHOULD support (and document) the protected negotiation of the ciphersuite used to protect the EAP conversation between the peer and server... In order to enable deployments requiring strong keys, EAP methods supporting key derivation SHOULD be capable of generating an MSK and EMSK, each with an effective key strength of at least 128 bits.

The authenticator and backend authentication server communicate using a AAA protocol such as RADIUS [RFC3579] or Diameter [I-D.ietf-aaa-eap]. As noted in [RFC3588] Section 13, Diameter must be protected by either IPsec ESP with non-null transform or TLS. As a result, Diameter requires per-packet integrity and confidentiality. Replay protection must be supported. For RADIUS, [RFC3579] Section 4.2 recommends that RADIUS be protected by IPsec ESP with a non-null transform, and where IPsec is implemented replay protection must be supported.

The peer and authenticator communicate using the Secure Association Protocol.

As noted in the figure, each party in the exchange mutually authenticates with each of the other parties, and derives a unique key. All parties in the diagram have access to the AAA-Key.

The EAP peer and backend authentication server mutually authenticate via the EAP method, and derive the TEKs and EMSK which are known only to them. The TEKs are used to protect some or all of the EAP conversation between the peer and authenticator, so as to guard against modification or insertion of EAP packets by an attacker. The degree of protection afforded by the TEKs is determined by the EAP method; some methods may protect the entire EAP packet, including the EAP header, while other methods may only protect the contents of the Type-Data field, defined in [RFC3748].

Since EAP is spoken only between the EAP peer and server, if a backend authentication server is present then the EAP conversation does not provide mutual authentication between the peer and authenticator, only between the EAP peer and EAP server (backend authentication server). As a result, mutual authentication between

the peer and authenticator only occurs where a Secure Association

protocol is used, such the unicast and group key derivation handshake supported in [IEEE-802.11i]. This means that absent use of a secure Association Protocol, from the point of view of the peer, EAP mutual authentication only proves that the authenticator is trusted by the backend authentication server; the identity of the authenticator is not confirmed.

Utilizing the AAA protocol, the authenticator and backend authentication server mutually authenticate and derive session keys known only to them, used to provide per-packet integrity and replay protection, authentication and confidentiality. The AAA-Key is distributed by the backend authentication server to the authenticator over this channel, bound to attributes constraining its usage, as part of the AAA-Token. The binding of attributes to the AAA-Key within a protected package is important so the authenticator receiving the AAA-Token can determine that it has not been compromised, and that the keying material has not been replayed, or mis-directed in some way.

The security properties of the EAP exchange are dependent on each leg of the triangle: the selected EAP method, AAA protocol and the Secure Association Protocol.

Assuming that the AAA protocol provides protection against rogue authenticators forging their identity, then the AAA-Token can be assumed to be sent to the correct authenticator, and where it is wrapped appropriately, it can be assumed to be immune to compromise by a snooping attacker.

Where an untrusted AAA intermediary is present, the AAA-Token must not be provided to the intermediary so as to avoid compromise of the AAA-Token. This can be avoided by use of re-direct as defined in [RFC3588].

When EAP is used for authentication on PPP or wired IEEE 802 networks, it is typically assumed that the link is physically secure, so that an attacker cannot gain access to the link, or insert a rogue device. EAP methods defined in [RFC3748] reflect this usage model. These include EAP MD5, as well as One-Time Password (OTP) and Generic Token Card. These methods support one-way authentication (from EAP

peer to authenticator) but not mutual authentication or key derivation. As a result, these methods do not bind the initial authentication and subsequent data traffic, even when the the ciphersuite used to protect data supports per-packet authentication and integrity protection. As a result, EAP methods not supporting mutual authentication are vulnerable to session hijacking as well as attacks by rogue devices.

On wireless networks such as IEEE 802.11 [IEEE-802.11], these attacks become easy to mount, since any attacker within range can access the wireless medium, or act as an access point. As a result, new ciphersuites have been proposed for use with wireless LANs [IEEE-802.11i] which provide per-packet authentication, integrity and replay protection. In addition, mutual authentication and key derivation, provided by methods such as EAP-TLS [RFC2716] are required [IEEE-802.11i], so as to address the threat of rogue devices, and provide keying material to bind the initial authentication to subsequent data traffic.

If the selected EAP method does not support mutual authentication, then the peer will be vulnerable to attack by rogue authenticators and backend authentication servers. If the EAP method does not derive keys, then TSKs will not be available for use with a negotiated ciphersuite, and there will be no binding between the initial EAP authentication and subsequent data traffic, leaving the session vulnerable to hijack.

If the backend authentication server does not protect against authenticator masquerade, or provide the proper binding of the AAA-Key to the session within the AAA-Token, then one or more AAA-Keys may be sent to an unauthorized party, and an attacker may be able to gain access to the network. If the AAA-Token is provided to an untrusted AAA intermediary, then that intermediary may be able to modify the AAA-Key, or the attributes associated with it, as described in [RFC2607].

If the Secure Association Protocol does not provide mutual proof of possession of the AAA-Key material, then the peer will not have assurance that it is connected to the correct authenticator, only that the authenticator and backend authentication server share a trust relationship (since AAA protocols support mutual

authentication). This distinction can become important when multiple authenticators receive AAA-Keys from the backend authentication server, such as where fast handoff is supported. If the TSK derivation does not provide for protected ciphersuite and capabilities negotiation, then downgrade attacks are possible.

6.4. Man-in-the-middle Attacks

As described in [I-D.puthenkulam-eap-binding], EAP method sequences and compound authentication mechanisms may be subject to man-in-the-middle attacks. When such attacks are successfully carried out, the attacker acts as an intermediary between a victim and a legitimate authenticator. This allows the attacker to authenticate successfully to the authenticator, as well as to obtain access to the network.

In order to prevent these attacks, [I-D.puthenkulam-eap-binding] recommends derivation of a compound key by which the EAP peer and server can prove that they have participated in the entire EAP exchange. Since the compound key must not be known to an attacker posing as an authenticator, and yet must be derived from quantities that are exported by EAP methods, it may be desirable to derive the compound key from a portion of the EMSK. In order to provide proper key hygiene, it is recommended that the compound key used for man-in-the-middle protection be cryptographically separate from other keys derived from the EMSK, such as fast handoff keys, discussed in Section 2.3.

6.5. Denial of Service Attacks

The caching of security associations may result in vulnerability to denial of service attacks. Since an EAP peer may derive multiple EAP SAs with a given EAP server, and creation of a new EAP SA does not implicitly delete a previous EAP SA, EAP methods that result in creation of persistent state may be vulnerable to denial of service attacks by a rogue EAP peer.

