

# Extensible Authentication Protocols

**Revised: March 20, 2009, OL-17222-03**

Cisco Access Registrar (CAR) supports the Extensible Authentication Protocol (EAP) to provide a common protocol for differing authentication mechanisms. EAP enables the dynamic selection of the authentication mechanism at authentication time based on information transmitted in the Access-Request. (This type of EAP authentication mechanism is called an authentication exchange.)

Extensible Authentication Protocols (EAP) provide for support of multiple authentication methods. CAR 4.2 supports the following EAP authentication methods:

- [EAP-FAST, page 8-2](#)
- [EAP-GTC, page 8-12](#)
- [EAP-LEAP, page 8-13](#)
- [EAP-MD5, page 8-14](#)
- [EAP-Negotiate, page 8-15](#)
- [EAP-MSChapV2, page 8-17](#)
- [EAP-SIM, page 8-18](#)
- [EAP-Transport Level Security \(TLS\), page 8-21](#)
- [EAP-TTLS, page 8-24](#)
- [Protected EAP, page 8-34](#)
  - [PEAP Version 0, page 8-34](#) (Microsoft PEAP)
  - [PEAP Version 1, page 8-38](#) (Cisco PEAP)

In general, you enable each EAP method by creating and configuring a service of the desired type. Use the **radclient**

Both versions of Protected EAP (PEAP) are able to use other EAP methods as the authentication mechanism that is protected by PEAP encryption. For PEAP Version 0, the supported authentication methods are EAP-MSChapV2, EAP-SIM, EAP-TLS and EAP-Negotiate. For PEAP Version 1, the supported authentication methods are EAP-GTC, EAP-SIM, EAP-TLS and EAP-Negotiate.

The PEAP protocol consists of two phases: an authentication handshake phase and a tunnel phase where another complete EAP authentication exchange takes place protected by the session keys negotiated by phase one. CAR 4.2 supports the tunneling of other EAP methods within the PEAP phase two exchange.

# EAP-FAST

only during the client's initial EAP-FAST authentication. Subsequent authentications rely on the provisioned credential and will usually omit the provisioning step.

EAP-FAST is an authentication protocol designed to address the performance shortcomings of prior TLS-based EAP methods while retaining features such as identity privacy and support for password-based protocols. The EAP-FAST protocol is described by the IETF draft *draft-cam-winget-eap-fast-00.txt*

You can use the `aregemd` test tool to confirm that the EAP services are properly configured and operational.

## Configuring EAP-FAST

### `aregemd`

#### Step 1

```
cd /Radius/Services
add eap-fast-service
```

#### Step 2

```
cd eap-fast-service
set type eap-fast
```

#### Step 3

```
set AuthorityIdentifier authority-identifier
```

#### Step 4

```
set AuthorityInformation authority-information
```

#### Step 5

```
eap-gtc-service
```

#### Step 6

```
set ProvisionService eap-mschapv2-service
```

```
[ //localhost/Radius/Services/eap-fast ]
Name = eap-fast
Description =
Type = eap-fast
IncomingScript~ =
OutgoingScript~ =
AuthorityIdentifier =
AuthorityInformation =
MaximumMessageSize = 1024
PrivateKeyPassword =
ServerCertificateFile =
ServerRSAKeyFile =
CACertificateFile =
CACertificatePath =
CRLDistributionURL =
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = True
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120
CredentialLifetime = Forever
AuthenticationService =
ProvisionMode = Anonymous
ProvisionService =
AlwaysAuthenticate = True
```

**Table 8-1 EAP-FAST Service Properties**

Property	Description
	Optional script CAR server runs when it receives a request from a client for EAP-FAST service.
OutgoingScript	Optional script CAR server runs before it sends a response to a client using EAP-FAST.
AuthorityIdentifier	A string that uniquely identifies the credential (PAC) issuer. The client uses this value to select the correct PAC to use with a particular server from the set of PACs it might have stored locally.  Ensure that the AuthorityIdentifier is globally unique and that it does not conflict with identifiers used by other EAP-FAST servers or PAC issuers.
AuthorityInformation	A string that provides a descriptive text for this credential issuer. The value can be displayed to the client for identification purposes and might contain the enterprise or server names.
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message can have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.

**EAP-FAST Service Properties (continued)**

ServerRSAKeyFile	<p>The full pathname of the file containing the server's RSA private key. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are "PEM" and "DER". If an encoding prefix is not present, the file is assumed to be in PEM format.</p> <p>The following example assumes that the subdirectory <b>pki</b> under <b>/cisco-ar</b> contains the server's certificate file. The file <b>server-key.pem</b> is assumed to be in PEM format. The file extension <b>.pem</b></p> <p style="text-align: center;"><b>set ServerRSAKeyFile PEM:/cisco-ar/pki/server-key.pem</b></p>
	<p>the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate.</p> <p>For example, if a certificate file named <b>ca-cert.pem</b> is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in <b>ca-cert.path.pem</b> is 1b96dd93, then a symbolic link named 1b96dd93 must point to <b>ca-cert.pem</b>.</p> <p>If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.</p>
CRLDistributionURL	<p>Optional. Enter the URL that CAR should use to retrieve the CRL. You can specify a URL that uses HTTP or LDAP.</p> <p>The following is an example for an HTTP URL:  <code>&lt;http://crl.verisign.com/pca1.1.1.crl&gt;</code></p> <p><code>ldap://209.165.200.225:388/CN=development-CA,CN=acs-westcoast2,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=cisco,DC=com</code></p>

	not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	<p>If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication.</p> <p>SessionTimeout is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following:</p> <p style="text-align: center;"><b>Set SessionTimeout “1 Hour 45 Minutes”</b></p>
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
CredentialLifetime	<p>Specifies the maximum lifetime of a Protected Access Credential (PAC). Clients that successfully authenticate with an expired PAC will be re-provisioned with a new PAC.</p> <p>CredentialLifetime is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. Credentials that never expire should be specified as Forever.</p>
AuthenticationService	Specifies the name of the EAP-GTC service is used for authentication. The named service must have the UseLabels parameter set to True.
ProvisionMode	Specifies the TLS mode used for provisioning. Clients only support the default Anonymous mode.
ProvisionService	Specifies the name of the EAP-MSChapV2 service used for provisioning.
AlwaysAuthenticate	Indicates whether provisioning should always automatically rollover into authentication without relying on a separate session. Most environments, particularly wireless, will perform better when this parameter is set to True, the default value.



