LinkAuthentication:1

Service Template Version 1.01

For UPnP^{TM} Version 1.0
Status: Standardized DCP
Date: October 17, 2003

Authors	Company
Christopher Lord	Intel Corporation
William Lupton	GlobespanVirata Inc
Ajay Garg	Intel Corporation

Contents

List of Tables

1. Overview and Scope

This device template is compliant with the UPnP™ Device Architecture, Version 1.0.

This service-type enables a UPnP™ control point to configure and control the parameters pertaining to authentication by an authentication server. The service specifies variables and actions that are used by control points to add, update and delete records used for authentication. This would typically be used for maintaining per-client authentication parameters on a device. This service would support a user/client list with the credentials (password, public key) and the specific access rights on a per-user basis. The service is mainly designed for authentication on a wireless access point (AP) that implements link layer security such as 802.1x. It may be used for other purposes - e.g., to securely store client credentials such as certificates and asymmetric keys for network-layer security protocols.

The working committee has however looked at this service only from the perspective of 802.1x usage and therefore this document makes several references to the 802.1x protocol. This service may be co-located with the access point device that requires the authentication service or located on a different device on the network such as an Internet Gateway Device (IGD). The service was defined to associate WLAN clients and their credentials to bootstrap a secure WLAN in a UPnP™ technology compliant ***WLANAccessPointDevice***[*].

This service is defined as a standalone service and will remain at the component level. Any product that implements a standard device specification will have the option to implement this standard service specification. The product will be tested at certification testing time for this service in addition to being tested to the product's original device type (e.g., ***WLANAccessPointDevice, InternetGatewayDevice***).

[*] Refer to companion documents defined by the UPnP™ Internet Gateway working committee for more details on specific devices and services referenced in this document.

2. Service Modeling Definitions

2.1. ServiceType

The following service type identifies a service that is compliant with this template:

urn:schemas-upnp-org:service:*LinkAuthentication:1*

2.2. State Variables

Table 1: State Variables

Variable Name	Req. or Opt.[1]	Data Type	Allowed Value [2]	Default Value [2]	Eng. Units
`NumberOfEntries`	R	ui2	>=0	0	N/A
`Identifier`	R	String	<= 64 char	Empty string	N/A
`Secret`	R	String	Encoded in BASE64, <= 1024 char	Empty string	N/A
`SecretType`	R	String	See Table 1.1, <= 32 char	Not specified	N/A
`AuthType`	R	String	See Table 1.2, <= 32 char	Not specified	N/A
`AuthState`	R	String	See Table 1.3, <= 32 char	“Unconfigured”	N/A
`CredentialState`	R	String	See Table 1.4, <= 32 char	“Unconfigured”	N/A
`Description`	R	String	<= 256 char	Empty string	N/A
`MACAddress`	R	String	MAC address, “xx:xx:xx:xx:xx:xx”, case-independent, 17 char	Empty string	N/A
`CredentialDuration`	R	ui4	>= 0	0	Seconds
`LinkedIdentifier`	R	String	<= 64 char	Empty string	N/A
`LastChange`	R	String	<= 1024 char	Empty string	N/A
`LastError`	R	String	<= 1024 char	Empty string	N/A
Non-standard state variables implemented by an UPnP™	*X*	*TBD*	*TBD*	*TBD*	*TBD*

Variable Name		

[3] R = Required, O = Optional.

2.2.5. `AuthType`

This variable specifies the type of authentication. Possible string values are specified in table 1.2

Table 1.2: allowedValueList for `AuthType`

Value	Req. or Opt. [1]	Description
SharedSecret	R	This value indicates the 802.1x client is using an authentication method that requires addition of a credential such as shared secret that is missing, i.e., *`TextPassword`*. When AuthState is *`Pending`* the user would be expected to provide the shared secret to the AP via the control point using the `UpdateEntry` action. This results in a change in the `Secret` field value.
ValidateCredentials	R	This value indicates the 802.1x client is using an authentication method that requires the verification of existing credentials, i.e. as in an X509Certificate. When `AuthState` is *`Pending`* this requires the user or control point to validate the credential in the `Secret` field.
Vendor-defined	R	R
Vendor-defined	O	O

[1] R = Required, O = Optional.

2.2.6. `AuthState`

This variable specifies the current state of the authentication process. Possible string values are specified in table 1.3. Refer to the state diagram and theory of operation for details.

Table 1.3: allowedValueList for `AuthState`

Value	Req. or Opt. [1]	Description
Unconfigured	R	A WLAN client corresponding to this `Identifier` has yet to authenticate.
Failed	R	A WLAN client corresponding to this `Identifier` failed to successfully authenticate the contents of the `secret` field - for example, with 802.1x authentication. This state informs control points the secret that was entered or validated has failed.
Succeeded	R	A WLAN client corresponding to this `Identifier` successfully authenticated using the contents of the `secret` field, for example, with 802.1x authentication. This state informs control points that the secret that was entered or validated has succeeded.
Vendor-defined	R	R
Vendor-defined	O	O

[1] R = Required, O = Optional.