As a result, EAP methods creating persistent state may wish to limit the number of cached EAP SAs (Phase 1a) corresponding to an EAP peer. For example, an EAP server may choose to only retain a few EAP SAs for each peer. This prevents a rogue peer from denying access to other peers.

Similarly, an authenticator may have multiple AAA-Key SAs corresponding to a given EAP peer; to conserve resources an authenticator may choose to limit the number of cached AAA-Key (Phase 1 b) SAs for each peer.

Depending on the media, creation of a new unicast Secure Association SA may or may not imply deletion of a previous unicast secure association SA. Where there is no implied deletion, the authenticator may choose to limit Phase 2 (unicast and multicast) Secure Association SAs for each peer.

6.6. Impersonation

Both the RADIUS and Diameter protocols are potentially vulnerable to impersonation by a rogue authenticator.

While AAA protocols such as RADIUS [RFC2865] or Diameter [RFC3588] support mutual authentication between the authenticator (known as the AAA client) and the backend authentication server (known as the AAA server), the security mechanisms vary according to the AAA protocol.

In RADIUS, the shared secret used for authentication is determined by the source address of the RADIUS packet. As noted in [RFC3579] Section 4.3.7, it is highly desirable that the source address be checked against one or more NAS identification attributes so as to detect and prevent impersonation attacks.

When RADIUS requests are forwarded by a proxy, the NAS-IP-Address or NAS-IPv6-Address attributes may not correspond to the source address. Since the NAS-Identifier attribute need not contain an FQDN, it also may not correspond to the source address, even indirectly. [RFC2865] Section 3 states:

> A RADIUS server MUST use the source IP address of the RADIUS UDP packet to decide which shared secret to use, so that RADIUS requests can be proxied.

This implies that it is possible for a rogue authenticator to forge NAS-IP-Address, NAS-IPv6-Address or NAS-Identifier attributes within a RADIUS Access-Request in order to impersonate another

authenticator. Among other things, this can result in messages (and MSKs) being sent to the wrong authenticator. Since the rogue authenticator is authenticated by the RADIUS proxy or server purely based on the source address, other mechanisms are required to detect the forgery. In addition, it is possible for attributes such as the Called-Station-Id and Calling-Station-Id to be forged as well.

As recommended in [RFC3579], this vulnerability can be mitigated by having RADIUS proxies check authenticator identification attributes against the source address.

To allow verification of session parameters such as the Called-Station- Id and Calling-Station-Id, these can be sent by the EAP peer to the server, protected by the TEKs. The RADIUS server can then check the parameters sent by the EAP peer against those claimed by the authenticator. If a discrepancy is found, an error can be logged.

While [RFC3588] requires use of the Route-Record AVP, this utilizes FQDNs, so that impersonation detection requires DNS A/AAAA and PTR RRs to be properly configured. As a result, it appears that Diameter is as vulnerable to this attack as RADIUS, if not more so. To address this vulnerability, it is necessary to allow the backend authentication server to communicate with the authenticator directly, such as via the redirect functionality supported in [RFC3588].

6.7. Channel binding

It is possible for a compromised or poorly implemented EAP authenticator to communicate incorrect information to the EAP peer and/or server. This may enable an authenticator to impersonate another authenticator or communicate incorrect information via out-of-band mechanisms (such as via AAA or the lower layer protocol).

Where EAP is used in pass-through mode, the EAP peer typically does not verify the identity of the pass-through authenticator, it only verifies that the pass-through authenticator is trusted by the EAP server. This creates a potential security vulnerability, described in

[RFC3748] Section 7.15.

[RFC3579] Section 4.3.7 describes how an EAP pass-through authenticator acting as a AAA client can be detected if it attempts to impersonate another authenticator (such by sending incorrect NAS-Identifier [RFC2865], NAS-IP-Address [RFC2865] or NAS-IPv6-Address [RFC3162] attributes via the AAA protocol). However, it is possible for a pass-through authenticator acting as a AAA client to provide correct information to the AAA server while communicating misleading information to the EAP peer via a lower layer protocol.

For example, it is possible for a compromised authenticator to utilize another authenticator's Called-Station-Id or NAS-Identifier in communicating with the EAP peer via a lower layer protocol, or for a pass-through authenticator acting as a AAA client to provide an incorrect peer Calling-Station-Id [RFC2865][RFC3580] to the AAA server via the AAA protocol.

As noted in [RFC3748] Section 7.15, this vulnerability can be addressed by use of EAP methods that support a protected exchange of channel properties such as endpoint identifiers, including (but not limited to): Called-Station-Id [RFC2865][RFC3580], Calling-Station-Id [RFC2865][RFC3580], NAS-Identifier [RFC2865], NAS-IP-Address [RFC2865], and NAS-IPv6-Address [RFC3162].

Using such a protected exchange, it is possible to match the channel properties provided by the authenticator via out-of-band mechanisms against those exchanged within the EAP method. For example, see [ServiceIdent].

7. Security Requirements

This section summarizes the security requirements that must be met by EAP methods, AAA protocols, Secure Association Protocols and Ciphersuites in order to address the security threats described in this document. These requirements MUST be met by specifications

requesting publication as an RFC. Each requirement provides a pointer to the sections of this document describing the threat that it mitigates.

7.1. EAP Method Requirements

It is possible for the peer and EAP server to mutually authenticate and derive keys. In order to provide keying material for use in a subsequently negotiated ciphersuite, an EAP method supporting key derivation MUST export a Master Session Key (MSK) of at least 64 octets, and an Extended Master Session Key (EMSK) of at least 64 octets. EAP Methods deriving keys MUST provide for mutual authentication between the EAP peer and the EAP Server.

The MSK and EMSK MUST NOT be used directly to protect data; however, they are of sufficient size to enable derivation of a AAA-Key subsequently used to derive Transient Session Keys (TSKs) for use with the selected ciphersuite. Each ciphersuite is responsible for specifying how to derive the TSKs from the AAA-Key.

The AAA-Key is derived from the keying material exported by the EAP method (MSK and EMSK). This derivation occurs on the AAA server. In many existing protocols that use EAP, the AAA-Key and MSK are equivalent, but more complicated mechanisms are possible.

EAP methods SHOULD ensure the freshness of the MSK and EMSK even in cases where one party may not have a high quality random number generator. A RECOMMENDED method is for each party to provide a nonce of at least 128 bits, used in the derivation of the MSK and EMSK.

EAP methods export the MSK and EMSK and not Transient Session Keys so as to allow EAP methods to be ciphersuite and media independent. Keying material exported by EAP methods MUST be independent of the ciphersuite negotiated to protect data.