Note

## EAP-FAST Keystores

`/Radius/Advanced/KeyStores/EAP-FAST`

```
RolloverPeriod = "1 Week"
```

**Table 8-2**      *KeyStores Properties*

Property	Description

## Testing EAP-FAST with radclient

Step 1

```
cd /cisco-ar/usrbin  
./radclient -s
```

Step 2

```
tunnel eap-mschapv2
```

Step 3

```
simple_eap_fast_test user-name password
```

**Step 4**

```
tunnel eap-gtc
```

**Step 5**

```
simple_eap_fast_test user-name password
```

---

```
simple_eap_fast_test
```

## PAC Provisioning

```
pac show
```

```
No PAC(s) available to show
```

```
tunnel eap-mschapv2
```

```
PEAP tunnel method is eap-mschapv2  
EAP-FAST tunnel method is eap-mschapv2
```

```
simple_eap_fast_test bob bob
```

```
EAP-FAST authentication status:  
[0x0e07] TLS authentication succeeded  
Response to EAP-FAST message was not an Access-Accept  
p012
```

```
pac show
```

```
PAC 1 version 1 (219 bytes)  
A-ID      : AR-4.0  
A-ID-Info : Cisco Systems Access Registrar  
I-ID      : bob  
Expires   : Never (0)  
Key#      : 12  
TLV 1     : PAC-Key (1) mandatory (32 bytes)  
TLV 2     : PAC-Opaque (2) mandatory (120 bytes)  
TLV 3     : PAC-Info (9) mandatory (51 bytes)
```

```
simple_eap_fast_test
```

```
pac show
```

## Authentication

### tunnel eap-gtc

```
PEAP tunnel method is eap-gtc
EAP-FAST tunnel method is eap-gtc
```

### simple\_eap\_fast\_test bob bob

```
EAP-FAST authentication status :
  [0x0e07] TLS authentication succeeded
SUCCESS : Correctly formatted Session Keys received from the server
p01e
```

In this example, the EAP\_FAST authentication using the PAC from the previous provisioning step succeeded. The AccessAccept packet received from CAR can be displayed to confirm that it contains the expected attributes including the MS-MPPE session keys.

## Parameters Used for Certificate-Based Authentication

EAP-FAST might optionally use RSA certificates to securely create the tunnel that is used for PAC provisioning. However, the Cisco client does not support the use of certificates and the following parameters will be ignored and should be left at their default values:

- PrivateKeyPassword
- ServerCertificateFile
- ServerRSAKeyFile
- CACertificateFile
- CACertificatePath
- ClientVerificationMode
- VerificationDepth
- EnableSessionCache
- SessionTimeout

The parameters for configuring certificate-based operation are identical to those used for PEAP and EAP-TLS.

[Table 8-3](#) describes the parameters used for certificate-based authentication.

**Table 8-3 Certificate-Based Authentication Parameters**

Parameter	Description
	the enterprise and/or server names.
MaximumMessageSize	Indicates the maximum length in bytes that a EAP-FAST message can have before it is fragmented. If certificates are not used for authentication, fragmentation should not be an issue.
AuthenticationTimeout	Indicates the maximum number of seconds before an authentication operation times out and is rejected.
CredentialLifetime	Specifies the maximum lifetime of a PAC (Protected Access Credential). Clients that successfully authenticate with an expired PAC will be re-provisioned with a new PAC.
AuthenticationService	Specifies the name of the EAP-GTC service that is used for authentication. The named service must have the UseLabels parameter set to True.
ProvisionMode	Specifies the TLS mode that is used for provisioning. As of this writing, clients only support the default Anonymous mode.
ProvisionService	Specifies the name of the EAP-MSChapV2 service that is used for provisioning.
AlwaysAuthenticate	Indicates whether provisioning should always automatically rollover into authentication without relying on a separate session. Most environments, particularly wireless, will perform better when this parameter is set to True (the default value).

## radclient Command Reference

### radclient

#### eap-trace

##### eap-trace

##### eap-trace *level*

##### eap-trace 4

## **eap-trace**

The following example sets the trace level to 5 for EAP-FAST only. The trace level for other EAP methods is not affected.

### **eap-trace eap-fast 5**



---

The **eap-trace** command is for client-side trace information only and is independent of the server trace level that can be set using **aregcmd**.

---

## **tunnel**

The **tunnel** command is used to specify the inner provisioning and authentication methods for EAP-FAST. The specified EAP method type must agree with the server's configured methods or authentication will fail.

### **tunnel eap-method**

For EAP-FAST provisioning, the only allowable tunnel method is eap-mchapv2. For EAP-FAST authentication, the only allowable tunnel method is eap-gtc.

## **simple\_eap\_fast\_test**

The arguments are passed to the inner authentication method as its authentication parameters. If a PAC is not present, the tunnel method should be eap-mschapv2 and provisioning will occur. If a PAC is present, the tunnel method should be eap-gtc and authentication will occur.

### **simple\_eap\_fast\_test username password**

There are also variants for the **simple** test command for other EAP methods as shown in the following examples:

### **simple\_eap\_mschapv2\_test bob bob**

### **simple\_eap\_gtc\_test bob bob**

## **pac**

The **pac** command is used display, save, and delete PACs that are received from the server during testing. **radclient** maintains a cache of PACs that it knows about and that can be used for authentication testing. The current PAC cache can be displayed with the **pac show** command. PACs created during a test session can be stored to files with the **pac save** command, and reloaded in another session with the **pac load** command. The contents of the PAC cache are completely deleted with **pac delete**. If the optional parameter cache is included, PACs are also erased from disk.

### **pac load | save | show { hex } | delete { cache }**

The **pac show** command displays the currently cached PACs. If the optional parameter *hex* is included, additional detailed information including hex dumps are included in the display output.

### **pac show { hex }**

The **pac load** command loads any previously saved PACS from disk into the active cache.

The **pac save** command saves all PACs from the active cache to disk. Any previously existing PACs for the same user will be over-written.

The **pac delete** command deletes all PACs from the active cache. If the optional cache parameter is included then PACs are also erased from disk.

```
pac delete { cache }
```

## PAC—Credential Export Utility

You can manually provision EAP-FAST PACs to clients and avoid the use of the protocol provisioning phase. This might be desirable from a security perspective since the default provisioning protocol uses an anonymous (unauthenticated) method to construct the tunnel used to download the PAC to the client.

Manual provisioning involves exporting a PAC from CAR to a file which is then copied to the client machine and used by the import utility. After a PAC has been manually imported, the client should be able to authenticate via EAP-FAST while bypassing the initial provisioning phase. Care should be taken while storing and transporting PAC files since they contain information that potentially allows a client to authenticate via EAP-FAST.

PACs are exported from CAR via the **pac** command which is a new utility for this release. (Note that this **pac** command is a stand-alone executable which is different from the Radclient **pac** command.) The **pac** command has two capabilities:

- Exports a PAC to a file
- Displays information about an existing PAC file

## PAC Export

Use the **pac export** command to create a new PAC file. In the following example, *eap-fast* is the name of the CAR service configured for EAP-FAST authentication, *bob* is the name of the user this PAC will be used for, and *password* is the password used to derive a key for encrypting the resulting file. (This password is not the same as the administrator's password). The PAC file will be named **bob.pac** by default. You can use the **-f** option to give the file a different name.

```
pac -s export eap-fast bob password
```

If you omit the password parameter, a default password will be used.



### Note

---

Using the default password is strongly discouraged for security reasons.

---

## PAC Display

Use the **pac show** command to display information about a PAC file. In the following example, **bob.pac** is the name of the PAC file and *password* is the password used to decrypt the file contents.

```
pac -s show bob.pac password
```

## Syntax Summary

The complete **pac** command syntax is as follows:

```
pac { options } export <service-name> <user-name> <file-password>
```

```
pac { options } show <file-name> file-<password>
```

Where:

- C <cluster>—Specifies the cluster to be used.
- N <user>—Specifies the user.
- P <user-password>—Specifies the password to be used.
- s —Logs in using defaults
- v—Enables verbose output
- f—Exports file name (default = {user-name}.pac)

## EAP-GTC

EAP-GTC, defined in RFC 2284, is a simple method for transmitting a user's name and password to an authentication server. EAP-GTC should not be used except as an authentication method for PEAP Version 1 because the password is not protected.

## Configuring EAP-GTC

To enable EAP-GTC, use **aregcmd** to create and configure a service of type *eap-gtc*.