2.2.7. `CredentialState`

This variable specifies the current state of the credential approval/validation process. Possible string values are specified in table 1.4. Refer to the state diagram and theory of operation for details.

Table 1.4: allowedValueList for `CredentialState`

Value	Req. or Opt. [1]	Description
Unconfigured	R	This value indicates that other variables in the particular record are uninitialized or in an invalid state. Examples of such variables include `Secret` and `AuthType`. This record will not be used for authentication; it may be used for other purposes.
Pending (alert CP for action)	R	This value indicates the record referred to by the `Identifier` does not have a validated `Secret` field. Control points are expected to prompt the end-user to enter the SharedSecret or validate a credential. Upon end-user acceptance the control point changes `AuthState` to either `Accepted` or `Denied`
Accepted	R	This value indicates the database record referred to by the `Identifier` field has had its `Secret` field entered/validated by the end-user.
Denied	R	This value indicates that when a client attempts to authenticate it is not allowed to proceed with link layer authentication. This state is intended to restrict devices from the network and restrict events from those devices.
Vendor-defined	*R*	*R*
Vendor-defined	*O*	*O*

[1] R = Required, O = Optional.

The component performing authentication and authorization for network access is expected to refer to the client database for credential information. If a given `Identifier` is not found in the database, a new record is created, populated, and set to Pending. The user, via a control point, can update the record to `Accepted`, or `Denied`. The control point may query the user for an amount of time to grant the client temporary network access. The amount of time is specified in seconds and is stored in the `CredentialDuration` variable.

2.2.8. `Description`

This variable stores a string used exclusively by control points to aid in identifying authentication records. For example, a control point may prompt the user to supply a friendly name for the client device. It is stored in the authentication records and may not be used by the AP device. The Description may contain the characters < > & which will “break” the surrounding XML, therefore the Description string must be ‘escaped’. Refer to DeviceSecurity section “XML Strings as UPnP™ Parameters”.

2.2.9. `MACAddress`

This variable specifies the MAC address of the client. It is not intended to be used to authenticate the client, as MAC Addresses are mutable. This field can be used by control points to display the device's MAC address during the initial authentication process when the client device is first added to the network and authentication database. This field is dynamically updated by the AP when a client device is authenticating. This field is also updated to reflect the last MAC address that successfully or unsuccessfully authenticated. See the theory of operation section for more details.

2.2.10. `CredentialDuration`

This variable determines the time (in number of seconds) a client record with a `CredentialState` of *`Accepted`* is allowed to authenticate. A value of 0 means the client record is permanent, that is, there is no expiration period. Non-zero values specify temporary access. When a non-zero `CredentialDuration` transitions to zero (the number of seconds has expired) the record must be deleted and LastChange event triggered. Note that permanent client records with `CredentialDuration` of zero persist across device resets or reboots. It is up to vendors to implement persistence as appropriate for their device, for example store in non-volatile storage such as flash or disk. Non-zero `CredentialDuration` values do not persist across device resets or reboots, that is, the temporary client is expected to be re-authenticated. The remaining number of seconds can be retrieved via the `GetGenericEntry` and `GetSpecificEntry` actions. Automatic second decrements to the `CredentialDuration` do NOT generate a `LastChange` event.

2.2.11. `LinkedIdentifier`

Some EAP authentication methods allow two EAP negotiations. For example, PEAP optionally negotiates two phases, a Part 1 and a Part 2 phase to complete authentication. Each phase may negotiate using unique Identifiers, AuthTypes, and credentials. To allow the EAP server to indicate related records to the control point this field contains the Identifier for the initial EAP negation. For example, if PEAP Part1 used an Identifier of User1, AuthType of `ValidateCredentials` and SecretType of PubKeyHash160 and PEAP Part 2 used an Identifier of User2, AuthType of SharedSecret, then two records would exist with the LinkedIdentifier field of the User2 record containing User1. Therefore, if the Part 2 authentication phase fails, the EAP server should delete both records User 1 and User 2. Additionally, if the user refuses to authenticate Part 2, the control point should set both records CredentialState to Denied or optionally delete both records User1 and User2.

The LinkedIdentifier may contain the characters < > & which will "break" the surrounding XML, therefore the LinkedIdentifier string must be 'escaped'. Refer to DeviceSecurity section "XML Strings as UPnP™ Parameters".

2.2.12.`LastChange`

This variable is used for eventing purposes to allow control points to receive meaningful event notifications when a record is added, deleted or changed.

