Extensible Authentication Protocols

Cisco Prime Access Registrar (Prime Access Registrar) 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. Cisco Prime Access Registrar supports the following EAP authentication methods:

- EAP-AKA
- EAP-FAST
- EAP-GTC
- EAP-LEAP
- EAP-MD5
- EAP-Negotiate
- EAP-MSChapV2
- EAP-SIM
- EAP-Transport Level Security (TLS)
- EAP-TTLS
- Protected EAP
  - PEAP Version 0 (Microsoft PEAP)
  - PEAP Version 1 (Cisco PEAP)

In general, you enable each EAP method by creating and configuring a service of the desired type. Use the **radclient** test tool to confirm that the EAP service has been properly configured and is operational.

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. Cisco Prime Access Registrar supports the tunneling of other EAP methods within the PEAP phase two exchange.

EAP-AKA

Authentication and Key Agreement (AKA) is an EAP mechanism for authentication and session key distribution. It is used in the 3rd generation mobile networks Universal Mobile Telecommunications System (UMTS) and CDMA2000. AKA is based on symmetric keys, and typically runs in a UMTS Subscriber Identity Module (USIM), or a (Removable) User Identity Module ((R) UIM), similar to a smart card. EAP-AKA (Extensible Authentication Protocol Method for UMTS Authentication and Key Agreement) includes optional identity privacy support, optional result indications, and an optional fast reauthentication procedure.

In support of EAP-AKA, the following features are supported:

- support of MAP protocol over SIGTRAN
- support of HLR and/or HSS (3GPP compliant)
- Wx interface
- Support M3UA-SIGTRAN over IP

For more information on Wx Interface Support, see the Wx Interface Support for SubscriberDB Lookup, page 17-48.

Prime Access Registrar server supports migration to a converged IP Next Generation Networks (IP NGN) by supporting SS7 and SIGTRAN (SS7 over IP) for HLR communication to enable the seamlessly transition to next-generation IP-based signaling networks.

Prime Access Registrar supports M3UA-SIGTRAN to fetch the authentication vectors from HLR for EAP-AKA authentication, See SIGTRAN-M3UA for more information.

EAP-AKA is based on rfc-4187 (**http://www.ietf.org/rfc/rfc4187.txt**). This document specifies the details of the algorithms and messages.

This section contains the following topics:

Configuring EAP-AKA

You can use aregcmd to create and configure a service of type eap-aka.

Table 9-1 lists and describes the EAP-AKA service properties.

Table 9-1 EAP-AKA Service Properties

Property	Description
AlwaysRequestIdentity	When True, enables the server to obtain the subscriber's identity via EAP/AKA 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.

Table 9-1 EAP-AKA Service Properties (continued)

Property	Description
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. **Note** 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 it's 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 reauthentication option is enabled. The default value is False.
MaximumReauthentications	Specifies the maximum number of times a reauthentication identity might be reused before it must be renewed. The default value is 16.
ReauthenticationTimeout	Specifies the time in seconds that reauthentication identities are cached by the server. Subscribers that attempt to reauthenticate using identities that are older than this value will be forced to use full authentication instead. The default value is 3600 (one hour).
ReauthenticationRealm	Optional. If you configure the realm, this value is appended to the FastReauthenticationUserId.
AuthenticationTimeout	Time in seconds to wait for authentication to complete. The default is 2 minutes; range is 10 seconds to 10 minutes.
QuintetGenerationScript~	Optional. If the script is set, the custom scripting point can be used to read the quintets from a flat file or generate quintets instead of fetching the quintets from HLR.If the script is not set, the Prime Access Registrar sends the request to HLR configured in remote server to fetch the quintets.
UseProtectedResults	Enables or disables the use of protected results messages. Results messages indicate the state of the authentication but are cryptographically protected.
TripletCacheTimeout	Required; timeout value of triplet cache.

Table 9-1 EAP-AKA Service Properties (continued)

Property	Description
Subscriber_DBLookup	Required. Must be set to either DIAMETER or SIGTRAN-M3UA. When set to DIAMETER, the HSS lookup happens using the Diameter Wx Interface. You need to configure the DestinationRealm to send the Diameter packets to the RemoteServer. When set to SIGTRAN-M3UA, the HLR/HSS lookup happens using the SIGTRAN protocol. You need to configure the SIGTRAN remote server.
FetchAuthorizationInfo	Required. When set True, it fetches MSISDN from HLR. This field is dispayed when you set Subscriber_DBLookup as SIGT-RAN-M3UA.
IncomingScript~	Optional script Prime Access Registrar server runs when it receives a request from a client for an EAP-AKA/EAP-SIM service.
OutgoingScript~	Optional script Prime Access Registrar server runs before it sends a response to a client using an EAP-AKA/EAP-SIM service.
OutageScript~	Optional. If set to the name of a script, Prime Access Registrar runs the script when an outage occurs. This property allows you to create a script that notifies you when the server detects a failure.
RemoteServers	Remote server which can provide the service.

To enable EAP-AKA authentication:

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

cd /Radius/Services

add eap-aka-service

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

cd eap-aka-service

set Type eap-aka

The following example shows the default configuration for an EAP-AKA service:

```
[ //localhost/Radius/Services/test ]
    Name = test
    Description =
    Type = eap-aka
    AlwaysRequestIdentity = False
    EnableIdentityPrivacy = False
    PseudonymSecret = <encrypted>
    PseudonymRenewtime = "24 Hours"
    PseudonymLifetime = Forever
    Generate3GPPCompliantPseudonym = False
    EnableReauthentication = False
```

```
MaximumReauthentications = 16
ReauthenticationTimeout = 3600
ReauthenticationRealm =
AuthenticationTimeout = 120
QuintetGenerationScript~ =
UseProtectedResults = False
SendReAuthIDInAccept = False
SubscriberDBLookup = SIGTRAN-M3UA
FetchAuthorizationInfo = FALSE
MultipleServersPolicy = Failover
IncomingScript~ =
OutgoingScript~ =
OutageScript~ =
RemoteServers/
```