Depending on the lower layer, EAP methods may run before or after ciphersuite negotiation, so that the selected ciphersuite may not be known to the EAP method. By providing keying material usable with any ciphersuite, EAP methods can used with a wide range of ciphersuites and media.

It is RECOMMENDED that methods providing integrity protection of EAP packets include coverage of all the EAP header fields, including the Code, Identifier, Length, Type and Type-Data fields.

In order to preserve algorithm independence, EAP methods deriving keys SHOULD support (and document) the protected negotiation of the ciphersuite used to protect the EAP conversation between the peer and

server. This is distinct from the ciphersuite negotiated between the peer and authenticator, used to protect data.

The strength of Transient Session Keys (TSKs) used to protect data is ultimately dependent on the strength of keys generated by the EAP method. If an EAP method cannot produce keying material of sufficient strength, then the TSKs may be subject to brute force attack. In order to enable deployments requiring strong keys, EAP methods supporting key derivation SHOULD be capable of generating an MSK and EMSK, each with an effective key strength of at least 128 bits.

Methods supporting key derivation MUST demonstrate cryptographic separation between the MSK and EMSK branches of the EAP key hierarchy. Without violating a fundamental cryptographic assumption (such as the non-invertibility of a one-way function) an attacker recovering the MSK or EMSK MUST NOT be able to recover the other quantity with a level of effort less than brute force.

Non-overlapping substrings of the MSK MUST be cryptographically separate from each other. That is, knowledge of one substring MUST NOT help in recovering some other non-overlapping substring without breaking some hard cryptographic assumption. This is required because some existing ciphersuites form TSKs by simply splitting the AAA-Key to pieces of appropriate length. Likewise, non-overlapping substrings of the EMSK MUST be cryptographically separate from each other, and from substrings of the MSK. The EMSK MUST NOT be transported to, or shared with, additional parties.

Since EAP does not provide for explicit key lifetime negotiation, EAP peers, authenticators and authentication servers MUST be prepared for situations in which one of the parties discards key state which remains valid on another party.

The development and validation of key derivation algorithms is difficult, and as a result EAP methods SHOULD reuse well established and analyzed mechanisms for MSK and EMSK key derivation (such as those specified in IKE [RFC2409] or TLS [RFC2246]), rather than inventing new ones.

7.1.1. Requirements for EAP methods

In order for an EAP method to meet the guidelines for EMSK usage it must meet the following requirements:

- It MUST specify how to derive the EMSK
- The key material used for the EMSK MUST be

computationally independent of the MSK and TEKs.

- The EMSK MUST NOT be used for any other purpose than the key derivation described in this document.
- The EMSK MUST be secret and not known to someone observing the authentication mechanism protocol exchange.
- The EMSK MUST NOT be exported from the EAP server.
- The EMSK MUST be unique for each session.
- The EAP mechanism SHOULD a unique identifier suitable for naming the EM

7.1.2. Requirements for EAP applications

In order for an application to meet the guidelines for EMSK usage it must meet the following requirements:

- New applications following this specification SHOULD NOT use the MSK. If more than one application uses the MSK, then the cryptographic separation is not achieved. Implementations SHOULD prevent such combinations.
- A peer MUST NOT use the EMSK directly for cryptographic protection of data.

7.2. AAA Protocol Requirements

AAA protocols suitable for use in transporting EAP MUST provide the following facilities:

Security services
: AAA protocols used for transport of EAP keying material MUST implement and SHOULD use per-packet integrity and authentication, replay protection and confidentiality. These requirements are met by Diameter EAP [I-D.ietf-aaa-eap], as well as RADIUS over IPsec [RFC3579].

Session Keys
: AAA protocols used for transport of EAP keying material MUST implement and SHOULD use dynamic key management in order to derive

fresh session keys, as in Diameter EAP [I-D.ietf-aaa-eap] and RADIUS over IPsec [RFC3579], rather than using a static key, as originally defined in RADIUS [RFC2865].

Mutual authentication
AAA protocols used for transport of EAP keying material MUST

provide for mutual authentication between the authenticator and backend authentication server. These requirements are met by Diameter EAP [I-D.ietf-aaa-eap] as well as by RADIUS EAP [RFC3579].

Authorization
AAA protocols used for transport of EAP keying material SHOULD provide protection against rogue authenticators masquerading as other authenticators. This can be accomplished, for example, by requiring that AAA agents check the source address of packets against the origin attributes (Origin-Host AVP in Diameter, NAS-IP-Address, NAS-IPv6-Address, NAS-Identifier in RADIUS). For details, see [RFC3579] Section 4.3.7.

Key transport
Since EAP methods do not export Transient Session Keys (TSKs) in order to maintain media and ciphersuite independence, the AAA server MUST NOT transport TSKs from the backend authentication server to authenticator.

Key transport specification
In order to enable backend authentication servers to provide keying material to the authenticator in a well defined format, AAA protocols suitable for use with EAP MUST define the format and wrapping of the AAA-Token.

EMSK transport
Since the EMSK is a secret known only to the backend authentication server and peer, the AAA-Token MUST NOT transport the EMSK from the backend authentication server to the authenticator.

AAA-Token protection
To ensure against compromise, the AAA-Token MUST be integrity protected, authenticated, replay protected and encrypted in transit, using well-established cryptographic algorithms.

Session Keys
: The AAA-Token SHOULD be protected with session keys as in Diameter [RFC3588] or RADIUS over IPsec [RFC3579] rather than static keys, as in [RFC2548].

Key naming
: In order to ensure against confusion between the appropriate keying material to be used in a given Secure Association Protocol exchange, the AAA-Token SHOULD include explicit key names and context appropriate for informing the authenticator how the keying material is to be used.

Key Compromise
: Where untrusted intermediaries are present, the AAA-Token SHOULD NOT be provided to the intermediaries. In Diameter, handling of keys by intermediaries can be avoided using Redirect functionality [RFC3588].

7.3. Secure Association Protocol Requirements

The Secure Association Protocol supports the following:

Entity Naming
: The peer and authenticator SHOULD identify themselves in a manner that is independent of their attached ports.

Mutual proof of possession
: The peer and authenticator MUST each demonstrate possession of the keying material transported between the backend authentication server and authenticator (AAA-Key).

Key Naming
: The Secure Association Protocol MUST explicitly name the keys used in the proof of possession exchange, so as to prevent confusion when more than one set of keying material could potentially be used as the basis for the exchange.