`LastChange` is an evented string variable whose value is an escaped XML string (as used by the ***DeviceSecurity*** service) with the following format (white space is shown for readability but is not necessary or desirable):

```
<action>
    <fieldname>value</fieldname>…
</action>
```

where *`action`* is one of {`Add, Delete, Update`}, *`fieldname`* is one of {`Identifier, Secret, SecretType, AuthType, AuthState, CredentialState, LinkedIdentifier`}, and *`value`* is specified per the State Variables table. The `Identifier` must always be present. To prevent sending the `Secret,` which can be a large string, over the network an empty string should be sent in its place. This is intended to reduce the traffic when the `LastChange` event triggers, and also of course avoids sending sensitive information via event messages. Multiple changes to a single record can be concatenated to trigger only a single event to subscribed control points. When a record is updated, it is recommended that only those fields whose values have changed be sent, but control points should not assume this.

For example, creation of a new record might result in the following value for `LastChange`.

```
<Add>
    <Identifier>Foo</Identifier>
    <Secret/>
    <SecretType>TextPassword</SecretType>
    <AuthType>SharedSecret</AuthType>
    <AuthState>Unconfigured</AuthState>
    <CredentialState>Unconfigured</CredentialState>
</Add>
```

A subsequent change to CredentialState might result in the following value.

```
<Update>
    <Identifier>Foo</Identifier>
    <CredentialState>Pending</CredentialState>
</Update>
```

2.2.13. `LastError`

This variable is used for eventing purposes to allow control points to discover when an asynchronous error (i.e. an error that is not the direct result of a UPnP™ action) has occurred in the authentication server backend handler. For example, the EAP server might have tried to add a new record to the authentication database but failed due to lack of resources.

`LastError` is an evented string variable whose value is an escaped XML string (as used by the ***DeviceSecurity*** service) with the following format (white space is shown for readability but is not necessary or desirable):

```
<Error>
    <Code>integer-code</Code>
    <Description>error-description</Description>
</Error>
```

Where appropriate, standard UPnP™ error codes and descriptions can be used. New codes should be allocated according to the conventions described in Section 2.4.9.

2.3. Eventing and Moderation

Table 2: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
`LastChange`	Yes	No	N/A	N/A	N/A

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
`LastError`	Yes	No	N/A	N/A	N/A
Non-standard state variables implemented by an UPnP™ device vendor go here.	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

[1] Determined by N, where Rate = (Event)/(N secs).
[2] (N) * (allowedValueRange Step).

2.3.1. Event Model

Control points can use `LastChange` events to notify end-users of clients attempting to access the network for the first time, and can use `LastError` events to notify them of asynchronous errors. Additionally, control points can use events for duplication or to backup the authentication database to a control point such as a PC or into another wireless access point. Refer to the Theory of operation for further details.

Note: events alone cannot be used to duplicate a database, because they will not contain the value of the `Secret` field. A control point could, on noting that `Secret` had changed, use `GetSpecificEntry` to retrieve its value.

2.4. Actions

Table 3 lists the required and optional actions for the UPnP™ AP device. This is followed by detailed information about these actions, including short descriptions of the actions, the effects of the actions on state variables, and error codes defined by the actions.

Securing UPnP™ actions in this service is optional but strongly recommended, using UPnP™ security protocols as defined by UPnP™ Security working group. If the AP implements security for UPnP™ actions, Table 3 indicates which actions MUST be secure. The others may be implemented as secure or open. Secure actions MUST be protected for both confidentiality and integrity.

Access permissions will be inherited from the containing device (e.g., *WLANAccessPointDevice*).

Table 3: Actions

Name	Secure or Open*	Req. or Opt. [1]
`GetGenericEntry`	S	R
`GetSpecificEntry`	S	R
`AddEntry`	S	R
`UpdateEntry`	S	R
`DeleteEntry`	S	R
`GetNumberOfEntries`	S	R
`FactoryDefaultReset`	S	R
`ResetAuthentication`	S	R
Non-standard actions implemented by an UPnP™ device vendor go here.	*X*	*X*

[1] R = Required, O = Optional, X = Non-standard.

* This column is relevant if DeviceSecurity service is present in the container device

2.4.1. `GetGenericEntry`

This action retrieves authentication records one entry at a time. Control points can call this action with an incrementing array index until no more entries are found in the authentication record list. If `LastChange` is updated during a call, the process may have to start over. Entries in the array are contiguous. As entries are deleted, the array is compacted, and the evented variable `LastChange` is triggered. Authentication records are logically stored as an array and retrieved using an array index ranging from 0 to `NumberOfEntries-1`.