The following example shows the default configuration for an EAP-AKA Wx service:

```
[ //localhost/Radius/Services/eap-aka-wx ]
Name = eap-aka-wx
Description =
Type = eap-aka
AlwaysRequestIdentity = False
EnableIdentityPrivacy = False
PseudonymSecret = <encrypted>
PseudonymRenewtime = "24 Hours"
PseudonymLifetime = Forever
Generate3GPPCompliantPseudonym = False
EnableReauthentication = False
MaximumReauthentications = 16
ReauthenticationTimeout = 3600
ReauthenticationRealm =
AuthenticationTimeout = 120
QuintetGenerationScript~ =
UseProtectedResults = False
SendReAuthIDInAccept = False
SubscriberDBLookup = Diameter
DestinationRealm = mpc.com
PreRequestTranslationScript~ =
PostRequestTranslationScript~ =
PreResponseTranslationScript~ =
PostResponseTranslationScript~ =
```

Testing EAP-AKA with radclient

To test the EAP-AKA service, launch **radclient** and use the **simple_eap_aka_test** command. The **simple_eap_aka_test** command sends an Access-Request for the designated user with the user's secret key and sequence number.

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

simple_eap_aka_test bob secret 2

To test from radclient, you have to configure **/cisco-ar/conf/imsi.conf** file on radius server and reload the server. This file content should have imsi users in the format below:

```
<username>:<secret>:<sequence number>
```

For example:

```
bob:bob:1
```

EAP-FAST

Cisco Prime Access Registrar supports the EAP-FAST authentication method. EAP-FAST uses the EAP-MSChapV2 method for credential provisioning and EAP-GTC for authentication. Credential provisioning typically occurs 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*.

The EAP-FAST credential is known as a Protected Access Credential (PAC) and contains information used to secure the authentication operations. Parts of the PAC are encrypted by the server and are not visible to other entities. Clients are expected to securely store PACs locally for use during authentication.

Configuring EAP-FAST involves creating and configuring the required EAP-MSChapV2 and EAP-GTC services as well as the EAP-FAST service with the appropriate parameters.

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

This section contains the following topics:

Configuring EAP-FAST

You can use **aregcmd** to create and configure a service of type *eap-fast*.

To enable EAP-FAST:

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

cd /Radius/Services

add eap-fast-service

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

cd eap-fast-service

set type eap-fast

Step 3 Set the AuthorityIdentifier:

set AuthorityIdentifier ***authority-identifier***

Step 4 : Set the AuthorityInformation:

set AuthorityInformation ***authority-information***

Step 5 : Set the AuthentitcationService:

set AuthenticationService ***eap-gtc-service***

Step 6 :Set the ProvisionService:

set ProvisionService ***eap-mschapv2-service***

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

```
[ //localhost/Radius/Services/eap-fast-service ]
   Name = eap-fast-service
   Description =
   Type = eap-fast
   IncomingScript~ =
   OutgoingScript~ =
   MaximumMessageSize = 1024
   PrivateKeyPassword = <encrypted>
   ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
   ServerKeyFile = /opt/CSCOar/pki/server-key.pem
   CACertificateFile = /opt/CSCOar/pki/root-cert.pem
   CACertificatePath = /opt/CSCOar/pki
   CRLDistributionURL =
   ClientVerificationMode = Optional
   VerificationDepth = 4
   EnableSessionCache = true
   UseECCCertificates = true
   SessionTimeout = "5 Minutes"
   AuthenticationTimeout = 120
```

Table 9-2 lists and describes the EAP-FAST service properties.

Table 9-2 EAP-FAST Service Properties

Property	Description
IncomingScript	Optional script Prime Access Registrar server runs when it receives a request from a client for EAP-FAST service.
OutgoingScript	Optional script Prime Access Registrar 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.

Table 9-2 EAP-FAST 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.
ServerKeyFile	The full pathname of the file containing the server's RSA or ECC 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. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-key.pem** is assumed to be in PEM format. The file extension ***.pem*** is not significant. **set ServerKeyFile PEM:/cisco-ar/pki/server-key.pem**
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	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. 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. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to **ca-cert.pem**. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extension as in 1b96dd93.0 and 1b96dd93.1.
CRLDistributionURL	Optional. Enter the URL that Prime Access Registrar should use to retrieve the CRL.You can specify a URL that uses HTTP or LDAP. The following is an example for an HTTP URL: < `//crl.verisign.com/pca1.1.1.crl>`. 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`

Table 9-2 EAP-FAST Service Properties (continued)

Property	Description
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
UseECCCertificates	Determines the applicability of the authentication mechanism in SmartGrid Solutions, see the Smart Grid Solution Management, page 17-50 for more information. When UseECCCertificates is set to True, it can use the ECC, RSA, or combination of both certificate for certificate based verification. When UseECCCertificates is set to False, it can only use the RSA certificate for certificate based verification. The default location to fetch the certificate file is **/cisco-ar/pki**.
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: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
CredentialLifetime	Specifies the maximum lifetime of a Protected Access Credential (PAC). Clients that successfully authenticate with an expired PAC will be reprovisioned with a new PAC. 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.
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.

Table 9-2 EAP-FAST Service Properties (continued)

Property	Description
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 Prime Access Registrar verifies the certificate during the TLS-based authentication. CRL validation is done before accepting a client certificate during the TLS authentication.

EAP-FAST Keystores

The EAP-FAST service manages a set of keys used to protect the security and integrity of the PACs it issues. The keys are stored in**/Radius/Advanced/KeyStores/EAP-FAST** and are maintained automatically requiring minimal administration. Administrators can specify the maximum number of keys that are stored and the frequency of key updates.

The following is the default KeyStores settings:

```
[ //localhost/Radius/Advanced/KeyStores/EAP-FAST ]
        NumberOfKeys = 256
        RolloverPeriod = "1 Week"
```

Table 9-3 defines the KeyStores properties.

Table 9-3 KeyStores Properties

Property	Description
NumberOfKeys	Number (from 1-1024) that specifies the maximum number of keys stored for EAP-FAST.
RolloverPeriod	Specifies the amount of time between key updates.