---

**Step 1** Launch **aregcmd** and create an EAP-GTC service.

```
cd /Radius/Services
add eap-gtc-service
```

**Step 2** Change directory to the service and set its type to *eap-gtc*.

```
cd eap-gtc-service
set type eap-gtc
```

The follow example shows the default configuration for an EAP-GTC service:

```
eap-gtc
```

[Table 8-4](#) lists and describes the EAP-GTC specific properties for EAP-GTC authentication.

**Table 8-4 EAP-GTC Properties**

Property	Description
UserService	Required; name of service that can be used to authenticate using cleartext passwords.
UserPrompt	Optional string the client might display to the user; default is "Enter password:." Use the <code>userprompt</code> command to change the prompt, as in the following:
UseLabels	Required; must be set to TRUE for EAP-FAST authentication and set to FALSE for PEAP authentication. Set to FALSE by default.

Set the service's UserService to local-users or another local authentication service that is able to authenticate using clear-text passwords.

**set UserService local-users**

If configuring for EAP-FAST, set the UseLabels property to TRUE.

To test the EAP-GTC service, launch **radclient** and use the **simple\_eap\_gtc\_test** command. The **simple\_eap\_gtc\_test** command sends an Access-Request for the designated user with the user's password.

The response packet should indicate an Access-Accept if authentication was successful. View the response packet to ensure the authentication was successful.

**simple\_eap\_gtc\_test bob bob**

```
Framed-Compression = VJ TCP/IP header compression
Framed-IPX-Network = 1
EAP-Message = 03:01:00:04
Ascend-Idle-Limit = 1800
Message-Authenticator = d3:4e:b1:7e:2d:0a:ed:8f:5f:72:e0:01:b4:ba:c7:e0
```

## EAP-LEAP

Cisco AR 4.1 supports the new AAA Cisco-proprietary protocol called Light Extensible Authentication Protocol (LEAP), a proprietary Cisco authentication protocol designed for use in IEEE 802.11 wireless local area network (WLAN) environments. Important features of LEAP include:

- Mutual authentication between the network infrastructure and the user
- Secure derivation of random, user-specific cryptographic session keys
- Compatibility with existing and widespread network authentication mechanisms (e.g., RADIUS)
- Computational speed

**Note**


---

CAR supports a subset of EAP to support LEAP. This is not a general implementation of EAP for CAR.

---

The Cisco-Wireless or Lightweight Extensible Authentication Protocol is an EAP authentication mechanism where the user password is hashed based on an MD4 algorithm and verified by a challenge from both client and server.

## Configuring EAP-LEAP

To enable EAP-LEAP, use `radius service` to create and configure a service of type `eap-leap`. When you create an EAP-LEAP service type, you must also specify a `UserService` to perform AAA service. The `UserService` can be any configured authentication service.

---

**Step 1** Launch `radius service` and create an EAP-LEAP service.

**Step 2** Set the service type to `eap-leap`.

```
[ //localhost/Radius/Services/eap-leap-service ]
  Name = newone
  Description =
  Type =
  IncomingScript~ =
  OutgoingScript~ =
  AuthenticationTimeout = 120
  UserService =
```

**Step 3** Set the `UserService` property to a configured authentication service.

---

## EAP-MD5

CAR 4.2 supports EAP-MD5, or MD5-Challenge, another EAP authentication exchange. In EAP-MD5 there is a CHAP-like exchange and the password is hashed by a challenge from both client and server to verify the password is correct. After verified correct, the connection proceeds, although the connection is periodically re-challenged (per RFC 1994).

## Configuring EAP-MD5

Specify type `md5` when you create an EAP-MD5 service. When you create an EAP-MD5 service type, you must also specify a `UserService` to perform AAA service. The `UserService` can be any configured authentication service.

To enable EAP-MD5, use `enable eap md5` to create and configure a service of type `md5`. When you create an EAP-MD5 service type, you must also specify a `UserService` to perform AAA service. The `UserService` can be any configured authentication service.

---

**Step 1** Launch `configure` and create an EAP-MD5 service.

**Step 2** Set the service type to `md5`.

**Step 3** Set the `UserService` property to a configured authentication service.

---

## EAP-Negotiate

EAP-Negotiate is a special service used to select at run-time the EAP service to be used to authenticate the client. EAP-Negotiate is configured with a list of candidate EAP services that represent the allowable authentication methods in preference order. When an EAP session begins, the EAP-Negotiate service tries the first service in the list. If the client does not support that method, it will respond with an EAP-Nak message which triggers EAP-Negotiate to try the next method on the list until a valid method is found or the list is exhausted in which case authentication fails.

EAP-Negotiate is useful when the client population has deployed a mix of different EAP methods that must be simultaneously supported by CAR. It can be difficult or impossible to reliably distinguish which clients require which methods simply by examining RADIUS attributes or other packet properties. EAP-Negotiate solves this problem by using the method negotiation feature of the EAP protocol. Negotiation can be used to select the primary EAP method used for authentication and also to select the inner method for PEAP.

## Configuring EAP-Negotiate

To enable EAP-Negotiate, first use `configure eap` to create and configure the EAP services that will be used for authentication, then create and configure a service of type `eap-negotiate`.

---

**Step 1** Launch `configure eap` and create an EAP-LEAP service.

**Step 2** Set the service type to `leap`.

**Step 3** Set the `ServiceList` property to a list of pre-configured EAP authentication services.

The `ServiceList` property lists the names of the EAP services that can be negotiated with this instance of EAP-Negotiate. The `ServiceList` property is a space-separated list and must consist of valid EAP service name, *not service types*, in preference order from left to right. Each service and type on the list must be unique; duplicates are not allowed.

```
set ServiceList "eap-leap-service eap-md5-service peap-v1-service"
```

---

## Negotiating PEAP Tunnel Services

EAP-Negotiate can also be used to negotiate the inner tunnel service used for phase two of PEAP-V0 or PEAP-V1. To do this, create and configure a service of type `eap-negotiate`. The `ServiceList` can only contain services that are legal for the version of PEAP that it is used with. Set the PEAP service's `TunnelService` parameter to the name of the `eap-negotiate` service.



### Note

Not all supplicants support negotiation of the PEAP inner method. EAP-Negotiate can only be used with supplicants that can use EAP-Nak to reject an unsupported inner method.

---

## Testing EAP-Negotiate with radclient

You can test EAP-Negotiate using the same **radclient** commands used to test the other EAP services. For example, you can use the commands for testing `eap-leap` and `peap-v1`.

# EAP-MSChapV2

EAP-MSChapv2 is based on **draft-kamath-pppext-eap-mschapv2-00.txt**, an informational IETF draft document. EAP-MSChapv2 encapsulates the MSChapV2 protocol (specified by RFC 2759) and can be used either as an independent authentication mechanism or as an inner method for PEAP Version 0 (recommended).

## Configuring EAP-MSChapV2

To enable EAP-MSChapv2, use **aregcmd** to create and configure a service of type *eap-mschapv2*.