2.4.1.1. Arguments

Table 4: Arguments for `GetGenericEntry`

Argument	Direction	relatedStateVariable
`NewIndex`	*IN*	`NumberOfEntries`
`NewIdentifier`	*OUT*	`Identifier`
`NewSecret`	*OUT*	`Secret`
`NewSecretType`	*OUT*	`SecretType`
`NewAuthType`	*OUT*	`AuthType`
`NewAuthState`	*OUT*	`AuthState`
`NewCredentialState`	*OUT*	`CredentialState`
`NewDescription`	*OUT*	`Description`
`NewMACAddress`	*OUT*	`MACAddress`
`NewCredentialDuration`	*OUT*	`CredentialDuration`
`NewLinkedIdentifier`	*OUT*	`LinkedIdentifier`

2.4.1.2. Dependency on State (if any)

2.4.1.3. Effect on State (if any)

None.

2.4.1.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
713	SpecifiedArrayIndexInvalid	The specified array index is out of bounds.

2.4.2. `GetSpecificEntry`

This action retrieves one authentication record entry specified by the input parameter `NewIdentifierKey`. Authentication records are logically stored as an array in the authentication record list and can be retrieved using their Identifier as a unique value.

2.4.2.1. Arguments

Table 5: Arguments for `GetSpecificEntry`

Argument	Direction	relatedStateVariable
`NewIdentifierKey`	*IN*	`Identifier`
`NewIdentifier`	*OUT*	`Identifier`
`NewSecret`	*OUT*	`Secret`
`NewSecretType`	*OUT*	`SecretType`
`NewAuthType`	*OUT*	`AuthType`
`NewAuthState`	*OUT*	`AuthState`
`NewCredentialState`	*OUT*	`CredentialState`
`NewDescription`	*OUT*	`Description`
`NewMACAddress`	*OUT*	`MACAddress`
`NewCredentialDuration`	*OUT*	`CredentialDuration`
`NewLinkedIdentifier`	*OUT*	`LinkedIdentifier`

2.4.2.2. Dependency on State (if any)

2.4.2.3. Effect on State (if any)

None.

2.4.2.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
605	String Argument Too Long	A string argument is too long for the device to handle properly.

702	IdentifierKeyNotPresent	The record corresponding to input Identifier key is not found in the authentication record list.

2.4.3. `AddEntry`

This action creates a new authentication record.

2.4.3.1. Arguments

Table 6: Arguments for `AddEntry`

Argument	Direction	relatedStateVariable
`NewIdentifier`	*IN*	`Identifier`
`NewSecret`	*IN*	`Secret`
`NewSecretType`	*IN*	`SecretType`
`NewAuthType`	*IN*	`AuthType`
`NewAuthState`	*IN*	`AuthState`
`NewCredentialState`	*IN*	`CredentialState`
`NewDescription`	*IN*	`Description`
`NewMACAddress`	*IN*	`MACAddress`
`NewCredentialDuration`	*IN*	`CredentialDuration`
`NewLinkedIdentifier`	*IN*	`LinkedIdentifier`
`NewNumberOfEntries`	*OUT*	`NumberOfEntries`

2.4.3.2. Dependency on State (if any)

2.4.3.3. Effect on State (if any)

When adding a new record the `LastChange` variable is evented including the new fields and values.

2.4.3.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.

501	Action Failed	See UPnP™ Device Architecture section on Control.
605	String Argument Too Long	A string argument is too long for the device to handle properly.
701	EntryAlreadyPresent	If an existing record with Identifier already exists in the database return this error.

2.4.4. `UpdateEntry`

This action modifies an existing authentication record.

2.4.4.1. Arguments

Table 7: Arguments for `UpdateEntry`

Argument	Direction	relatedStateVariable
`NewIdentifier`	*IN*	`Identifier`
`NewSecret`	*IN*	`Secret`
`NewSecretType`	*IN*	`SecretType`
`NewAuthType`	*IN*	`AuthType`
`NewAuthState`	*IN*	`AuthState`
`NewCredentialState`	*IN*	`CredentialState`
`NewDescription`	*IN*	`Description`
`NewMACAddress`	*IN*	`MACAddress`
`NewCredentialDuration`	*IN*	`CredentialDuration`
`NewLinkedIdentifier`	*IN*	`LinkedIdentifier`
`NewNumberOfEntries`	*OUT*	`NumberOfEntries`

2.4.4.2. Dependency on State (if any)

2.4.4.3. Effect on State (if any)

The modified fields and values are evented via the `LastChange` event.

2.4.4.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
501	Action Failed	See UPnP™ Device Architecture section on Control.
605	String Argument Too Long	A string argument is too long for the device to handle properly.
714	EntryNotPresent	If existing record with Identifier does not exist return this error.

2.4.5. `DeleteEntry`

This action deletes an authentication record specified by InIdentifier.