Testing EAP-FAST with radclient

There are two distinct phases to testing EAP-FAST: provisioning and authentication. In the instructions below, Step 2 and Step 3 test provisioning and Steps 4 and Step 5 test authentication. At least one successful provisioning phase must be completed prior to testing authentication. Testing EAP-FAST with **radclient** requires that the EAP-MSChapV2 and EAP-GTC services be configured and functional.

The following instructions and examples assume that the AlwaysAuthenticate parameter has been set to False for testing purposes. This permits the provisioning and authentication steps to be tested separately. Most installations will set AlwaysAuthenticate to True for production use, and **radclient** works with that setting, but might display extra error messages that you can ignore.

To test EAP-FAST using **radclient**:

Step 1 Start **radclient**.

cd /cisco-ar/usrbin

./radclient –s

Step 2 Specify the inner provisioning method

tunnel eap-mschapv2

The only allowable method for provisioning is eap-mschapv2.

Step 3 Provision a new PAC:

simple_eap_fast_test user-name password

Step 4 Specify the inner authentication method.

tunnel eap-gtc

The only allowable method for authentication is eap-gtc.

Step 5 Authenticate using the PAC.

simple_eap_fast_test user-name password

The **simple_eap_fast_test** command passes its arguments to the inner authentication mechanism which in turn treats the arguments as a username and a password. The command in Step 3 should result in provisioning a new PAC, and Step 5 should result in successful authentication using that PAC.

PAC Provisioning

The following example provisions a PAC for user bob.

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      : Prime AR
A-ID-Info : Cisco Prime 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)
```

In this example the **simple_eap_fast_test** command indicates that it did not receive an AccessAccept. This is normal because the provisioning step always results in an AccessReject even when a new PAC has been successfully provisioned. The last **pac show** command displayed some status information from the new PAC and is used to verify that provisioning succeeded and authentication can now be tested. The PAC information displayed will vary and depends on how EAP-FAST is configured.

Authentication

The following example authenticates user bob (continuing from the PAC Provisioning example).

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 Prime Access Registrar 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 or ECC 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
- ServerKeyFile
- CACertificateFile
- CACertificatePath
- ClientVerificationMode
- VerificationDepth
- UseECCCertificates
- EnableSessionCache

- SessionTimeout

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

Table 9-4 describes the parameters used for certificate-based authentication.

Table 9-4 Certificate-Based Authentication Parameters

Parameter	Description
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. Care should be taken to ensure that the AuthorityIdentifier is globally unique, that is, is distinct from other PAC issuers
AuthorityInformation	A string that provides some descriptive text for this credential issuer. The value can be displayed to the client for identification purposes. It can contain 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 reprovisioned 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

This section describes the **radclient** commands you can use to test EAP-FAST.

eap-trace

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

Set the trace level for all EAP methods.

eap-trace ***level***

For example, the following command sets the trace level to 4 for all EAP methods.

eap-trace 4

Set the trace level for the specified EAP method.

eap-trace ***method level***

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

Note 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-mchavp2. 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 Prime Access Registrar 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 Prime Access Registrar via the **pac** command which is a new utility for this release. (Note that this pac command is a standalone 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 Prime Access Registrar 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.

This section contains the following topics:

Configuring EAP-GTC

Table 9-5 lists and describes the EAP-GTC specific properties for EAP-GTC authentication.

Table 9-5 EAP-GTC Properties

Property	Description
UserService	Required; name of service that can be used to authenticate using cleartext passwords.

Table 9-5 EAP-GTC Properties (continued)

Property	Description
UserPrompt	Optional string the client might display to the user; default is Enter password:" Use the **set** command to change the prompt, as in the following: **set UserPrompt "Admin Password:"**
UseLabels	Required; must be set to TRUE for EAP-FAST authentication and set to FALSE for PEAP authentication. Set to FALSE by default.

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:

```
[ //localhost/Radius/Services/eap-gtc-service ]
    Name = eap-gtc
    Description =
    Type = eap-gtc
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    UserService =
    UserPrompt = "Enter password:"
    UseLabels = False
```

Step 3 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

Step 4 If configuring for EAP-FAST, set the UseLabels property to TRUE.

Testing EAP-GTC with radclient

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

```
Packet: code = Access-Accept, id = 2, length = 104, attributes =
        Service-Type = Framed
        Framed-Protocol = PPP
        Framed-IP-Address = 192.168.0.0
        Framed-IP-Netmask = 255.255.255.0
        Framed-Routing = None
        Framed-MTU = 1500
        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

Prime Access Registrar 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 Prime Access Registrar supports a subset of EAP to support LEAP. This is not a general implementation of EAP for Prime Access Registrar.

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

You can use **aregcmd** 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.

To enable EAP-LEAP:

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

cd /Radius/Services

add eap-leap-service

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

cd eap-leap-service

set type 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

Cisco Prime Access Registrar 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 **eap-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.

You can use **aregcmd** to create and configure a service of type **eap-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.

To enable EAP-MD5:

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

cd /Radius/Services

add eap-md5-service

Step 2 Set the service type to **eap-md5**.

cd eap-md5-service

set type eap-md5

```
[ //localhost/Radius/Services/eap-md5-service ]
    Name = newone
    Description =
    Type =
    IncomingScript~ =
```

```
OutgoingScript~ =
AuthenticationTimeout = 120
UserService =
```

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

EAP-Negotiate

EAP-Negotiate is a special service used to select at runtime 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 tires 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 Prime Access Registrar. 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.

This section contains the following topics:

Configuring EAP-Negotiate

You may first use **aregcmd** to create and configure the EAP services that will be used for authentication, then create and configure a service of type eap-negotiate.

To enable EAP-Negotiate:

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

cd /Radius/Services

add eap-negotiate-service

Step 2 Set the service type to **eap-negotiate**.

cd eap-negotiate-service

set type eap-negotiate

```
[ //localhost/Radius/Services/negotiate ]
    Name = negotiate
    Description =
    Type = eap-negotiate
    IncomingScript~ =
```

```
OutgoingScript~ =
AuthenticationTimeout = 120
ServiceList =
```

Step 3 Set the ServiceList property to a list of preconfigured 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).