---

**Step 1** Launch **aregcmd** and create an EAP-MSChapV2 service.

```
cd /Radius/Services
add eap-mschapv2
```



---

**Note** This example named the service *eap-mschapv2*, but you can use any valid name for your service.

---

**Step 2** Set the service's type to *eap-mschapv2*.

```
cd eap-mschapv2
set Type eap-mschapv2
```

**Step 3** Set the service's *UserService* to *local-users* or another local authentication service that is able to authenticate using MSChapV2.

```
set UserService local-users
```

**Step 4** You might (optionally) set a string for System ID that identifies the sender of the MSChapV2 challenge message, as in the following:

```
set SystemID system_ID_string
```

---

## Testing EAP-MSChapV2 with radclient

To test the EAP-MSChapVersion 2 service using **radclient**, perform the following steps:

- 
- Step 1** Launch **radclient**.
- Step 2** Use the **simple\_eap\_mschapv2\_test** command to authenticate using EAP-MSChapV2, as in the following:

```
simple_eap_mschapv2_test bob bob
```

The **simple\_eap\_mschapv2\_test** command above sends an Access-Request for user bob with the user's password. The response packet should indicate an Access-Accept if authentication was successful.

- Step 3** View the response packet to ensure the authentication was successful.

```
p006
```

```
Access-Accept
```

```
EAP-Message
```

---

## EAP-SIM

CAR 4.2 supports EAP-SIMv16. In a GSM network a subscriber is issued a *smart card* called the subscriber identity module (SIM) that contains a secret key (Ki) and an International Mobile Subscriber Identity (IMSI). The key (Ki) is also stored in the GSM authentication center located with the Home Location Registry (HLR).

An access point uses the CAR RADIUS server to perform EAP-SIM authentication of mobile clients. CAR must obtain authentication information from the HLR. CAR contacts the MAP gateway that performs the MAP protocol over SS7 to the HLR.

## Configuring EAP-SIM

To enable EAP-SIM authentication, use **aregcmd** to create and configure a service of type *eap-sim*.

- 
- Step 1** Launch **aregcmd** and create an EAP-TLS service.
- ```
cd /Radius/Services
```
- ```
add eap-sim-service
```
- Step 2** Change directory to the service and set its type to *eap-sim*.

```
cd eap-sim-service
set Type eap-sim
```

```
PseudonymRenewtime = "24 Hours"
PseudonymLifetime = Forever
EnableReauthentication = False
MaximumReauthentications = 16
ReauthenticationTimeout = 3600
ReauthenticationRealm =
TripletCacheTimeout = 120
AuthenticationTimeout = 120
UseProtectedResults = True
RemoteServers/
```

**Table 8-5** EAP-SIM Service Properties

Property	Description
IncomingScript~	Optional script CAR server runs when it receives a request from a client for an EAP-SIM service.
OutgoingScript~	Optional script CAR server runs before it sends a response to a client using an EAP-SIM service.
OutageScript~	Optional. If set to the name of a script, CAR runs the script when an outage occurs. This property allows you to create a script that notifies you when the RADIUS server detects a failure.
MultipleServersPolicy	Required. Must be set to either Failover or RoundRobin.  When set to Failover, CAR directs requests to the first server in the list until it determines the server is off-line. At which time, CAR redirects all requests to the next server in the list until it finds a server that is on-line.  When set to RoundRobin, CAR directs each request to the next server in the RemoteServers list in order to share the resource load across all of the servers listed in the RemoteServers list.
NumberOfTriplets	Number of triplets (1, 2, or 3) to use for authentication; default is 2.
UseSimDemoTriplets	Set to TRUE to enable the use of demo triplets. This must be disabled for release builds.

Table 8-5 EAP-SIM Service Properties (continued)

Property	Description
AlwaysRequestIdentity	When True, enables the server to obtain the subscriber's identity via EAP/SIM messages instead of relying on the EAP messages alone. This might be useful in cases where intermediate software layers can modify the identity field of the EAP-Response/Identity message. The default value is False.
EnableIdentityPrivacy	When True, the identity privacy feature is enabled. The default value is False.
PseudonymSecret	The secret string that is used as the basis for protecting identities when identity privacy is enabled. This should be at least 16 characters long and have a value that is impossible for an outsider to guess. The default value is secret.  <b>Note</b> It is very important to change PseudonymSecret from its default value to a more secure value when identity privacy is enabled for the first time.
PseudonymRenewtime	Specifies the maximum age a pseudonym can have before it is renewed. When the server receives a valid pseudonym that is older than this, it generates a new pseudonym for that subscriber. The value is specified as a string consisting of pairs of numbers and units, where the units might be of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. The default value is "24 Hours".  Examples are: "8 Hours", "10 Hours 30 Minutes", "5 D 6 H 10 M"
PseudonymLifetime	Specifies the maximum age a pseudonym can have before it is rejected by the server, forcing the subscriber to authenticate using its permanent identity. The value is specified as a string consisting of pairs of numbers and units, where the units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. It can also be Forever, in which case, pseudonyms do not have a maximum age. The default value is "Forever".  Examples are: "Forever", "3 Days 12 Hours 15 Minutes", "52 Weeks"
EnableReauthentication	When True, the fast re-authentication option is enabled. The default value is False.
MaximumReauthentications	Specifies the maximum number of times a re-authentication identity might be reused before it must be renewed. The default value is 16.
ReauthenticationTimeout	Specifies the time in seconds that re-authentication identities are cached by the server. Subscribers that attempt to re-authenticate using identities that are older than this value will be forced to use full authentication instead. The default value is 3600 (one hour).
ReauthenticationRealm	This information will be supplied later.
TripletCacheTimeout	Time in seconds an entry remains in the triplet cache. A zero (0) indicates that triplets are not cached. The maximum is 28 days; the default is 0 (no caching).

**Table 8-5** EAP-SIM Service Properties (continued)

Property	Description
AuthenticationTimeout	Time in seconds to wait for authentication to complete. The default is 2 minutes; range is 10 seconds to 10 minutes.
UseProtectedResults	Enables or disables the use of protected results messages. Results messages indicate the state of the authentication but are cryptographically protected.
RemoteServers/	List of remote RADIUS servers which are map gateways. The remote server type must be set to map-gateway.

**Note**

The EAP-SIM property `OutagePolicy` present in earlier versions of CAR is no longer part of the EAP-SIM configuration.

## EAP-Transport Level Security (TLS)

EAP-Transport Level Security (EAP-TLS), described in RFC 2716, is an authentication method designed to mitigate several weaknesses of EAP. EAP-TLS leverages TLS, described in RFC 2246, to achieve certificate-based authentication of the server and (optionally) the client. EAP-TLS provides many of the same benefits as PEAP but differs from it in the lack of support for legacy authentication methods.

## Configuring EAP-TLS

To enable EAP-TLS authentication, use **aregcmd** to create and configure a service of type `eap-tls`.

**Step 1** Launch **aregcmd** and create an EAP-TLS service.

```
cd /Radius/Services
add eap-tls-service
```

**Step 2** Change directory to the service and set its type to `eap-tls`.

```
cd eap-tls-service
set Type eap-tls
```

```
[ //localhost/Radius/Services/eap-tls-service ]
Name = eap-tls
Description =
Type = eap-tls
IncomingScript~ =
OutgoingScript~ =
MaximumMessageSize = 1024
PrivateKeyPassword =
ServerCertificateFile =
ServerRSAKeyFile =
CACertificateFile =
```

```

CACertificatePath =
CRLDistributionURL =
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = True
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120

```

Table 8-6 describes the EAP-TLS configuration properties:

**Table 8-6 EAP-TLS Service Properties**

Property	Description
IncomingScript	Optional script CAR server runs when it receives a request from a client for EAP-TLS service
OutgoingScript	Optional script CAR server runs before it sends a response to a client using EAP-TLS
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message can have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.
ServerRSAKeyFile	<p>The full pathname of the file containing the server's RSA private key. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are "PEM" and "DER". If an encoding prefix is not present, the file is assumed to be in PEM format.</p> <p>The following example assumes that the subdirectory <b>pki</b> under <b>/cisco-ar</b> contains the server's certificate file. The file <b>server-key.pem</b> is assumed to be in PEM format. The file extension is not significant.</p> <p style="text-align: center;"><b>set ServerRSAKeyFile PEM:/cisco-ar/pki/server-key.pem</b></p>
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file can contain more than one certificate, but all certificates must be in PEM format. DER encoding is not allowed.