Creation and Deletion
: In order to support the correct processing of phase 2 security associations, the Secure Association (phase 2) protocol MUST

support the naming of phase 2 security associations and associated transient session keys, so that the correct set of transient session keys can be identified for processing a given packet. The phase 2 Secure Association Protocol also MUST support transient session key activation and SHOULD support deletion, so that establishment and re-establishment of transient session keys can be synchronized between the parties.

Integrity and Replay Protection
: The Secure Association Protocol MUST support integrity and replay protection of all messages.

Direct operation
: Since the phase 2 Secure Association Protocol is concerned with the establishment of security associations between the EAP peer and authenticator, including the derivation of transient session keys, only those parties have "a need to know" the transient session keys. The Secure Association Protocol MUST operate directly between the peer and authenticator, and MUST NOT be passed-through to the backend authentication server, or include additional parties.

Derivation of transient session keys
: The Secure Association Protocol negotiation MUST support derivation of unicast and multicast transient session keys suitable for use with the negotiated ciphersuite.

TSK freshness
: The Secure Association (phase 2) Protocol MUST support the derivation of fresh unicast and multicast transient session keys, even when the keying material provided by the backend authentication server is not fresh. This is typically supported by including an exchange of nonces within the Secure Association Protocol.

Bi-directional operation
: While some ciphersuites only require a single set of transient session keys to protect traffic in both directions, other ciphersuites require a unique set of transient session keys in each direction. The phase 2 Secure Association Protocol SHOULD provide for the derivation of unicast and multicast keys in each direction, so as not to require two separate phase 2 exchanges in order to create a bi-directional phase 2 security association.

Secure capabilities negotiation
: The Secure Association Protocol MUST support secure capabilities negotiation. This includes security parameters such as the security association identifier (SAID) and ciphersuites, as well as negotiation of the lifetime of the TSKs, AAA-Key and exported EAP keys. Secure capabilities negotiation also includes confirmation of the capabilities discovered during the discovery phase (phase 0), so as to ensure that the announced capabilities have not been forged.

Key Scoping
: The Secure Association Protocol MUST ensure the synchronization of key scope between the peer and authenticator. This includes negotiation of restrictions on key usage.

7.4. Ciphersuite Requirements

Ciphersuites suitable for keying by EAP methods MUST provide the following facilities:

TSK derivation
: In order to allow a ciphersuite to be usable within the EAP keying framework, a specification MUST be provided describing how transient session keys suitable for use with the ciphersuite are derived from the AAA-Key.

EAP method independence
: Algorithms for deriving transient session keys from the AAA-Key MUST NOT depend on the EAP method. However, algorithms for deriving TEKs MAY be specific to the EAP method.

Cryptographic separation
: The TSKs derived from the AAA-Key MUST be cryptographically separate from each other. Similarly, TEKs MUST be cryptographically separate from each other. In addition, the TSKs MUST be cryptographically separate from the TEKs.

8. IANA Considerations

This document does not create any new name spaces nor does it

allocate any protocol parameters.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

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

9.2. Informative References

[CTP] Loughney, J., Nakhjiri, M., Perkins, C. and R. Koodli, "Context Transfer Protocol", draft-ietf-seamoby-ctp-11.txt, Internet draft (work in progress), August 2004.

[DESMODES] National Institute of Standards and Technology, "DES Modes of Operation", FIPS PUB 81, December 1980, <http://www.itl.nist.gov/fipspubs/fip81.htm>.

[FIPSDES] National Institute of Standards and Technology, "Data Encryption Standard", FIPS PUB 46, January 1977.

[Housley] Housley, R. and B. Aboba, "AAA Key Management", draft-housley-aaa-key-mgmt-00.txt, Internet draft (work in progress), June 2005..IP [IEEE-802] Institute of Electrical and Electronics

Engineers, "IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture", ANSI/IEEE Standard 802, 1990.

[IEEE-802.11] Institute of Electrical and Electronics Engineers, "Information technology - Telecommunications and information

exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE IEEE Standard 802.11-2003, 2003.

[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.

[IEEE-802.1Q]
Institute of Electrical and Electronics Engineers, "IEEE Standards for Local and Metropolitan Area Networks: Draft Standard for Virtual Bridged Local Area Networks", IEEE Standard 802.1Q/D8, January 1998.

[IEEE-802.11i]
Institute of Electrical and Electronics Engineers, "Supplement to STANDARD FOR Telecommunications and Information Exchange between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Specification for Enhanced Security", IEEE 802.11i, December 2004.

[IEEE-802.11F]
Institute of Electrical and Electronics Engineers, "Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation", IEEE 802.11F, July 2003.

[IEEE-02-758]
Mishra, A., Shin, M., Arbaugh, W., Lee, I. and K. Jang, "Proactive Caching Strategies for IAPP Latency Improvement during 802.11 Handoff", IEEE 802.11 Working Group, IEEE-02-758r1-F Draft 802.11I/D5.0, November 2002.

[IEEE-03-084]
Mishra, A., Shin, M., Arbaugh, W., Lee, I. and K. Jang, "Proactive Key Distribution to support fast and secure roaming", IEEE 802.11 Working Group, IEEE-03-084r1-I,

http://www.ieee802.org/11/Documents/DocumentHolder/ 3-084.zip, January 2003.

[IEEE-03-155]
Aboba, B., "Fast Handoff Issues", IEEE 802.11 Working Group, IEEE-03-155r0-I, http://www.ieee802.org/11/Documents/DocumentHolder/3-155.zip, March 2003.

[I-D.ietf-roamops-cert]
Aboba, B., "Certificate-Based Roaming", draft-ietf-roamops-cert-02 (work in progress), April 1999.

[I-D.ietf-aaa-eap]
Eronen, P., Hiller, T. and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", draft-ietf-aaa-eap-10 (work in progress), November 2004.

[I-D.puthenkulam-eap-binding]
Puthenkulam, J., "The Compound Authentication Binding Problem", draft-puthenkulam-eap-binding-04 (work in progress), October 2003.

[I-D.arkko-pppext-eap-aka]
Arkko, J. and H. Haverinen, "EAP AKA Authentication", draft-arkko-pppext-eap-aka-15.txt (work in progress), December 2004.

[IKEv2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", draft-ietf-ipsec-ikev2-17 (work in progress), September 2004.

[MD5Attack]
Dobbertin, H., "The Status of MD5 After a Recent Attack", CryptoBytes, Vol.2 No.2, 1996.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[RFC1968] Meyer, G. and K. Fox, "The PPP Encryption Control Protocol (ECP)", RFC 1968, June 1996.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2246] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2419] Sklower, K. and G. Meyer, "The PPP DES Encryption Protocol, Version 2 (DESE-bis)", RFC 2419, September 1998.

[RFC2420] Kummert, H., "The PPP Triple-DES Encryption Protocol (3DESE)", RFC 2420, September 1998.