This section contains the following topics:

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

```
[ //localhost/Radius/Services/eap-mschapv2 ]
    Name = eap-mschapv2
    Description =
    Type = eap-mschapv2
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    UserService =
    SystemID =
```

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**:

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

```
p006
```

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

```
Packet: code = Access-Accept, id = 4, length = 104, attributes =
```

```
Service-Type = Framed
Framed-Protocol = PPP
Framed-IP-Address = 192.168.0.0
Framed-IP-Netmask = 255.255.255.0
Framed-Routing = None
Framed-MTU = 1500
Framed-Compression = VJ TCP/IP header compression
Framed-IPX-Network = 1
EAP-Message = 03:01:00:04
Ascend-Idle-Limit = 1800
Message-Authenticator = 27:90:7e:20:78:34:43:2e:9d:cd:a8:75:82:53:03:65
```

EAP-SIM

Cisco Prime Access Registrar 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 Prime Access Registrar RADIUS server to perform EAP-SIM authentication of mobile clients. Prime Access Registrar must obtain authentication information from the HLR. Prime Access Registrar contacts the MAP gateway that performs the MAP protocol over SS7 to the HLR, see SIGTRAN-M3UA for more information.

In support of EAP-SIM, the Wx Interface feature will be supported. For more information on Wx Interface Support, see the Wx Interface Support for SubscriberDB Lookup, page 17-48.

Configuring EAP-SIM

You can use **aregcmd** to create and configure a service of type *eap-sim*.

Table 9-6 lists and describes the EAP-SIM specific properties.

Table 9-6 EAP-SIM Service Properties

Property	Description
AlwaysRequestIdentity	When True, enables the server to obtain the subscriber's identity via EAP/AKA 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. **Note** 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.

Table 9-6 EAP-SIM Service Properties (continued)

Property	Description
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 it's 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 reauthentication option is enabled. The default value is False.
MaximumReauthentica-tions	Specifies the maximum number of times a reauthentication identity might be reused before it must be renewed. The default value is 16.
ReauthenticationTimeout	Specifies the time in seconds that reauthentication identities are cached by the server. Subscribers that attempt to reauthenticate using identities that are older than this value will be forced to use full authentication instead. The default value is 3600 (one hour).
ReauthenticationRealm	Optional. If you configure the realm, this value is appended to the FastReauthenticationUserId.
AuthenticationTimeout	Time in seconds to wait for authentication to complete. The default is 2 minutes; range is 10 seconds to 10 minutes.
QuintetGenerationScript~	Optional. If the script is set, the custom scripting point can be used to read the quintets from a flat file or generate quintets instead of fetching the quintets from HLR.If the script is not set, the Prime Access Registrar sends the request to HLR configured in remote server to fetch the quintets.
UseProtectedResults	Enables or disables the use of protected results messages. Results messages indicate the state of the authentication but are cryptographically protected.
TripletCacheTimeout	Required; timeout value of triplet cache.
SubscriberDBLookup	Required. Must be set to either DIAMETER or SIGTRAN-M3UA. When set to DIAMETER, the HSS lookup happens using the Diameter Wx Interface. You need to configure the DestinationRealm to send the Diameter packets to the RemoteServer. When set to SIGTRAN-M3UA, the HLR/HSS lookup happens using the SIGTRAN protocol. You need to configure the SIGTRAN remote server.

Table 9-6 EAP-SIM Service Properties (continued)

Property	Description
FetchAuthorizationInfo	Required. When set True, it fetches MSISDN from HLR. This field is dispayed when you set Subscriber_DBLookup as SIGT-RAN-M3UA.
IncomingScript~	Optional script Prime Access Registrar server runs when it receives a request from a client for an EAP-AKA/EAP-SIM service.
OutgoingScript~	Optional script Prime Access Registrar server runs before it sends a response to a client using an EAP-AKA/EAP-SIM service.
OutageScript~	Optional. If set to the name of a script, Prime Access Registrar runs the script when an outage occurs. This property allows you to create a script that notifies you when the server detects a failure.
RemoteServers	Remote server which can provide the service.

To enable EAP-SIM authentication using argcmd:

Step 1 Launch **aregcmd** and create an EAP-SIM 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

```
[ //localhost/Radius/Services/EAP-SIM ]
    Name = EAP-SIM
    Description =
    Type = eap-sim
    NumberOfTriplets = 2
    UseSimDemoTriplets = False
    AlwaysRequestIdentity = False
    EnableIdentityPrivacy = False
    PseudonymSecret = <encrypted>
    PseudonymRenewtime = "24 Hours"
    PseudonymLifetime = Forever
    Generate3GPPCompliantPseudonym = False
    EnableReauthentication = False
    MaximumReauthentications = 16
    ReauthenticationTimeout = 3600
    ReauthenticationRealm =
    TripletCacheTimeout = 120
    AuthenticationTimeout = 120
    UseProtectedResults = False
    SendReAuthIDInAccept = False
    SubscriberDBLookup = SIGTRAN-M3UA
    FetchAuthorizationInfo = FALSE
    MultipleServersPolicy = Failover
    IncomingScript~ =
    OutgoingScript~ =
```

```
    OutageScript~ =
    RemoteServers/

[ //localhost/Radius/Services/eap-sim-wx ]
Name = eap-sim-wx
Description =
Type = eap-sim
NumberOfTriplets = 2
UseSimDemoTriplets = False
AlwaysRequestIdentity = False
EnableIdentityPrivacy = False
PseudonymSecret = <encrypted>
PseudonymRenewtime = "24 Hours"
PseudonymLifetime = Forever
Generate3GPPCompliantPseudonym = False
EnableReauthentication = False
MaximumReauthentications = 16
ReauthenticationTimeout = 3600
ReauthenticationRealm =
TripletCacheTimeout = 120
AuthenticationTimeout = 120
UseProtectedResults = False
SendReAuthIDInAccept = False
SubscriberDBLookup = DIameter
DestinationRealm = hss.com
PreRequestTranslationScript~ =
PostRequestTranslationScript~ =
PreResponseTranslationScript~ =
PostResponseTranslationScript~
```

Note The EAP-SIM property OutagePolicy present in earlier versions of Prime Access Registrar is no longer part of the EAP-SIM configuration.