Table 8-6 EAP-TLS Service Properties (continued)

Property	Description
CACertificatePath	<p>The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references.</p> <p>Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate.</p> <p>For example, if a certificate file named <b>ca-cert.pem</b> is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in <b>ca-cert.path.pem</b> is 1b96dd93, then a symbolic link named 1b96dd93 must point to <b>ca-cert.pem</b>.</p> <p>If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.</p>
CRLDistributionURL	<p>Optional. The URL that CAR should use to retrieve the CRL. You can specify a URL that uses HTTP or LDAP.</p> <p>The following is an example for an HTTP URL: &lt;http://crl.verisign.com/pca1.1.1.crl&gt;.</p> <p>The following is an example for an LDAP URL: ldap://209.165.200.225:388/CN=development-CA,CN=acs-westcoast2,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=cisco,DC=com</p>
ClientVerificationMode	<p>Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional.</p> <ul style="list-style-type: none"> <li>• RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one.</li> <li>• None will not request a client certificate.</li> <li>• Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.</li> </ul>
VerificationDepth	<p>Specifies the maximum length (<b>in bytes?</b>) of the certificate chain used for client verification.</p>
EnableSessionCache	<p>Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.</p>

**Table 8-6** EAP-TLS Service Properties (continued)

Property	Description
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication.  SessionTimeout is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following:  <b>Set SessionTimeout “1 Hour 45 Minutes”</b>
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.

**Note**

CAR 4.2 verifies the certificate during the TLS-based authentication. CRL validation is done before accepting a client certificate during the TLS authentication.

## Testing EAP-TLS with radclient

To test the EAP-TLS service, launch **radclient** and use the **simple\_eap\_tls\_test** command, as in the following:

```
simple_eap_tls_test arg1
```

The argument is arbitrary for the **simple\_eap\_tls\_test** command and can be anything. (In the future, the argument can be used to select a client certificate.)

## Testing EAP-TLS with Client Certificates

You can test EAP-TLS using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in **/cisco-ar/pki** and be named **client-cert.pem** and **client-key.pem** respectively. Both files must be in PEM format.

## EAP-TTLS

CAR supports the Extensible Authentication Protocol Tunneled TLS (EAP-TTLS). EAP-TTLS is an EAP protocol that extends EAP-TLS. In EAP-TLS, a TLS handshake is used to mutually authenticate a client and server. EAP-TTLS extends this authentication negotiation by using the secure connection established by the TLS handshake to exchange additional information between client and server.

EAP-TTLS leverages TLS (RFC 2246) to achieve certificate-based authentication of the server (and optionally the client) and creation of a secure session that can then be used to authentication the client using a legacy mechanism. EAP-TTLS provides several benefits:

- Industry standard authentication of the server using certificates (TLS)
- Standardized method for session key generation using TLS PRF
- Strong mutual authentication

- Identity privacy
- Fast reconnect using TLS session caching
- EAP message fragmentation
- Secure support for legacy client authentication methods

EAP-TTLS is a two-phase protocol. Phase 1 conducts a complete TLS session and derives the session keys used in Phase 2 to securely tunnel attributes between the server and the client. The attributes tunneled during Phase 2 can be used to perform additional authentication(s) via a number of different mechanisms.

The authentication mechanisms that can be used during Phase 2 include PAP, CHAP, MS-CHAP, MS-CHAPv2, and EAP. If the mechanism is EAP, then several different EAP methods are possible.

The Phase 2 authentication can be performed by the local AAA server (the same server running EAP-TTLS) or it can be forwarded to another server (known as the home AAA server). In the latter case, the home server has no involvement in the EAP-TTLS protocol and can be any AAA service that understands the authentication mechanism in use and is able to authenticate the user. It is not necessary for the home server to understand EAP-TTLS.

## Configuring EAP-TTLS

Configuring EAP-TTLS involves two major tasks:

1. Configuring the TLS parameters used for Phase 1
2. Selecting the Phase 2 authentication methods and specifying whether authentication is performed locally or forwarded to the home server.

If authentication is forwarded, the configuration must include the identity of the remote home server and its shared secret.

You configure EAP-TTLS using the **aregcmd** CLI to create the appropriate services and specify their parameters. Use the **radclient** test tool to confirm that the services have been properly configured and are operational.

## Creating an EAP-TTLS Service

To enable EAP-TTLS authentication, use **aregcmd** to create and configure a service of type *eap-ttls*.

---

**Step 1** Launch **aregcmd** and create an EAP-TTLS service.

```
cd /Radius/Services
```

```
add eap-ttls-service
```

**Step 2** Change directory to the service and set its type to eap-ttls.

```
cd eap-ttls-service
```

```
set Type eap-ttls
```

```
[ //localhost/Radius/Services/eap-ttls-service ]
Name = eap-ttls
Description =
Type = eap-ttls
```

```

IncomingScript~ =
OutgoingScript~ =
MaximumMessageSize = 1024
PrivateKeyPassword =
ServerCertificateFile =
ServerRSAKeyFile =
CACertificateFile =
CACertificatePath =
CRLDistributionURL =
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = True
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120
AuthenticationService =

```

Table 8-7 describes the EAP-TTLS configuration properties:

**Table 8-7 EAP-TTLS Service Properties**

Property	Description
IncomingScript	Optional script CAR server runs when it receives a request from a client for EAP-TTLS service.
OutgoingScript	Optional script CAR server runs before it sends a response to a client using EAP-TTLS.
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message can have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.
ServerRSAKeyFile	<p>The full pathname of the file containing the server's RSA private key. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are "PEM" and "DER". If an encoding prefix is not present, the file is assumed to be in PEM format.</p> <p>The following example assumes that the subdirectory <b>pki</b> under <b>/cisco-ar</b> contains the server's certificate file. The file <b>server-key.pem</b> is assumed to be in PEM format. The file extension is not significant.</p> <pre>set ServerRSAKeyFile PEM:/cisco-ar/pki/server-key.pem</pre>
CACertificateFile	<p>The full pathname of the file containing trusted CA certificates used for client verification. The file can contain more than one certificate, but all certificates must be in PEM format.</p> <p><b>Note</b> DER encoding is not allowed.</p>

Table 8-7 EAP-TTLS Service Properties (continued)

Property	Description
CACertificatePath	<p>The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if used, there are some special preparations required for the directory it references.</p> <p>Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate.</p> <p>For example, if a certificate file named <b>ca-cert.pem</b> is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in <b>ca-cert.path.pem</b> is 1b96dd93, then a symbolic link named 1b96dd93 must point to <b>ca-cert.pem</b>.</p> <p>If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.</p> <p>See <a href="#">rehash-ca-certs Utility</a>, page 8-32 for information about how to create the required certificate file hash links.</p>
CRLDistributionURL	<p>Optional. The URL that CAR should use to retrieve the CRL. You can specify a URL that uses HTTP or LDAP.</p> <p>The following is an example for an HTTP URL: &lt;http://crl.verisign.com/pca1.1.1.crl&gt;.</p> <p>The following is an example for an LDAP URL: ldap://209.165.200.225:388/CN=development-CA,CN=acs-westcoast2,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=cisco,DC=com</p>
ClientVerificationMode	<p>Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional.</p> <ul style="list-style-type: none"> <li>• RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one.</li> <li>• None will not request a client certificate.</li> <li>• Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.</li> </ul>
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.