2.4.5.1. Arguments

Table 8: Arguments for `DeleteEntry`

Argument	Direction	relatedStateVariable
`NewIdentifier`	*IN*	`Identifier`
`NewNumberOfEntries`	*OUT*	`NumberOfEntries`

2.4.5.2. Dependency on State (if any)

2.4.5.3. Effect on State (if any)

As each entry is deleted, the array is compacted, and the evented variable `LastChange` is triggered. `LastChange` only includes the ActionField set to Delete followed by the `Identifier` field.

2.4.5.4. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
605	String Argument Too Long	A string argument is too long for the device to handle properly.
702	IdentifierKeyNotPresent	The record corresponding to input Identifier key is not found in the authentication record list.

2.4.6. `GetNumberOfEntries`

This action retrieves the value `NumberOfEntries`.

2.4.6.1. Arguments

Table 9: Arguments for `GetNumberOfEntries`

Argument	Direction	relatedStateVariable
`NewNumberOfEntries`	*OUT*	`NumberOfEntries`

2.4.6.2. Dependency on State (if any)

2.4.6.3. Effect on State (if any)

2.4.6.4. Errors

ErrorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.

2.4.7. `FactoryDefaultReset`

This action resets the service to its factory defaults. On successful completion, the authentication database will contain only those entries that were pre-defined by the vendor, if any.

It is recommended that the relevant `LastChange` events be posted when resetting to factory defaults.

All authenticated sessions using credentials in the credential store must be disassociated.

FactoryDefaultReset is processed by the LinkAuthentication service when a containing device has its DeviceSecurity FactoryDefaultReset invoked. Additionally, the IGD working committee has determined that the LinkAuthentication service is a child service of the WLANConfiguration service when contained in the WLANAccessPointDevice. Therefore when WLANConfiguration FactoryDefaultReset is invoked, the implementation of WLANConfiguration FactoryDefaultReset must invoke the implementation of LinkAuthentication FactoryDefaultReset.

2.4.7.1. Arguments

None

2.4.7.2. Dependency on State (if any)

2.4.7.3. Effect on State (if any)

2.4.7.4. Errors

ErrorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
501	Action Failed	See UPnP™ Device Architecture section on Control.

2.4.8. `ResetAuthentication`

This action performs a soft reset of the service. This forces all clients to re-authenticate and guarantees that `CredentialState` and `AuthState` are consistent, as follows.

- All authenticated sessions using credentials in the credential store must be disassociated.
- All non-permanent authentication database entries (i.e. those that have non-zero `CredentialDuration` fields) are deleted.
- All entries with a `CredentialState` other than *`Accepted`* (i.e. those whose secrets have never been validated) are deleted.
- `AuthState` is set to *`Unconfigured`* for all remaining entries, and the EAP server proceeds with re-authentication.

It is required that the relevant `LastChange` events be posted when performing a soft reset.

This soft reset logic is also executed on each reboot.

The IGD working committee has determined that the LinkAuthentication service is an implied child service of the WLANConfiguration service when contained in the WLANAccessPointDevice. Therefore when WLANConfiguration ResetAuthentication is invoked, the implementation of WLANConfiguration ResetAuthentication must invoke the implementation of LinkAuthentication ResetAuthentication.

2.4.8.1. Arguments

None

2.4.8.2. Dependency on State (if any)

2.4.8.3. Effect on State (if any)

2.4.8.4. Errors

ErrorCode	errorDescription	Description
402	Invalid Args	See UPnP™ Device Architecture section on Control.
501	Action Failed	See UPnP™ Device Architecture section on Control.

2.4.9. Common Error Codes

The following table lists error codes common to actions for this service type. If an action results in multiple errors, the most specific error should be returned.

Table 10: Common Error Codes

errorCode	errorDescription	Description
401	Invalid Action	See UPnP™ Device Architecture section on Control.
402	Invalid Args	See UPnP™ Device Architecture section on Control.
404	Invalid Var	See UPnP™ Device Architecture section on Control.
501	Action Failed	See UPnP™ Device Architecture section on Control.

errorCode	errorDescription	Description
600-699	TBD	Common action errors. Defined by UPnP™ Forum Technical Committee.
701-799		Common action errors defined by the UPnP™ Forum working committees.
800-899	*TBD*	*(Specified by UPnP™ device vendor.)*

2.5. Theory of Operation

The LinkAuthentication service provides a mechanism for control points to access a per-client credential store. It is strongly recommended that actions of this service are controlled using *DeviceSecurity:1.0* as defined in the UPnP™ Security working group. The term "per-client" means that users and/or WLAN client devices have unique credentials, as opposed to using a single network-wide common credential. The credential store and this service can be used for any authentication mechanism but one of the main uses is with 802.1x. The following sections describe how it works with 802.1x, but is also applicable to other authentication processes, for example, any client can store their public keys in the credential store keyed with its unique Identifier.