To enable EAP-SIM authentication using **radclient**:

Step 1 Create an EAP-SIM service.

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

Step 3 Execute the below command in radclient to set session keys in the server.

simple_eap_sim_test 987456321123654 secret

Note The IMSI number that is stored in HLR is used for EAP-SIM authentication.

Step 4 Enter the server name in which the session key is created to view the *eap-sim* service details.

p006

```
Packet: code = Access-Accept, id = 3, length = 207, attributes =
User-Name = 987456321123654
```

```
MS-MPPE-Send-Key =
9c:56:e5:36:9f:fe:84:a2:26:16:80:0a:13:74:fb:b7:87:30:00:5c:45:99:ea:78:af:7d:ae:37:0e
:b1:3a:2e:2b:b1:c8:4f:20:39:33:04:eb:dc:ba:27:e7:6f:56:08:21:56
EAP-Message = 03:02:00:04
Cisco-AVPair = auth-algo-type=eap-sim
MS-MPPE-Recv-Key =
8b:27:42:c5:47:79:ce:6a:41:ae:34:1f:15:2f:cf:b8:ee:18:e7:b5:1c:64:41:26:f7:4b:bc:53:bd
:54:57:70:a3:3b:df:78:9e:34:33:47:b3:a2:ff:4e:f1:fe:6f:8f:ee:aa
Message-Authenticator = 45:02:01:97:55:3d:bc:80:34:76:a4:5a:6b:29:ac:bc
```

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.

This section contains the following topics:

Configuring EAP-TLS

You can use **aregcmd** to create and configure a service of type *eap-tls*. Table 9-7 describes the EAP-TLS configuration properties:

Table 9-7 EAP-TLS Service Properties

Property	Description
IncomingScript	Optional script Prime Access Registrar server runs when it receives a request from a client for EAP-TLS service
OutgoingScript	Optional script Prime Access Registrar 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.

Table 9-7 EAP-TLS Service Properties (continued)

Property	Description
ServerKeyFile	The full pathname of the file containing the server's RSA or ECC 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. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-key.pem** is assumed to be in PEM format. The file extension ***.pem*** is not significant. **set ServerKeyFile PEM:/cisco-ar/pki/server-key.pem**
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	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. 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. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to **ca-cert.pem**. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extension as in 1b96dd93.0 and 1b96dd93.1.

CRLDistributionURL	Optional. The URL that Prime Access Registrar should use to retrieve the CRL.You can specify a URL that uses HTTP or LDAP.
	The following is an example for an HTTP URL:
	`<http://crl.verisign.com/pca1.1.1.crl>.`
	The following is an example for an LDAP URL:
	`ldap://209.165.200.225:388/CN=development-CA,CN=acs-westcoast 2,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=cisco,DC=com`

Table 9-7 EAP-TLS Service Properties (continued)

Property	Description
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length **(in bytes?)** of the certificate chain used for client verification.
UseECCCertificates	Determines the applicability of the authentication mechanism in SmartGrid Solutions, see the Smart Grid Solution Management, page 17-50 for more information. When UseECCCertificates is set to True, it can use the ECC, RSA, or combination of both certificate for certificate based verification. When UseECCCertificates is set to False, it can only use the RSA certificate for certificate based verification. The default location to fetch the certificate file is **/cisco-ar/pki**.
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: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.

To enable EAP-TLS authentication:

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-service
```

```
Description =
Type = eap-tls
IncomingScript~ =
OutgoingScript~ =
MaximumMessageSize = 1024
PrivateKeyPassword = <encrypted>
ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
ServerKeyFile = /opt/CSCOar/pki/server-key.pem
CACertificateFile = /opt/CSCOar/pki/root-cert.pem
CACertificatePath = /opt/CSCOar/pki
CRLDistributionURL =
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = true
UseECCCertificates = true
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120
```

Note Prime Access Registrar 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 RSA or ECC Certificate using 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. You can either select RSA or ECC client certificates using this argument.

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 or ECC 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

Prime Access Registrar 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.

This section contains the following topics:

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

You can use **aregcmd** to create and configure a service of type *eap-ttls*. Table 9-8 describes the EAP-TTLS configuration properties:

Table 9-8 EAP-TTLS Service Properties

Property	Description
IncomingScript	Optional script Prime Access Registrar server runs when it receives a request from a client for EAP-TTLS service.
OutgoingScript	Optional script Prime Access Registrar 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.
ServerKeyFile	The full pathname of the file containing the server's RSA or ECC 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. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-key.pem** is assumed to be in PEM format. The file extension ***.pem*** is not significant. **set ServerKeyFile PEM:/cisco-ar/pki/server-key.pem**
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. **Note** DER encoding is not allowed.

Table 9-8 EAP-TTLS Service Properties (continued)

Property	Description
CACertificatePath	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. 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. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to **ca-cert.pem**. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extension as in 1b96dd93.0 and 1b96dd93.1. See rehash-ca-certs Utility, page 9-40 for information about how to create the required certificate file hash links.
CRLDistributionURL	Optional. The URL that Prime Access Registrar should use to retrieve the CRL.You can specify a URL that uses HTTP or LDAP. The following is an example for an HTTP URL: `<http://crl.verisign.com/pca1.1.1.crl>`. 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`
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.

Table 9-8 EAP-TTLS Service Properties (continued)

Property	Description
UseECCCertificates	Determines the applicability of the authentication mechanism in SmartGrid Solutions, see the Smart Grid Solution Management, page 17-50 for more information. When UseECCCertificates is set to True, it can use the ECC, RSA, or combination of both certificate for certificate based verification. When UseECCCertificates is set to False, it can only use the RSA certificate for certificate based verification. The default location to fetch the certificate file is **/cisco-ar/pki**.
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 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: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out. The default is 120.
AuthenticationService	Mandatory; specifies the authentication service to use to authenticate users. See Configuring an EAP-TTLS Authentication Service, page 9-35 for more information. **Note** The authentication service must exist before you can save the EAP-TTLS service configuration.

To enable EAP-TTLS authentication:

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-service
   Description =
   Type = eap-ttls
   IncomingScript~ =
   OutgoingScript~ =
   MaximumMessageSize = 1024
   PrivateKeyPassword = <encrypted>
   ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
   ServerKeyFile = /opt/CSCOar/pki/server-key.pem
   CACertificateFile = /opt/CSCOar/pki/root-cert.pem
   CACertificatePath = /opt/CSCOar/pki
   CRLDistributionURL =
   ClientVerificationMode = Optional
   VerificationDepth = 4
   EnableSessionCache = true
   UseECCCertificates = true
   SessionTimeout = "5 Minutes"
   AuthenticationTimeout = 120
```

Note Prime Access Registrar 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 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.