Table 8-7 EAP-TTLS Service Properties (continued)

Property	Description
SessionTimeout	<p>If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and require a subsequent full authentication.</p> <p>SessionTimeout is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following:</p> <p style="text-align: center;"><b>Set SessionTimeout “1 Hour 45 Minutes”</b></p>
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out. The default is 120.
AuthenticationService	<p>Mandatory; specifies the authentication service to use to authenticate users. See <a href="#">Configuring an EAP-TTLS Authentication Service</a> for more information.</p> <p><b>Note</b> The authentication service must exist before you can save the EAP-TTLS service configuration.</p>

**Note**

CAR 4.2 verifies the certificate during the TLS-based authentication. CRL validation is done before accepting a client certificate during the TLS authentication.

## Configuring an EAP-TTLS Authentication Service

The EAP-TTLS service can authenticate users by with either a legacy method such as PAP, CHAP, MSCHAP, or MSCHAPv2 or with an EAP method such as EAP-MSCHAPv2 or EAP-GTC. The authentication can be performed by the local server (the same server running EAP-TTLS) or it can be forwarded to a remote AAA server (the home server for the user’s domain).

This section provides examples of several different ways to configure an EAP-TTLS authentication service. The following examples assume that you are using aregcmd and have already created the EAP-TTLS service.

**Note**

After you make a configuration change, you must save the configuration before it can be used.

### Authenticating Local Users with a Legacy Method

You can use a service like the local-users service (created as part of the example configuration) to authenticate users in the local UserList.

```
set AuthenticationService local-users
```

This service can be used to authenticate using PAP, CHAP, MSCHAP, and MSCHAPv2.

### Authenticating Users with EAP-MSChapV2

This example uses a service named eap-mschapv2 for authentication. Attempts to authenticate using any other method than EAP-MSChapV2 (assuming the service type is also eap-mschapv2) will fail.

```
set AuthenticationService eap-mschapv2
```

### Authenticating Users with EAP Negotiate

You can use the EAP-negotiate method to authenticate using more than one EAP type. The following example defines an EAP service named eap-negotiate that can negotiate EAP-MSChapV2 or EAP-GTC then configures an EAP-TTLS service to authenticate using that service.

---

**Step 1** Create a service of type *eap-negotiate*.

```
cd /Radius/Services
add eap-nego
cd eap-nego
set Type eap-negotiate
set ServiceList "eap-mschapv2 eap-gtc"
```

**Step 2** Configure the EAP-TTLS AuthenticationService.

```
cd /Radius/Services/eap-ttls
set AuthenticationService eap-nego
```

---

### Authenticating Users with Legacy and EAP Methods

You can configure EAP-TTLS to authenticate using both legacy and EAP methods with a Group service using an OR result rule. A configuration like that shown in the following example first attempts to authenticate with the eap-negotiate service. If that fails, the server attempts to authenticate with the local-users service.

---

**Step 1** Create the Group service

```
cd /Radius/Services
add local-or-eap
cd local-or-eap
set Type group
set ResultRule OR
cd GroupServices
add 1 eap-negotiate
add 2 local-users
```

**Step 2** Configure the EAP-TTLS AuthenticationService.

```
cd /Radius/Services/eap-ttls
set AuthenticationService local-or-eap
```

---

### Authenticating Using a Remote AAA Server

You can configure an EAP-TTLS service to forward authentication to a remote AAA server known (or the home server). The following configures a RADIUS service to use a remote server, then configures EAP-TTLS to use that service for authentication.

The first step in the following example configures a remote RADIUS server (aaa-remote) with its IP address and the shared secret that it shares with the local server. You might also specify other important parameters such as ports, timeouts, and maximum number of retries. See [Services, page 4-11](#), for information about configuring RADIUS services.

---

**Step 1** Configure a remote AAA server.

```
cd /Radius/RemoteServers
add aaa-remote
cd aaa-remote
set Protocol Radius
set IPAddress 10.1.2.3
set SharedSecret secret
```

The following step configures a RADIUS service to use the remote server created in the previous step. You might also configure other important parameters such as the failover strategy. See [Services, page 4-11](#), for information about configuring RADIUS services.

**Step 2** Configure an AAA service.

```
cd /Radius/Services
add home
cd home
set Type Radius
cd RemoteServers
add 1 aaa-remote
```

**Step 3** Configure the EAP-TTLS AuthenticationService:

```
cd /Radius/Services/eap-ttls
set AuthenticationService home
```

---

Other configurations are also possible. For example, a group service can be used to perform some authentications locally and forward others to a remote server.

## Testing EAP-TTLS with radclient

To test the EAP-TLS service, launch **radclient** and use the **simple\_eap\_ttls\_test** command. The **simple\_eap\_ttls\_test** command has the following syntax:

```
simple_eap_ttls_test
```

Where:

*identity* is the user's name.

*password* is the user's password

*method* is one of: PAP, CHAP, MSChap, MSChapV2, or PEAP.



---

**Note** If the method parameter is EAP, the **tunnel** command must be used to specify the EAP method type.

---

## Testing EAP-TTLS Using Legacy Methods

The following example uses EAP-TTLS with PAP as the Phase 2 method to authenticate a user named bob whose password is bob (from the example configuration).

---

**Step 1** Launch **radclient**.

```
cd /cisco-ar/usrbin  
./radclient -s
```

**Step 2** Authenticate using EAP-TTLS PAP.

```
simple_eap_ttls_test bob bob pap
```

The following commands show how to test the other valid legacy methods.

```
simple_eap_ttls_test bob bob chap  
simple_eap_ttls_test bob bob mschap  
simple_eap_ttls_test bob bob mschapv2
```

---

## Testing EAP-TTLS Using EAP Methods

The following example uses EAP-TTLS with EAP-MSChapV2 as the Phase 2 method to authenticate a user named bob whose password is bob (from the example configuration). Issue the **tunnel** command to specify the Phase 2 EAP method, then issue the **simple\_eap\_ttls\_test** command with eap as a method type.

---

**Step 1** Launch **radclient**

```
cd /cisco-ar/usrbin
```

```
./radclient -s
```

**Step 2** Authenticate using EAP-TTLS and EAP-MSChapV2.

```
tunnel eap-mschapv2
```

```
simple_eap_ttls_test bob bob eap
```

To test with a different EAP method, use the **tunnel** command to specify the method as shown in the following command to specify EAP-TLS.

```
tunnel eap-tls
```

```
simple_eap_ttls_test bob bob eap
```

---

## rehash-ca-certs Utility

The **rehash-ca-certs** utility works with the `CACertificatePath` property and enables you to create the required certificate file hash links (similar to those used with PEAP and EAP-TLS). The **rehash-ca-certs** utility is only used when the server is validating certificates from the client (which is optional and not a common case for EAP-TTLS).