2.5.1. 802.1x introduction

IEEE-802.1x is a request-response framework used for physical port-level access (as in an Ethernet switch) to local area networks. There are three roles in 802.1x: 1) the supplicant, 2) the authenticator, and 3) the authentication server. The supplicant is the client attempting to authenticate to gain network access. The authenticator is the device enforcing authentication, and the authentication server provides the mechanism to check the supplicant's credentials on behalf of the authenticator. 802.1x uses EAP (Extensible Authentication Protocol) to exchange authentication messages between the client requesting authentication and the authentication server. Several EAP authentication protocols have been specified including EAP-TLS, Protected EAP, EAP-TTLS and others, see RFC 2284 for details. Per the 802.1x specification – "an Authenticator and an Authentication server can be co-located within the same System, allowing that System to perform the authentication function without the need for communication with an external server." – IEEE Std 802.1x-2001 §6.1.

IEEE-802.1x has been adapted to 802.11 by the industry where the wireless station (the supplicant) authenticates with an authentication server via an AP (the authenticator). This is typically applied in enterprise networks where the authentication protocols rely on 'back-end' authentication servers (e.g. RADIUS). The servers are linked to user databases that are commonly found in large corporate class networks administered by Information Technology staff. However, one cannot assume that level of administrative expertise in the home environment. The ***LinkAuthentication*** service provides a simple and sufficient authentication server framework that can be updated via UPnP™ actions. Although the ***LinkAuthentication*** service is assumed generally to be co-located with the AP, it is possible that it may be running on a separate device on the LAN.

2.5.2. High level intended operation

Access points implementing 802.1x based authentication require the use of authentication server(s). Additionally, 802.1x client devices such as embedded devices with built-in or preconfigured unique ("per-client") credentials such as passwords or certificates need to have their credentials initially verified, added, and stored in the authentication server. The ***LinkAuthentication*** service can be used by an AP that supports 802.1x and does not have an authentication server such as RADIUS available (external to the AP device) on the network. This service provides a means for UPnP™ Control Points to store and access the authentication data and is essentially a front-end API to manipulate the authentication database the AP uses for 802.1x authentication. It also provides a mechanism to inform the control point via UPnP™ events that a particular entry needs to be validated. The authentication data store can be present anywhere on the network. For instance, it can be stored locally on the AP or in an embedded device such as an IGD that can host the ***LinkAuthentication*** service and the database. In the latter case, the AP could access the database on the IGD via RADIUS and the user could manipulate the database via the ***LinkAuthentication*** service. RFC 2716 (EAP-TLS) uses the term "EAP Server" to denote the "ultimate endpoint" actually performing the authentication process. This term is also used below.

The ***LinkAuthentication*** service maintains records with state variables to aid control points (via end-user interaction) in verifying, adding, and storing per-user or per-device unique credentials. There are two entities making changes to records in the database. For instance, in the case of the ***LinkAuthentication*** Service and the database residing locally in the AP, one entity is the 802.1x EAP Server while the other entity is the UPnP™ Control Point. Both are able to update fields in the authentication database and respond to changes. Any modification to the client records generates the appropriate UPnP™ events (see description for `LastChange` variable).