To configures an EAP-TTLS service to authenticate using eap-negotiate:

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.

To authenticate with the eap-negotiate 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 4-12, for information about configuring RADIUS services.

To configure a remote RADIUS server (aaa-remote) with its IP address and a shared secret:

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 4-12, 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 ***identity password { method }***

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

To authenticate a user using EAP-TTLS with PAP:

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.

To authenticate a user using EAP-TTLS with EAP-MSChapV2 as the Phase 2 method:

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

```
start rehashing ../pki
client-key.pem does not contain a PEM certificate
finished rehashing
```

The **rehash-ca-certs** utility warns about PEM files that do not contain certificates. On Cisco Prime Access Registrar, intermediate/chained certificates cannot be imported.

To run Prime Access Registrar on Solaris with PEAP authentication:

Step 1 Add both root and intermediate CA in the direcotry **/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. It contains the following topics:

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

Cisco Prime Access Registrar 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.

This section contains the following topics:

PEAP Version 0

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

Configuring PEAP Version 0

You can use **aregcmd** to create and configure a service of type *peap-v0*. Table 9-9 describes the PEAP service properties for PEAP Version 0.

Table 9-9 PEAP Version 0 Service Properties

Property	Description
IncomingScript	Optional script Prime Access Registrar server runs when it receives a request from a client for PEAP-v0 service.
OutgoingScript	Optional script Prime Access Registrar 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 9-9 PEAP Version 0 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. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-cert.pem** is assumed to be in PEM format; note that the file extension ***.pem*** is not significant. **set ServerCertificateFile PEM:/cisco-ar/pki/server-cert.pem**
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	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. 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. For example, if a certificate file name **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to the **ca-cert.pem** file. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extension as in 1b96dd93.0 and 1b96dd93.1.

CRLDistributionURL	Optional. The URL that Prime Access Registrar should use to retrieve the CRL.You can specify a URL that uses HTTP or LDAP.
	The following is an example for an HTTP URL:
	`<http://crl.verisign.com/pca1.1.1.crl>.`
	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`

Table 9-9 PEAP Version 0 Service Properties (continued)

Property	Description
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
UseECCCertificates	Determines the applicability of the authentication mechanism in SmartGrid Solutions, see the Smart Grid Solution Management, page 17-50 for more information. When UseECCCertificates is set to True, it can use the ECC, RSA, or combination of both certificate for certificate based verification. When UseECCCertificates is set to False, it can only use the RSA certificate for certificate based verification. The default location to fetch the certificate file is **/cisco-ar/pki**.
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: **Set SessionTimeout "1 Hour 45 Minutes"**
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.

Table 9-9 PEAP Version 0 Service Properties (continued)

Property	Description
MasterURL	When using WPS, specifies the URL of the provisioning server which is modified with the appropriate fragment and sent to the client.
WPSGuestUserProfile	When using WPS, specifies a profile to be used as a guest user profile; must be a valid profile under **/Radius/Profiles.** 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).

To enable PEAP Version 0:

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

```
//localhost/Radius/Services/eap-peap-v0-service ]
   Name = eap-peap-v0-service
   Description =
   Type = eap-peap-v0
   IncomingScript~ =
   OutgoingScript~ =
   MaximumMessageSize = 1024
   PrivateKeyPassword = <encrypted>
   ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
   ServerKeyFile = /opt/CSCOar/pki/server-key.pem
   CACertificateFile = /opt/CSCOar/pki/root-cert.pem
   CACertificatePath = /opt/CSCOar/pki
   CRLDistributionURL =
   ClientVerificationMode = Optional
   VerificationDepth = 4
   EnableSessionCache = true
   UseECCCertificates = true
   SessionTimeout = "5 Minutes"
   AuthenticationTimeout = 120
    EnableWPS = FALSE
```

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

Note Prime Access Registrar 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:

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 1124567891 and key 0112456789ABCDEF:

tunnel eap-sim

simple_eap_peapv0_test 1124567891 0112456789ABCDEF

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 or ECC 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

You can use **aregcmd** to create and configure a service of type *peap-v1*. Table 9-10 describes the PEAP service properties for both PEAP Version 1.

Table 9-10 PEAP Version 1 Service Properties

Property	Description
IncomingScript	Optional script Prime Access Registrar server runs when it receives a request from a client for PEAP-v1 service.
OutgoingScript	Optional script Prime Access Registrar 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.
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	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. 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. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to the **ca-cert.pem** file. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extension as in 1b96dd93.0 and 1b96dd93.1.

Table 9-10 PEAP Version 1 Service Properties (continued)

Property	Description
CRLDistributionURL	Optional. The URL that Prime Access Registrar should use to retrieve the CRL.You can specify a URL that uses HTTP or LDAP. The following is an example for an HTTP URL: `<http://crl.verisign.com/pca1.1.1.crl>`. The following is an example for an LDAP URL: `ldap://209.165.200.225:388/CN=development-CA,CN=acs-westcoast 2,CN=CDP,CN=Public Key Services,CN=Services,CN=Configuration,DC=cisco,DC=com`
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
UseECCCertificates	Determines the applicability of the authentication mechanism in SmartGrid Solutions, see the Smart Grid Solution Management, page 17-50 for more information. When UseECCCertificates is set to True, it can use the ECC, RSA, or combination of both certificate for certificate based verification. When UseECCCertificates is set to False, it can only use the RSA certificate for certificate based verification. The default location to fetch the certificate file is **/cisco-ar/pki**.
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: **Set SessionTimeout "1 Hour 45 Minutes"**
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.