[RFC2516] Mamakos, L., Lidl, K., Evarts, J., Carrel, D., Simone, D. and R. Wheeler, "A Method for Transmitting PPP Over Ethernet (PPPoE)", RFC 2516, February 1999.

[RFC2548] Zorn, G., "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, March 1999.

[RFC2607] Aboba, B. and J. Vollbrecht, "Proxy Chaining and Policy Implementation in Roaming", RFC 2607, June 1999.

[RFC2716] Aboba, B. and D. Simon, "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999.

[RFC2865] Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3078] Pall, G. and G. Zorn, "Microsoft Point-To-Point Encryption (MPPE) Protocol", RFC 3078, March 2001.

[RFC3079] Zorn, G., "Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE)", RFC 3079, March 2001.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G. and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, September 2003.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G. and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", RFC 3766, April

2004.

[RFC4017] Stanley, D., Walker, J. and B. Aboba, "EAP Method Requirements for Wireless LANs", RFC 4017, March 2005.

[8021XHandoff] Pack, S. and Y. Choi, "Pre-Authenticated Fast Handoff in a Public Wireless LAN Based on IEEE 802.1X Model", School of Computer Science and Engineering, Seoul National University, Seoul, Korea, 2002.

Acknowledgments

Thanks to Arun Ayyagari, Ashwin Palekar, and Tim Moore of Microsoft, Dorothy Stanley of Agere, Bob Moskowitz of TruSecure, Jesse Walker of Intel, Joe Salowey of Cisco and Russ Housley of Vigil Security for useful feedback.

Author Addresses

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: bernarda@microsoft.com
Phone: +1 425 706 6605
Fax: +1 425 936 7329

Dan Simon
Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

EMail: dansimon@microsoft.com

Phone: +1 425 706 6711
Fax: +1 425 936 7329

Jari Arkko
Ericsson
Jorvas 02420
Finland

Phone:
EMail: jari.arkko@ericsson.com

Pasi Eronen

Nokia Research Center
P.O. Box 407
FIN-00045 Nokia Group
Finland

EMail: pasi.eronen@nokia.com

Henrik Levkowetz (editor)
ipUnplugged AB
Arenavagen 27
Stockholm S-121 28
SWEDEN

Phone: +46 708 32 16 08
EMail: henrik@levkowetz.com

Appendix A - Ciphersuite Keying Requirements

To date, PPP and IEEE 802.11 ciphersuites are suitable for keying by EAP. This Appendix describes the keying requirements of common PPP and 802.11 ciphersuites.

PPP ciphersuites include DESEbis [RFC2419], 3DES [RFC2420], and MPPE [RFC3078]. The DES algorithm is described in [FIPSDES], and DES modes (such as CBC, used in [RFC2419] and DES-EDE3-CBC, used in [RFC2420]) are described in [DESMODES]. For PPP DESEbis, a single 56-bit encryption key is required, used in both directions. For PPP 3DES, a 168-bit encryption key is needed, used in both directions. As described in [RFC2419] for DESEbis and [RFC2420] for 3DES, the IV, which is different in each direction, is "deduced from an explicit 64-bit nonce, which is exchanged in the clear during the [ECP] negotiation phase." There is therefore no need for the IV to be provided by EAP.

For MPPE, 40-bit, 56-bit or 128-bit encryption keys are required in each direction, as described in [RFC3078]. No initialization vector is required.

While these PPP ciphersuites provide encryption, they do not provide per-packet authentication or integrity protection, so an authentication key is not required in either direction.

Within [IEEE-802.11], Transient Session Keys (TSKs) are required both for unicast traffic as well as for multicast traffic, and therefore separate key hierarchies are required for unicast keys and multicast keys. IEEE 802.11 ciphersuites include WEP-40, described in [IEEE-802.11], which requires a 40-bit encryption key, the same in either direction; and WEP-128, which requires a 104-bit encryption key, the same in either direction. These ciphersuites also do not support per-packet authentication and integrity protection. In addition to these unicast keys, authentication and encryption keys are required to wrap the multicast encryption key.

Recently, new ciphersuites have been proposed for use with IEEE 802.11 that provide per-packet authentication and integrity protection as well as encryption [IEEE-802.11i]. These include TKIP, which requires a single 128-bit encryption key and two 64-bit authentication keys (one for each direction); and AES CCMP, which requires a single 128-bit key (used in both directions) in order to authenticate and encrypt data.

As with WEP, authentication and encryption keys are also required to wrap the multicast encryption (and possibly, authentication) keys.

Appendix B - Transient EAP Key (TEK) Hierarchy

Figure B-1 illustrates the TEK key hierarchy for EAP-TLS [RFC2716], which is based on the TLS key hierarchy described in [RFC2246]. The TLS-negotiated ciphersuite is used to set up a protected channel for use in protecting the EAP conversation, keyed by the derived TEKs. The TEK derivation proceeds as follows:

```
master_secret = TLS-PRF-48(pre_master_secret, "master secret",
                client.random || server.random)
TEK           = TLS-PRF-X(master_secret, "key expansion",
                server.random || client.random)
Where:
TLS-PRF-X =     TLS pseudo-random function defined in [RFC2246],
                computed to X octets.
```