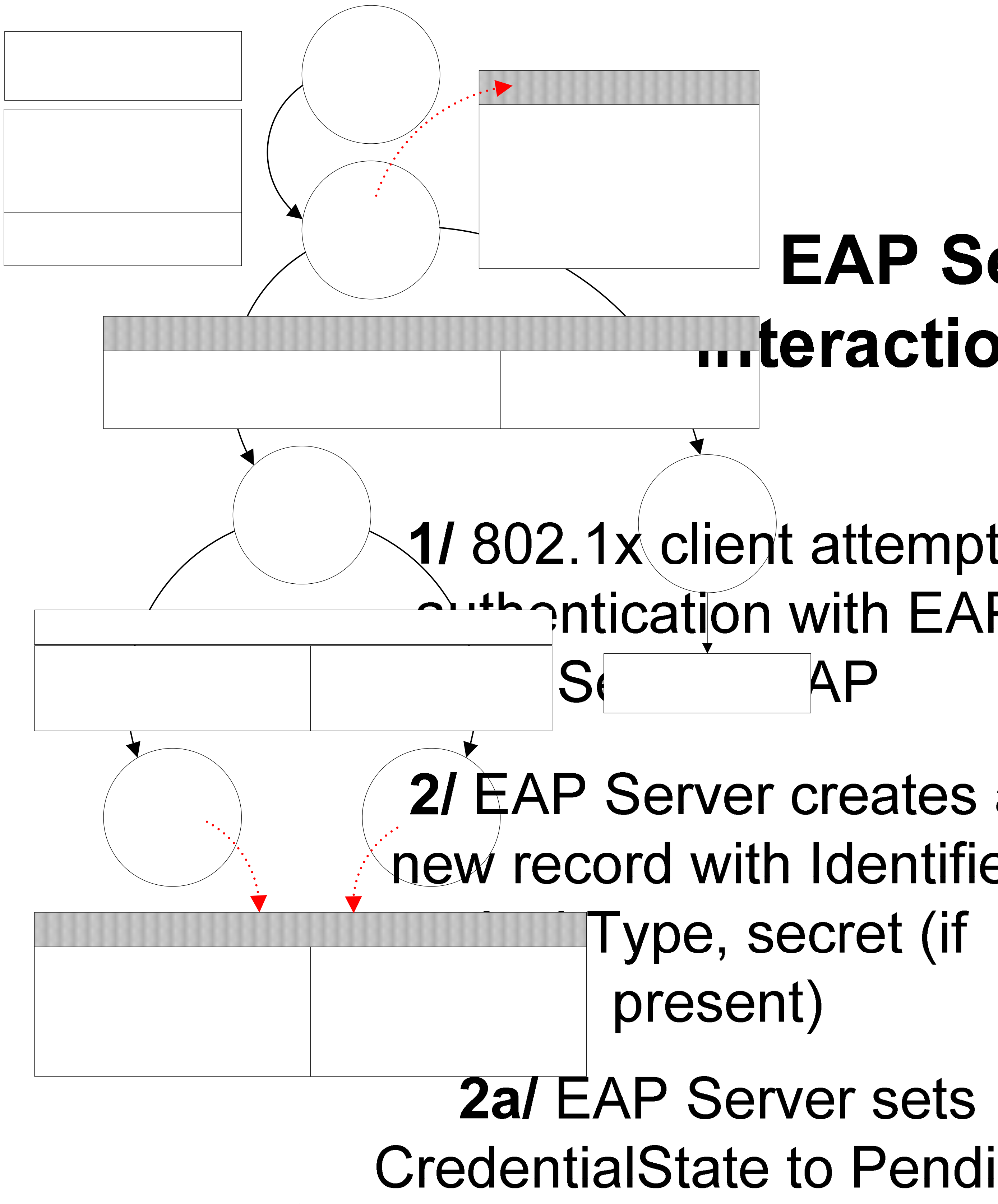
This service supports the most commonly utilized authentication protocols specified via 802.1x EAP. The EAP protocols are generalized in this document into two types. The first type requires that the public key of a client be verified and the other requires the end user to enter a secret such as a password. For example, with EAP-TLS the client's certificate (credentials) is sent to the AP and then to the Control Point. It is assumed that the user at the Control Point has a means to validate this certificate. Another EAP type is Protected EAP which involves EAP-TLS tunnel with the use of an inner protocol such as MSCHAPv2. In this case, a shared secret / password is entered via the Control Point and stored in the database. The secret would already be present in the client in permanent storage such as disk or put in flash memory at the time of manufacture. For example, the secret on a closed/embedded device could be accessed by the user by reading a sticker on the device or printed documentation. Once the user has entered the secret into the Control Point it is sent to the AP and used to authenticate the client.

2.5.3. Detailed level operation

The 802.1x EAP process starts with a request by the EAP Server for an EAP Identity string from the enrolling client. The EAP Server accesses the authentication records using the EAP Identity string to retrieve the record with the corresponding Identifier field. Therefore all devices accessing the wireless network must have unique EAP Identity strings. The EAP process then progresses to select an authentication method (EAP Type). Once an EAP type has been agreed upon by the server and client, the EAP authentication method is executed. It is expected that the EAP Server will refer to the corresponding client record with the given Identifier during the authentication process and verify that the credentials used for authentication match those in the authentication records.

If the EAP Identity string does not match any client `Identifiers`, then a new record is created by the EAP Server. The EAP server should populate the record with the `Identifier` field set to the EAP Identity string from the enrolling client and set the `AuthType` corresponding to the EAP authentication mode (such as EAP-TLS) selected by the EAP server . The EAP server should also update the MAC address field in the record. If credential exchanges have occurred by this time as in the case of EAP-TLS, the EAP server should populate the `Secret` field appropriately as well. The EAP server can now set the `CredentialState` to *`Pending`* which will trigger `LastChange` and alert control points of an enrolling client. The control points will prompt the user to accept or deny the client's credentials. If the user selects *`Denied`* the control point invokes `UpdateEntry` action, halting the client authentication process. This results in an EAP Failure sent to the client by the AP. If the user selects *`Accepted`* the client authentication proceeds and the EAP server verifies the credential in the secret field is actually being used for client authentication. At the conclusion of the authentication process the EAP server updates the `AuthState` field.

However, if the EAP Identity does match a client `Identifier` and the `CredentialState` is *`Accepted`* the EAP server proceeds with the authentication process and verifies the credential in the secret field is actually being used for client authentication. If `CredentialState` is already *`Denied`* the client authentication process halts and an EAP Failure is sent to the client. At the conclusion of the authentication process the EAP server updates the `AuthState` field.