To enable PEAP Version 1:

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

```
//localhost/Radius/Services/eap-peap-v1-service ]
   Name = eap-peap-v1-service
   Description =
   Type = eap-peap-v1
   IncomingScript~ =
   OutgoingScript~ =
   MaximumMessageSize = 1024
   PrivateKeyPassword = <encrypted>
   ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
   ServerKeyFile = /opt/CSCOar/pki/server-key.pem
   CACertificateFile = /opt/CSCOar/pki/root-cert.pem
   CACertificatePath = /opt/CSCOar/pki
   CRLDistributionURL =
   ClientVerificationMode = Optional
   VerificationDepth = 4
   EnableSessionCache = true
   UseECCCertificates = true
   SessionTimeout = "5 Minutes"
   AuthenticationTimeout = 120
```

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

Testing PEAP Version 1 with radclient

To test the PEAP Version 1:

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 or ECC 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.

How to Configure Oracle, Mysql Accounting with the Bufferring Option Enabled

Prime Access Registrar provides support for MySQL to query user records from Oracle database using sql interface and enables you to write accounting records into Oracle database. You can use insert, update, and delete queries to

- add new details into database.
- modify the existing details in the database.
- remove the outdated details from the database.

To Select the SQL Statement in Run Time Accounting

Prime Access Registrar provides support to query user account details from SQL database and enables you to add, delete, and update accounting details into SQL when using Oracle accounting.

You can execute the following SQL statements to perform various actions:

- Query
- Insert
- Update
- Delete
- Configuring Oracle, Mysql Accouting

Query

You can query the accounting details from Oracle by referring this service in **/Radius/DefaultAuthenticationService** and in **/Radius/DefaultAuthorization**.

The following example is an SQL statement used for Authentication and Authorization of the subscribed users. You can use the SQL and MarkerList properties statement to query the selected attributes from Oracle.

```
sql1/
  Name = sql1
  Description =
  Type = query
  SQL = "select password , username from arusers where username = ?"
  ExecutionSequenceNumber = 1
  MarkerList = UserName/SQL_CHAR
```

Insert

You can insert user details into SQL database by Oracle accounting. This service is used by referring the **/Radius/DefaultAccountingService** or **Accounting-Service** environment variable.

For instance, you can use the following SQL and MarkerList properties statement to insert the selected attributes:

```
sql1/
        Name = sql1
        Description =
        Type = insert
        SQL = "insert into sql_test (username,nas) values (?,?)"
        ExecutionSequenceNumber = 1
        MarkerList = "UserName/SQL_CHAR NAS-Identifier/SQL_CHAR"
```

Update

You can easily modify the details in an SQL table with the UPDATE statement.

For example, you can use the following SQL and MarkerList properties statement to update the selected attributes:

```
sql2/
        Name = sql2
        Description =
        Type = update
        SQL = "update sql_test set packet='stop' where username=?"
        ExecutionSequenceNumber = 2
        MarkerList = UserName/SQL_CHAR
```

Delete

You can remove the unnecessary records from SQL database using DELETE statement.

For example, you can use the following SQL and MarkerList properties statement to delete the selected attributes:

```
sql/
  Name = sql
  Description =
  Type =delete
  SQL = "delete from arusers_acct where username=?"
 ExecutionSequenceNumber = 1
 MarkerList = UserName/SQL_CHAR
```

Configuring Oracle, Mysql Accouting

The following script describes you how to configure Oracle, Mysql accounting with the buffering option enabled:

```
[ //localhost/Radius/Services/oracle-acc ]
    Name = oracle-acc
    Description =
    Type = oci-accounting
    IncomingScript~ = sql
    OutgoingScript~ =
    OutagePolicy~ = RejectAll
    OutageScript~ =
    MultipleServersPolicy = Failover
    RemoteServers/

[ //localhost/Radius/Services/oracle-acc/RemoteServers ]
    1. oracle-acc

[ //localhost/Radius/RemoteServers/oracle-acc ]
    Name = oracle-acc
    Description =
    Protocol = oci-accounting
    ReactivateTimerInterval = 300000
    Timeout = 15
    DataSourceConnections = 8
    ODBCDataSource = oracle
    SNMPTrapIP =
    SNMPTrapPort = 1521
    KeepAliveTimerInterval = 0
    BufferAccountingPackets = TRUE
    MaximumBufferFileSize = "10 Megabytes"
    NumberOfRetriesForBufferedPacket = 3
    BackingStoreEnvironmentVariables =
    UseLocalTimeZone = FALSE
    AttributeList =
    Delimiter =
    SQLDefinition/

[ //localhost/Radius/Advanced/ODBCDataSources/oracle ]
    Name = oracle
    Description =
    Type = oracle_oci
    UserID = scott
    Password = <encrypted>
    DataBase = ORCL

[ //localhost/Radius/Scripts/sql ]
    Name = sql
    Description =
    Language = tcl
    Filename = sql.tcl
    EntryPoint = sqltest
    InitEntryPoint =
    InitEntryPointArgs =
```

Script

The script statements are executed based on the IP address that you specified in the query. Here is a sample script to select the SQL statements.

```
proc sqltest {request response environ} {
```

```
set nas [ $request get NAS-Identifier ]
if { [ string compare $nas 1.1.1.1 ] == 0 } {
      $environ put SQL-Sequence "sql1"
      $environ put BackingStore-Env-Vars "SQL-Sequence"
}
if { [ string compare $nas 1.1.1.2 ] == 0 } {
      $environ put SQL-Sequence "sql2"
      $environ put BackingStore-Env-Vars "SQL-Sequence"
}
if { [ string compare $nas 1.1.1.3 ] == 0 } {
      $environ put SQL-Sequence "sql3"
     $environ put BackingStore-Env-Vars "SQL-Sequence"
}
if { [ string compare $nas 1.1.1.4 ] == 0 } {
      $environ put SQL-Sequence "sql4"
      $environ put BackingStore-Env-Vars "SQL-Sequence"
}
}
```