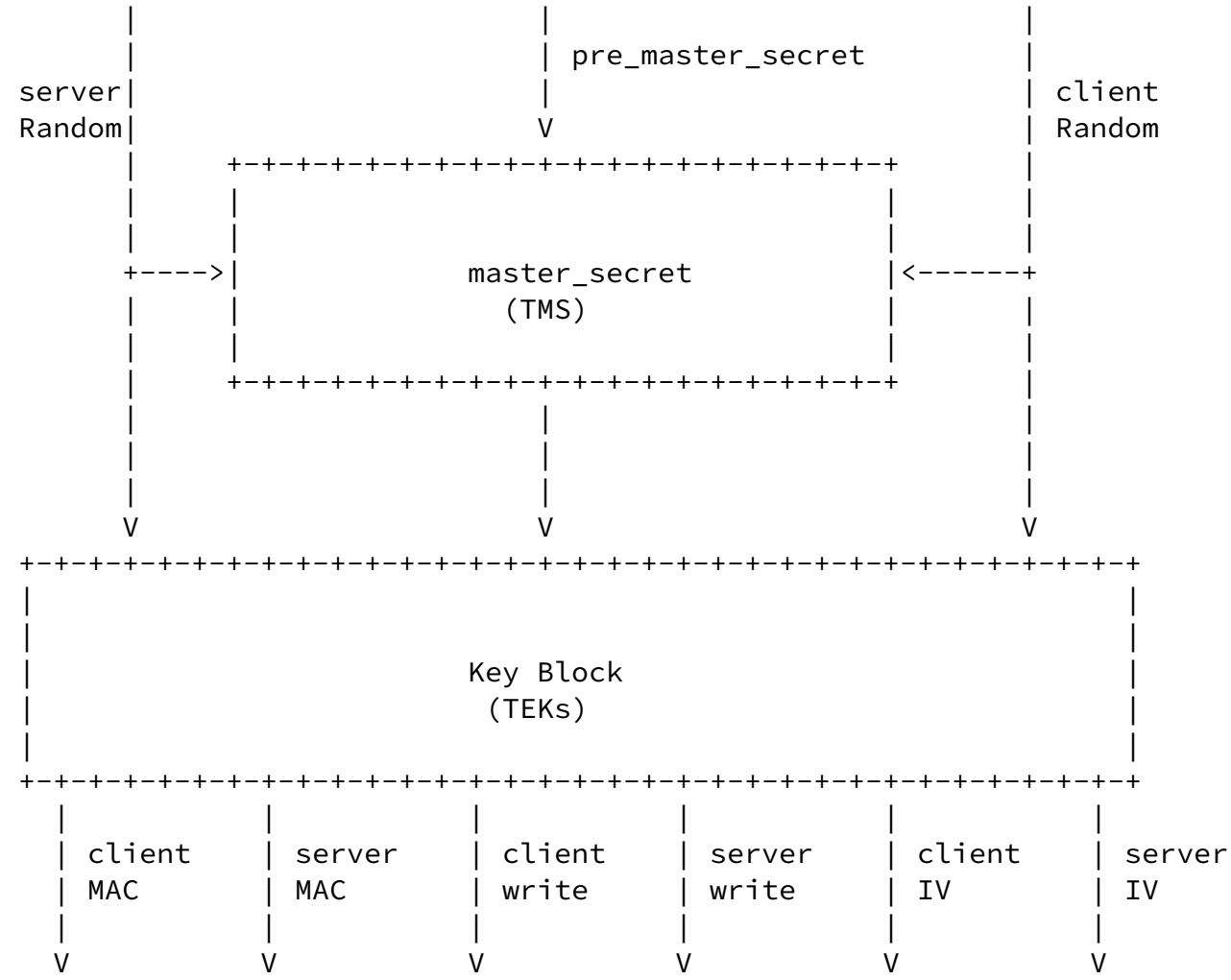


Figure B-1 - TLS [RFC2246] Key Hierarchy

Appendix C - EAP-TLS Key Hierarchy

In EAP-TLS [RFC2716], the MSK is divided into two halves, corresponding to the "Peer to Authenticator Encryption Key" (Enc-RECV-Key, 32 octets, also known as the PMK) and "Authenticator to Peer Encryption Key" (Enc-SEND-Key, 32 octets). In [RFC2548], the Enc-RECV-Key (the PMK) is transported in the MS-MPPE-Recv-Key attribute, and the Enc-SEND-Key is transported in the MS-MPPE-Send-

Key attribute.

The EMSK is also divided into two halves, corresponding to the "Peer to Authenticator Authentication Key" (Auth-RECV-Key, 32 octets) and "Authenticator to Peer Authentication Key" (Auth-SEND-Key, 32 octets). The IV is a 64 octet quantity that is a known value; octets 0-31 are known as the "Peer to Authenticator IV" or RECV-IV, and Octets 32-63 are known as the "Authenticator to Peer IV", or SEND-IV.

In EAP-TLS, the MSK, EMSK and IV are derived from the TLS master secret via a one-way function. This ensures that the TLS master secret cannot be derived from the MSK, EMSK or IV unless the one-way function (TLS PRF) is broken. Since the MSK is derived from the the TLS master secret, if the TLS master secret is compromised then the MSK is also compromised.

The key derivation scheme specified in RFC 2716 that was specified prior to the introduction of the terminology MSK and EMSK MUST be interpreted as follows:

```
MSK           = TLS-PRF-64(TMS, "client EAP encryption",
                   client.random || server.random)
EMSK          = second 64 octets of:
                TLS-PRF-128(TMS, "client EAP encryption",
                   client.random || server.random)
IV            = TLS-PRF-64("", "client EAP encryption",
                   client.random || server.random)

AAA-Key(0,31) = Peer to Authenticator Encryption Key (Enc-RECV-Key)
                (MS-MPPE-Recv-Key in [RFC2548]). Also known as the
                PMK.
AAA-Key(32,63)= Authenticator to Peer Encryption Key (Enc-SEND-Key)
                (MS-MPPE-Send-Key in [RFC2548])
EMSK(0,31)    = Peer to Authenticator Authentication Key (Auth-RECV-Key)
EMSK(32,63)   = Authenticator to Peer Authentication Key (Auth-Send-Key)
IV(0,31)      = Peer to Authenticator Initialization Vector (RECV-IV)
IV(32,63)     = Authenticator to Peer Initialization vector (SEND-IV)
```

Where:

AAA-Key(W,Z) = Octets W through Z includes of the AAA-Key.

```
IV(W,Z)        = Octets W through Z inclusive of the IV.
MSK(W,Z)       = Octets W through Z inclusive of the MSK.
EMSK(W,Z)      = Octets W through Z inclusive of the EMSK.
TMS            = TLS master_secret
TLS-PRF-X      = TLS PRF function defined in [RFC2246] computed to X octets
client.random = Nonce generated by the TLS client.
server.random = Nonce generated by the TLS server.
```

Figure C-1 describes the process by which the MSK,EMSK,IV and ultimately the TSKs, are derived from the TLS Master Secret.