The syntax for the **rehash-ca-certs** utility is:

```
rehash-ca-certs { -v } path1 { path2 ... pathn }
```

Each directory path specified on the command line is scanned by the **rehash-ca-certs** utility for filenames with the **pem** extension (such as **ca-cert.pem**) and the appropriate hash link is created as described above. Before creating links, **rehash-ca-certs** first removes all existing links in the directory, so each invocation creates fresh links. The `-v` option enables verbose output.

The following is an example of the **rehash-ca-certs** utility:

```
./rehash-ca-certs ../pki
```

The **rehash-ca-certs** utility warns about PEM files that do not contain certificates.

On Access Registrar, intermediate/chained certificates cannot be imported. Perform the following to run CAR on Solaris with PEAP authentication:

---

**Step 1** Add both root and intermediate CA in the directory **/opt/CSCOAr/pki** (as configured for `CACertificatePath` in the service `NYU-NetIDs-PEAPService`).

**Step 2** Change the directory to `pki`:

```
cd /opt/CSCOAr/pki
```

**Step 3** run `/opt/CSCOar/bin/rehash-ca-certs`

**Step 4** Stop ARserver and restart.

---

## radclient Command Reference

This section provides a summary of the **radclient** commands you can use to test PEAP and EAP-TLS.

### eap-trace

Use the **eap-trace** command to display additional client protocol trace information for EAP methods. Set the level to a number from 1 to 5 inclusively. Level 5 shows detailed hexadecimal dumps of all messages. Level 4 shows a message trace without hexadecimal dumps. Levels 3 and below show status and error information. To turn off trace displays, set the level to 0.

Use **eap-trace level** to set the trace level for all EAP methods. The following example command sets the trace level to 4 for all EAP methods:

```
eap-trace 4
```

Use **eap-trace method level** to set the trace level for the specified EAP method. The following example command sets the trace level to 5 for PEAP Version0 only. The trace level for other EAP methods is not affected.

```
eap-trace peap-v0 5
```



#### Note

---

The **eap-trace** command is for client-side trace information only and is independent of the server trace level you set using **aregcmd**.

---

### tunnel

Use the **tunnel** command to specify the inner authentication method for PEAP. The specified EAP method type must agree with the server's configured authentication method or authentication will fail.

```
tunnel eap-method
```

For PEAP Version 0, the allowable tunnel methods are EAP-MSCHAPV2 and EAP-SIM. For PEAP Version 1, the allowable tunnel methods are EAP-GTC and EAP-SIM.

```
simple_eap_mschapv2_test username password
```

```
simple_eap_gtc_test username password
```

```
simple_eap_peapv0_test arg1 arg2
```

The arguments are passed to the inner authentication method as its authentication parameters. For EAP-MSChapv2 the arguments are username and password; for EAP-SIM they are IMSI and key.

```
simple_eap_peapv1_test arg1 arg2
```

The arguments are passed to the inner authentication method as its authentication parameters. For EAP-GTC the arguments are username and password; for EAP-SIM they are IMSI and key.

```
simple_eap_tls_test arg1
```

## Protected EAP

Protected EAP (PEAP) is an authentication method designed to mitigate several weaknesses of EAP. PEAP leverages TLS (RFC 2246) to achieve certificate-based authentication of the server (and optionally the client) and creation of a secure session that can then be used to authenticate the client. PEAP provides several benefits:

- Industry standard authentication of the server using certificates (TLS)
- Standardized method for session key generation using TLS PRF
- Strong mutual authentication
- Identity privacy
- Fast reconnect using TLS session caching
- EAP message fragmentation
- Secure support for legacy client authentication methods

CAR 4.2 supports the two major existing variants of PEAP, PEAP Version 0 (Microsoft PEAP) and PEAP Version 1 (Cisco PEAP). PEAP Version 0 is described in IETF drafts **draft-kamath-pppext-peapv0-00.txt** and **draft-josefsson-pppext-eap-tls-eap-02.txt**. This version of PEAP can use either EAP-MSChapV2 or EAP-SIM as an authentication method. PEAP Version 1 is described by IETF draft **draft-zhou-pppext-peapv1-00.txt**. PEAP Version 1 can use either EAP-GTC or EAP-SIM as an authentication method.

## PEAP Version 0

This section describes configuring PEAP Version 0 and testing it with **radclient**.

### Configuring PEAP Version 0

To enable PEAP Version 0, use **aregcmd** to create and configure a service of type *peap-v0*.

---

**Step 1** Launch **aregcmd** and create a PEAP Version 0 service.

```
cd /Radius/Services
```

```
add peap-v0-service
```

**Step 2** Set the service's type to *peap-v0*.

```
cd peap-v0-service
```

```
set Type peap-v0
```

- Step 3** Set the service's TunnelService property to the name of an existing EAP-MSCHAPV2 or EAP-SIM service.

```
set TunnelService name_of_EAP-MSCHAPv2_service
```

*or*

```
set TunnelService name_of_EAP-SIM_service
```

[Table 8-8](#) describes the PEAP service properties for PEAP Version 0.

**Table 8-8** PEAP Version 0 Service Properties

Property	Description
IncomingScript	Optional script CAR server runs when it receives a request from a client for PEAP-v0 service.
OutgoingScript	Optional script CAR server runs before it sends a response to a client using PEAP-v0
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message can have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.

Table 8-8 PEAP Version 0 Service Properties (continued)

Property	Description
ServerCertificateFile	<p>The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.</p> <p>The following example assumes that the subdirectory <b>pki</b> under <b>/cisco-ar</b> contains the server's certificate file. The file <b>server-cert.pem</b> is assumed to be in PEM format; note that the file extension <b>.pem</b> is not significant.</p> <pre>set ServerCertificateFile PEM:/cisco-ar/pki/server-cert.pem</pre>
CACertificateFile	<p>The full pathname of the file containing trusted CA certificates used for client verification. The file can contain more than one certificate, but all certificates must be in PEM format. DER encoding is not allowed.</p>
CACertificatePath	<p>The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references.</p> <p>Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate.</p> <p>For example, if a certificate file name <b>ca-cert.pem</b> is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in <b>ca-cert.path.pem</b> is 1b96dd93, then a symbolic link named 1b96dd93 must point to the <b>ca-cert.pem</b> file.</p> <p>If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.</p>
CRLDistributionURL	<p>Optional. The URL that CAR should use to retrieve the CRL. You can specify a URL that uses HTTP or LDAP.</p> <p>The following is an example for an HTTP URL:</p> <p>The following is an example for an LDAP URL:</p>

Table 8-8 PEAP Version 0 Service Properties (continued)

Property	Description
ClientVerificationMode	<p>Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional.</p> <ul style="list-style-type: none"> <li>RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one.</li> <li>None will not request a client certificate.</li> <li>Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.</li> </ul>
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	<p>If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication.</p> <p>SessionTimeout is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following:</p> <p style="text-align: center;"><b>Set SessionTimeout “1 Hour 45 Minutes”</b></p>
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
TunnelService	Mandatory; must be the name of an existing EAP-MSCHAPv2 or EAP-SIM service for PEAP Version 0.
EnableWPS	When set to TRUE, enables Windows Provisioning Service (WPS) and provides two other properties, MasterURL and WPSGuestUserProfile. The default value is FALSE.
MasterURL	When using WPS, specifies the URL of the provisioning server which is modified with the appropriate fragment and sent to the client.
WPSGuestUserProfile	<p>When using WPS, specifies a profile to be used as a guest user profile; must be a valid profile under <b>/Radius/Profiles</b>.</p> <p>This profile is used for guests and users whose account has expired. This profile normally contains attributes denoting the VLAN-id of the guest network (which has the provisioning server alone) and might contain IP-Filters that would restrict the access of the guest (to only the provisioning server).</p>

**Note**

CAR 4.2 verifies the certificate during the TLS-based authentication. CRL validation is done before accepting a client certificate during the TLS authentication.