How Suffix and Prefix Rules Work with Prime Access Registrar

Prime Access Registrar includes several scripts that you can use with the rules. The following are the most commonly used rules:

- Prefix Rule, See **ExecPrefixRule, page 18-17** for more information
- Suffix Rule, See **ExecSuffixRule, page 18-18** for more information

Configuring Prefix and Suffix Policies

To configure prefix and suffix policies in Prime Access Registrar in order to provide authentication and authorization services for the subscribed users:

Step 1 Activate the Policy Engine by configuring **SelectPolicy**. This script explains you how to set a suffix and prefix policy in the grouping list.

```
--> cd selectPolicy/

[ //localhost/Radius/Policies/SelectPolicy ]
    Name = SelectPolicy
    Description =
    Grouping = suffixrule&prefixrule
```

Step 2 Run the configuration rules for Prefix and Suffix.

Step 3 Set Script = ExecSuffixRule in the prefix rule configuration.

```
[ //localhost/Radius/Rules ]
    Entries 1 to 2 from 2 total entries
    Current filter: <all>

    prefixrule/
        Name = prefixrule
        Description =
        Type = radius
        Script~ = ExecPrefixRule
        Attributes/
            Authentication-Service = local-users
            Authorization-Service = local-users
```

```
            Delimiters = @#%$/
            Prefix = cisco
            StripPrefix = no
```

Step 4 Specify Script = ExecRealmRule in the suffix configuration to scan.

```
suffixrule/
    Name = suffixrule
    Description =
    Type = radius
    Script~ = ExecRealmRule
    Attributes/
        Realm = @cisco.com
```

CRL Support for Cisco Prime Access Registrar

Prime Access Registrar checks for various certificates for validation purposes in its authentication services. The client sends a certificate along with the access-challenge to Prime Access Registrar. Prime Access Registrar verifies the validity of the certificate and approves the request if the certificate is valid. For certificate validation, Prime Access Registrar 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 Prime Access Registrar rejects any request that comes with these certificates. The CRLs are available in a public repository in Prime Access Registrar.

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 they that is associated with the certificate.

Prime Access Registrar 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 Prime Access Registrar. When you configure this property, Prime Access Registrar fetches the CRL from the specified URL, at the startup. A background thread in Prime Access Registrar keeps track of these CRLs. When any of the CRLs expires, Prime Access Registrar fetches the latest version of CRL using the specified URL. Each CRL contain the information related to its expiry.

Prime Access Registrar 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 Prime Access Registrar 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, Prime Access Registrar considers the certificate invalid.

This section contains the following topics:

Note Prime Access Registrar uses the **CRLDistributionURL** property in the following services:
eap-tls
eap-ttls
eap-fast
peap-v0
peap-v1

Configuring Certificate Validation Using CRL

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

```
//localhost/Radius/Services/eap-ttls-service ]
   Name = eap-ttls-service
   Description =
   Type = eap-ttls
   IncomingScript~ =
   OutgoingScript~ =
   MaximumMessageSize = 1024
   PrivateKeyPassword = <encrypted>
   ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
   ServerKeyFile = /opt/CSCOar/pki/server-key.pem
   CACertificateFile = /opt/CSCOar/pki/root-cert.pem
   CACertificatePath = /opt/CSCOar/pki
   CRLDistributionURL =
   ClientVerificationMode = Optional
   VerificationDepth = 4
   EnableSessionCache = true
   UseECCCertificates = true
   SessionTimeout = "5 Minutes"
   AuthenticationTimeout = 120
```

Table 9-8 describes the properties in this sample configuration.

Using Intermediate Certificates in Prime Access Registrar

The rehash-ca-certs utility can be used to import intermediate certificates in Prime Access Registrar. See rehash-ca-certs Utility, page 9-40 for information about how to create the required certificate file hash links.

To import intermediate certificates in Prime Access Registrar:

Step 1 Copy the Root CA, Intermediate CA of the client to a directory.

Step 2 Run **/opt/CSCOar/bin/rehash-ca-certs –v <path of the client certificate store>**

The utility creates the required hash links to maintain the chain between the Root CA certificate and Intermediate CA certificates.

Step 3 Set the CACertificateFile property in EAP service to the path where Root CA Certificate of the client is stored.

Step 4 Restart the Prime Access Registrar server.

The following shows an example to import intermediate certificates in Prime Access Registrar:

Step 1 Copy the Client Root CA and Intermediate CA Certificate in **/cisco-ar/certs/wimax/ directory**.

cp /tmp/wimax_device_root.pem /cisco-ar/certs/wimax/

cp /tmp/wimax_device_root_ca1.pem /cisco-ar/certs/wimax/

/opt/CSCOar/bin/rehash-ca-certs –v /cisco-ar/certs/wimax/

Step 2 Enter in to aregcmd.

/opt/CSCOar/bin/aregcmd –s

a. Configure the eap service which uses these client certificates.

cd Radius/Services/eap-ttls

```
//localhost/Radius/Services/eap-ttls-service ]
       Name = eap-ttls-service
       Description =
       Type = eap-ttls
       IncomingScript~ =
       OutgoingScript~ =
       MaximumMessageSize = 1024
       PrivateKeyPassword = <encrypted>
       ServerCertificateFile = /opt/CSCOar/pki/server-cert.pem
       ServerKeyFile = /opt/CSCOar/pki/server-key.pem
       CACertificateFile = /opt/CSCOar/pki/root-cert.pem
       CACertificatePath = /opt/CSCOar/pki
       CRLDistributionURL =
       ClientVerificationMode = Optional
       VerificationDepth = 4
       EnableSessionCache = true
       UseECCCertificates = true
       SessionTimeout = "5 Minutes"
       AuthenticationTimeout = 120
```

set CACertificateFile PEM:/opt/CSCOar/pki/wimax_device_root.pem

```
Set CACertificateFile PEM:/opt/CSCOar/pki/wimax_device_root.pem
```

Step 3 Save the configuration.

save

Step 4 Restart the arserver.

/opt/CSCOar/bin/arserver restart