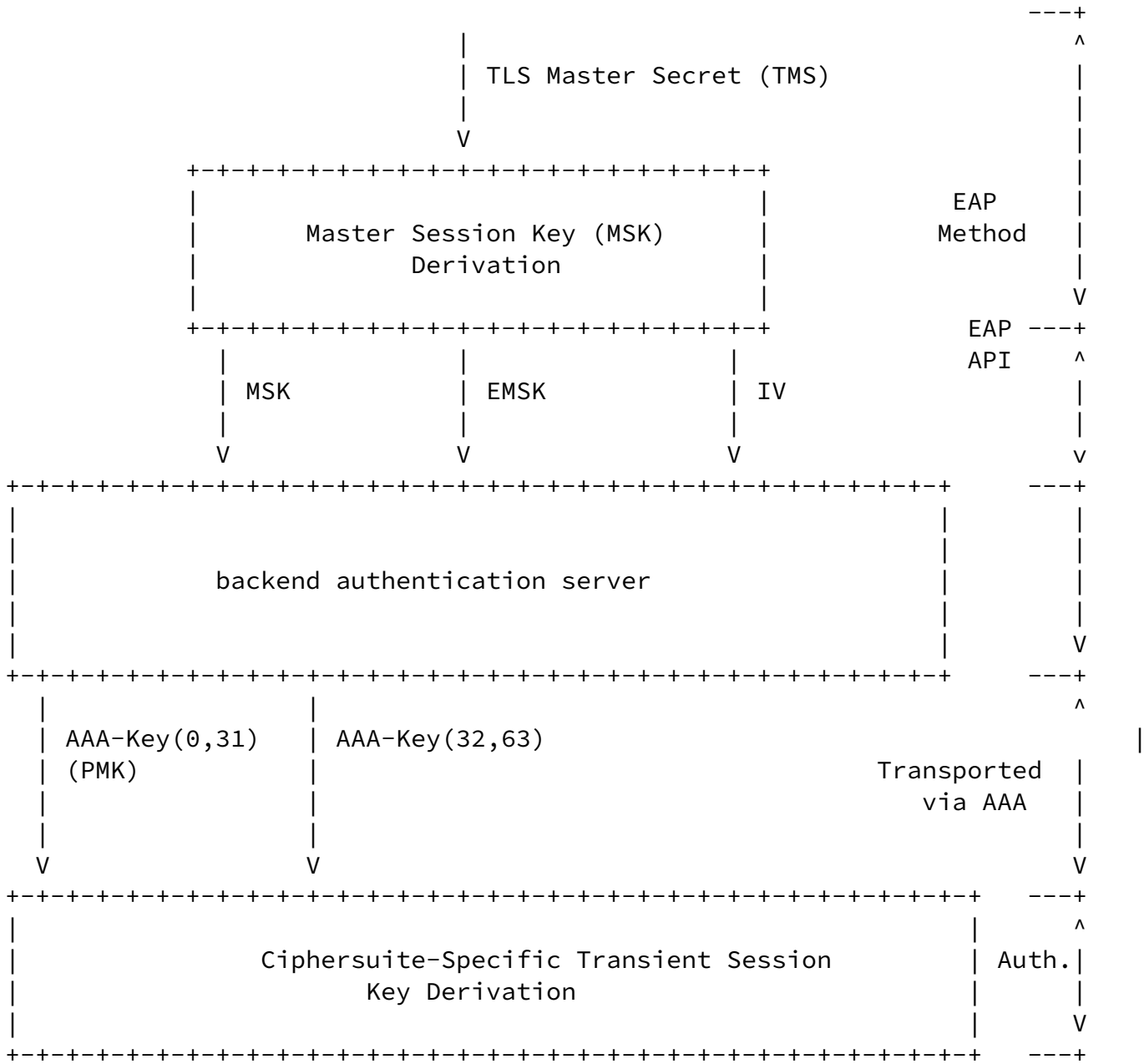


Figure C-1 - EAP TLS [RFC2716] Key hierarchy

Appendix D - Example Transient Session Key (TSK) Derivation

Within IEEE 802.11 RSN, the Pairwise Transient Key (PTK), a transient session key used to protect unicast traffic, is derived from the PMK (octets 0-31 of the MSK), known in [RFC2716] as the Peer to Authenticator Encryption Key. In [IEEE-802.11i], the PTK is derived from the PMK via the following formula:

```
PTK = EAPOL-PRF-X(PMK, "Pairwise key expansion", Min(AA,SA) ||
      Max(AA, SA) || Min(ANonce,SNonce) || Max(ANonce,SNonce))
```

Where:

```
PMK             = AAA-Key(0,31)
SA              = Station MAC address (Calling-Station-Id)
AA              = Access Point MAC address (Called-Station-Id)
ANonce          = Access Point Nonce
SNonce          = Station Nonce
EAPOL-PRF-X     = Pseudo-Random Function based on HMAC-SHA1, generating
                  a PTK of size X octets.
```

TKIP uses X = 64, while CCMP, WRAP, and WEP use X = 48.

The EAPOL-Key Confirmation Key (KCK) is used to provide data origin authenticity in the TSK derivation. It utilizes the first 128 bits (bits 0-127) of the PTK. The EAPOL-Key Encryption Key (KEK) provides confidentiality in the TSK derivation. It utilizes bits 128-255 of the PTK. Bits 256-383 of the PTK are used by Temporal Key 1, and Bits 384-511 are used by Temporal Key 2. Usage of TK1 and TK2 is ciphersuite specific. Details are available in [IEEE-802.11i].

Appendix E - Exported Parameters in Existing Methods

This Appendix specifies Method-ID, Peer-ID, Server-ID and Key-Lifetime for EAP methods that have been published prior to this specification. Future EAP method specifications MUST include a definition of the Method-ID, Peer-ID, and Server-ID (could be the empty string) and MAY also define the Key-Lifetime (assumed to be indeterminate if not described).

EAP-Identity

The EAP-Identity method does not derive keys, and therefore does not define the Key-Lifetime or Method-ID. The Peer-ID exported by the Identity method is determined by the octets included within the EAP- Response/Identity. The Server-ID is the empty string (zero length).

EAP-Notification

The EAP-Notification method does not derive keys and therefore does not define the Key-Lifetime and Method-ID. The Peer-ID and Server-ID are the empty string (zero length).

EAP-GTC

The EAP-GTC method does not derive keys and therefore does not define the Key-Lifetime and Method-ID. The Peer-ID and Server-ID are the empty string.

EAP-OTP

The EAP-OTP method does not derive keys and therefore does not define the Key-Lifetime and Method-ID. The Peer-ID and Server-ID are the empty string.

EAP-TLS

The EAP-TLS Method-Id is the concatenation of the peer and server nonces.

The Peer-ID and Server-ID are the contents of the altSubjectName in the peer and server certificates.

EAP-TLS does not negotiate a Key-Lifetime.

EAP-AKA

The EAP-AKA Method-Id is the contents of the RAND field from the

AT_RAND attribute, followed by the contents of the AUTN field in the AT_AUTN attribute.

The Peer-ID is the contents of the Identity field from the AT_IDENTITY attribute, using only the Actual Identity Length octets from the beginning, however. Note that the contents are used as they are transmitted, regardless of whether the transmitted identity was a permanent, pseudonym, or fast reauthentication identity. The Server- ID is an empty string. EAP-AKA does not negotiate a key lifetime.

EAP-SIM

The Method-Id is the contents of the RAND field from the AT_RAND attribute, followed by the contents of the NONCE_MT field in the AT_NONCE_MT attribute.