## Testing PEAP Version 0 with radclient

To test the PEAP Version 0, complete the following steps:

---

**Step 1** Launch **radclient**.

**Step 2** Specify the inner authentication method, `eap-mschapv2` or `eap-sim`, as in the following.

```
tunnel eap-mschapv2
```

*or*

```
tunnel eap-sim
```

**Step 3** Use the `simple_eap_peapv0_test` command to authenticate using PEAP Version 0, as in the following:

```
simple_eap_peapv0_test arg1 arg2
```

The `simple_eap_peapv0_test` command passes its arguments to the inner authentication mechanism which treats the arguments as either a username and a password (for `eap-mschapv2`) or as an IMSI and a key (for `eap-sim`).

---

The following example tests PEAP Version 0 with EAP-MSCHAPV2 as the inner authentication mechanism using username bob and password bob:

```
tunnel eap-mschapv2
```

```
simple_eap_peapv0_test bob bob
```

The following example tests PEAP Version 0 with EAP-SIM as the inner authentication mechanism using IMSI 1234567891 and key 0123456789ABCDEF:

```
tunnel eap-sim
```

```
simple_eap_peapv0_test 1234567891 0123456789ABCDEF
```

---

## Testing PEAP Version 0 with Client Certificates

You can test PEAP Version 0 using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in `/cisco-ar/pki` and be named `client-cert.pem` and `client-key.pem` respectively. Both files must be in PEM format.

## PEAP Version 1

This section describes configuring PEAP Version 1 and testing it with **radclient**.

### Configuring PEAP Version 1

To enable PEAP Version 1, use `aregcmd` to create and configure a service of type `peap-v1`.

**Step 1** Launch **aregcmd** and create a PEAP Version 1 service.

```
cd /Radius/Services
add peap-v1-service
```

**Step 2** Set the service's type to peap-v1.

```
cd peap-v1-service
set Type peap-v1
```

**Step 3** Set the service's TunnelService property to the name of an existing EAP-GTC or EAP-SIM service.

```
set TunnelService name_of_EAP-GTC_service

or

set TunnelService name_of_EAP-SIM_service
```

Table 8-9 describes the PEAP service properties for both PEAP Version 1.

**Table 8-9 PEAP Version 1 Service Properties**

Property	Description
IncomingScript	Optional script CAR server runs when it receives a request from a client for PEAP-v1 service.
OutgoingScript	Optional script CAR server runs before it sends a response to a client using PEAP-v1.
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message can have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.

Table 8-9 PEAP Version 1 Service Properties (continued)

Property	Description
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname can be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file can contain more than one certificate but all certificates must be in PEM format. DER encoding is not allowed.
CACertificatePath	<p>The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references.</p> <p>Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate.</p> <p>For example, if a certificate file named <b>ca-cert.pem</b> is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in <b>ca-cert.path.pem</b> is 1b96dd93, then a symbolic link named 1b96dd93 must point to the <b>ca-cert.pem</b> file.</p> <p>If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.</p>
CRLDistributionURL	<p>Optional. The URL that CAR should use to retrieve the CRL. You can specify a URL that uses HTTP or LDAP.</p> <p>The following is an example for an HTTP URL:</p> <p style="text-align: center;">.</p> <p>The following is an example for an LDAP URL:</p>
ClientVerificationMode	<p>Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional.</p> <ul style="list-style-type: none"> <li>RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one.</li> <li>None will not request a client certificate.</li> <li>Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.</li> </ul>
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.

**Table 8-9 PEAP Version 1 Service Properties (continued)**

Property	Description
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication.  SessionTimeout is specified as a string consisting of pairs of numbers and units, where units might be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following:  <b>Set SessionTimeout “1 Hour 45 Minutes”</b>
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
TunnelService	Mandatory; must be the name of an existing EAP-GTC or EAP-SIM service for PEAP Version 0.

## Testing PEAP Version 1 with radclient

To test the PEAP Version 1, complete the following steps:

- 
- Step 1** Launch **radclient**.
- Step 2** Specify the inner authentication method, EAP-GTC or EAP-SIM, as in the following.
- ```
tunnel eap-gtc
```
- or*
- ```
tunnel eap-sim
```
- Step 3** Use the **simple\_eap\_peapv1\_test** command to authenticate using PEAP Version 1, as in the following:
- ```
simple_eap_peapv1_test arg1 arg2
```

The **simple\_eap\_peapv1\_test** command passes its arguments to the inner authentication mechanism which treats the arguments as either a username and a password (for EAP-GTC) or as an IMSI and a key (for EAP-SIM).

---

## Testing PEAP Version 1 with Client Certificates

You can test PEAP Version 1 using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in **/cisco-ar/pki** and be named **client-cert.pem** and **client-key.pem** respectively. Both files must be in PEM format.

# CRL Support for Cisco Access Registrar

CAR checks for various certificates for validation purposes in its authentication services. The client sends a certificate along with the access-challenge to CAR. CAR verifies the validity of the certificate and approves the request if the certificate is valid. For certificate validation, CAR uses an advanced verification mechanism, which uses Certificate Revocation Lists (CRLs).

A CRL, which uses the X.509 certification format, is the signed data structure that the certificate authority (CA) issues periodically. It contains a list of the serial numbers and the timestamp of the revoked certificates. These revoked certificates are not valid and CAR rejects any request that comes with these certificates. The CRLs are available in a public repository in CAR.

A certificate can be revoked because of the following reasons:

- Expiration of the validity period.
- Change in the name of the user to whom the certificate is issued.
- Change in the association between the CA and the user.
- Loss of the private key that is associated with the certificate.

CAR uses the Lightweight Dynamic Authentication Protocol (LDAP) and HTTP for validating the certificates using CRL. The **CRLDistributionURL** in the TLS based EAP authentication services, is used for the CRL support in CAR. When you configure this property, CAR fetches the CRL from the specified URL, at the startup. A background thread in CAR keeps track of these CRLs. When any of the CRLs expires, CAR fetches the latest version of CRL using the specified URL. Each CRL contains the information related to its expiry.

CAR places all the CRLs in a CRL store. It uses these CRLs while it does a TLS authentication for certificate validation. During an authentication service, the certificate verifier in CAR checks for the validity of the certificate against the CRL issued by the CA that signed the certificate. It looks for the serial number of the certificate in the list of revoked certificates in the appropriate CRL. If it finds a match in the CRL, it compares the revocation time that is encoded in the CRL against the current time. If the current time is later than the revocation time, CAR considers the certificate invalid.

**Note**

CAR uses the **CRLDistributionURL** property in the following services:

```
eap-tls  
eap-ttls  
eap-fast  
peap-v0  
peap-v1
```

## Configuring Certificate Validation Using CRL

CAR uses the **CRLDistributionURL** property for the certificate validation using CRLs. The following shows a sample configuration for the certificate verification using CRLs in CAR:

```
CRLDistributionURL =
```

[Table 8-7](#) describes the properties in this sample configuration.