The Peer-ID is the contents of the Identity field from the AT_IDENTITY attribute, using only the Actual Identity Length octets from the beginning, however. Note that the contents are used as they are transmitted, regardless of whether the transmitted identity was a permanent, pseudonym, or fast reauthentication identity. The Server- ID is an empty string. EAP-SIM does not negotiate a key lifetime.

Appendix F - Security Association Examples

EAP Method SA Example: EAP-TLS

In EAP-TLS [RFC2716], after the EAP authentication the client (peer) and server can store the following information:

- Implicitly, the EAP method this SA refers to (EAP-TLS)
- Session identifier (a value selected by the server)
- Certificate of the other party (server stores the client's certificate and vice versa)
- Ciphersuite and compression method
- TLS Master secret (known as the EAP-TLS Master Key)
- SA lifetime (ensuring that the SA is not stored forever)
- If the client has multiple different credentials (certificates and corresponding private keys), a pointer to those credentials

When the server initiates EAP-TLS, the client can look up the EAP-TLS SA based on the credentials it was going to use (certificate and private key), and the expected credentials (certificate or name) of the server. If an EAP-TLS SA exists, and it is not too old, the client informs the server about the existence of this SA by including its Session-Id in the TLS ClientHello message. The server then looks up the correct SA based on the Session-Id (or detects that it doesn't yet have one).

EAP Method SA Example: EAP-AKA

In EAP-AKA [I-D.arkko-pppext-eap-aka], after EAP authentication the client and server can store the following information:

- Implicitly, the EAP method this SA refers to (EAP-AKA)
- A re-authentication pseudonym
- The client's permanent identity (IMSI)
- Replay protection counter
- Authentication key (K_aut)
- Encryption key (K_encr)
- Original Master Key (MK)
- SA lifetime (ensuring that the SA is not stored forever)

When the server initiates EAP-AKA, the client can look up the EAP-AKA SA based on the credentials it was going to use (permanent identity). If an EAP-AKA SA exists, and it is not too old, the client informs the server about the existence of this SA by sending its re-authentication pseudonym as its identity in EAP Identity Response message, instead of its permanent identity. The server then looks up the correct SA based on this identity.

AAA SA Example: RADIUS

In RADIUS, where shared secret authentication is used, the client and server store each other's IP address and the shared secret, which is used to calculate the Response Authenticator [RFC2865] and Message-Authenticator [RFC3579] values, and to encrypt some attributes (such as the AAA-Key, see [RFC3580] Section 3.16).

Where IPsec is used to protect RADIUS [RFC3579] and IKE is used for key management, the parties store information necessary to authenticate and authorize the other party (e.g. certificates, trust anchors and names). The IKE exchange results in IKE Phase 1 and Phase 2 SAs containing information used to protect the conversation (session keys, selected ciphersuite, etc.)

AAA SA Example: Diameter with TLS

When using Diameter protected by TLS, the parties store information

necessary to authenticate and authorize the other party (e.g. certificates, trust anchors and names). The TLS handshake results in a short-term TLS SA that contains information used to protect the actual communications (session keys, selected TLS ciphersuite, etc.).

Service SA Example: 802.11i

[IEEE802.11i] Section 8.4.1.1 defines the security associations used within IEEE 802.11. A summary follows; the standard should be consulted for details.

o Pairwise Master Key Security Association (PMKSA)

The PMKSA is a bi-directional SA, used by both parties for sending and receiving. The PMKSA is the Root Service SA. It is created on the peer when EAP authentication completes successfully or a pre-shared key is configured. The PMKSA is created on the authenticator when the PMK is received or created on the authenticator or a pre-shared key is configured. The PMKSA is used to create the PTKSA. PMKSAs are cached for their lifetimes. The PMKSA consists of the following elements:

- PMKID (security association identifier)
- Authenticator MAC address
- PMK
- Lifetime
- Authenticated Key Management Protocol (AKMP)
- Authorization parameters specified by the AAA server or by local configuration. This can include parameters such as the peer's authorized SSID.

On the peer, this information can be locally configured.
- Key replay counters (for EAPOL-Key messages)
- Reference to PTKSA (if any), needed to:
  - o delete it (e.g. AAA server-initiated disconnect)
  - o replace it when a new four-way handshake is done
- Reference to accounting context, the details of which depend on the accounting protocol used, the implementation and administrative details. In RADIUS, this could include (e.g. packet and octet counters, and Acct-Multi-Session-Id).

o Pairwise Transient Key Security Association (PTKSA)

The PTKSA is a bi-directional SA created as the result of a successful four-way handshake. The PTKSA is a unicast service SA. There may only be one PTKSA between a pair of peer and authenticator MAC addresses. PTKSAs are cached for the lifetime of the PMKSA. Since the PTKSA is tied to the PMKSA, it only has the additional information from the 4-way handshake. The PTKSA consists of the following:

- Key (PTK)
- Selected ciphersuite
- MAC addresses of the parties
- Replay counters, and ciphersuite specific state
- Reference to PMKSA: This is needed when:
  - A new four-way handshake is needed (lifetime, TKIP countermeasures), and we need to know which PMKSA to use

o Group Transient Key Security Association (GTKSA)

The GTKSA is a uni-directional SA created based on the four-way handshake or the group key handshake. The GTKSA is a multicast service SA. A GTKSA consists of the following:

- Direction vector (whether the GTK is used for transmit or receive)
- Group cipher suite selector
- Key (GTK)
- Authenticator MAC address
- Via reference to PMKSA, or copied here:
  - Authorization parameters
  - Reference to accounting context

Service SA Example: IKEv2/IPsec

Note that this example is intended to be informative, and it does not necessarily include all information stored.

o IKEv2 SA

- Protocol version
- Identities of the parties

- IKEv2 SPIs
- Selected ciphersuite
- Replay protection counters (Message ID)
- Keys for protecting IKEv2 messages (SK_ai/SK_ar/SK_ei/SK_er)
- Key for deriving keys for IPsec SAs (SK_d)
- Lifetime information
- On the authenticator, service authorization information received from the backend authentication server.

When processing an incoming message, the correct SA is looked up based on the SPIs.

o IPsec SAs/SPD

- Traffic selectors
- Replay protection counters
- Selected ciphersuite
- IPsec SPI
- Keys
- Lifetime information
- Protocol mode (tunnel or transport)

The correct SA is looked up based on SPI (for inbound packets), or SPD traffic selectors (for outbound traffic). A separate IPsec SA exists for each direction.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Open Issues

Open issues relating to this specification are tracked on the following web site:

http://www.drizzle.com/~aboba/EAP/eapissues.html