EAP Se
Interactio
1/ 802.1x client attempt
authentication with EAF
Se AP
2/ EAP Server creates
new record with Identifie
Type, secret (if
present)
2a/ EAP Server sets
CredentialState to Pendi

2.5.4. Record format

Each record is uniquely addressed via the `Identifier` field. As an example, the following authentication records contain two device records with `CredentialState` set to *`Accepted`*, named Dev1 and Dev2. For this example these two devices have already authenticated via EAP-TLS and the AP has stored the devices' public keys. Since the `AuthState` is *`Authenticated`*, when either of these devices sends its EAP Identity, the AP will find the matching record and compare the contents of the Secret field with the public key that the client sends during the TLS handshake phase. If they match the AP will continue the TLS handshake; if they do not match the AP should not continue the TLS handshake.

Identifier	Secret	Secret Type	Auth Type	Cred State	Auth State	Descrip.	MAC addr.	Cred Duration
Dev 1	Key1	Public Key	*Validate Credentials*	Accepted	Succeeded		MAC 1	0
Dev 2	Key2	Public Key	*Validate Credentials*	Accepted	Succeeded		MAC 2	0
…	…	…	…		…	…	…	…

With EAP-TLS the Identifier field and the public key will be sent in the clear over the wireless link during the TLS handshake phase. TLS will ensure that the client sending a public key has a matching private key. It is important to store and compare the credential for EAP-TLS, in this example the public key, in the authentication records with the client's public key since a rogue device can masquerade as Dev1 by sending Dev1 as its identifier and use Dev1's MAC address. Additionally, the rogue device can easily create a valid public key / private key pair and send the public key to the AP during the TLS handshake phase. If the rogue client's public key were not matched against Dev1's user authenticated Secret (its public key) in the authentication records the AP EAP server would challenge the rogue device using the rogue device's public key and the rogue client would successfully complete the TLS handshake. Therefore, to prevent rogue devices from successfully masquerading using their own keys, the public key or other credential should be maintained in the authentication records and matched during authentication.

2.5.5. Example using client certificate credentials

In this example, an 802.1x based client device is attempting to gain network access. Suppose the client and AP are both configured with certificates from a common certificate authority. The EAP-TLS process will exchange challenges and certificates. The certificates are typically validated in terms of expiration dates, and the certificate chain to the root certificate authority. In this case the AP certificate is validated by the client which then proceeds with establishing the TLS tunnel. The first time the client associates the AP is expected to find that the client's certificate (or hash of the public key in the cert) is not in the client credential records. In this case the EAP server creates and populates a record with the `Identifier, Secret` and `AuthType` of *`ValidateCredentials`*, then the EAP server sets the `CredentialState` to *`Pending`*. The CP receives the event(s) and informs the user that a device with an identifier of (a string such as serialnumber@manufacturer) and a "secret" of (some string) is requesting network access. The secret in this case is printed on the bottom of the client device. This is a hash of the public key. The user is expected to look under the device and match the number on the sticker to the number on the control point. If the user deems that the numbers match the user will click on the "accept" button and optionally enter a time interval for temporary network access. The CP will set `CredentialState` to *`Accepted`* and the EAP server will then continue the TLS session ensuring the public key from the client is indeed used for the TLS session. If the EAP-TLS authentication succeeds or fails the `AuthState` variable will be updated informing the control point. Note the client has been authenticated by the AP (by checking the hash on the device).

The above example relies on the clients and AP having pre-installed certificates. This is not practical for the home environment especially for small embedded devices without a user interface. This limitation will have to be addressed with some authentication method that provides mutual authentication but does not require certificate validation.

2.5.6. Limitations on Pending Records

The EAP server must delete `Pending` records when the 802.1x authentication process times out or is aborted. The state machines and time values are specified in the IEEE Std 802.1x-2001.

An attacker could attempt authentication with randomly selected EAP Identity strings and overwhelm the credential store by effectively creating too many `Pending` records. Each implementation should place an implementation dependent limit on the maximum number of `Pending` records.

3. XML Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>GetGenericEntry</name>
            <argumentList>
                <argument>
                    <name>NewIndex</name>
                    <direction>in</direction>
                    <relatedStateVariable>NumberOfEntries</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewIdentifier</name>
                    <direction>out</direction>
                    <relatedStateVariable>Identifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewSecret</name>
                    <direction>out</direction>
                    <relatedStateVariable>Secret</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewSecretType</name>
                    <direction>out</direction>
                    <relatedStateVariable>SecretType</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewAuthType</name>
                    <direction>out</direction>
                    <relatedStateVariable>AuthType</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewAuthState</name>
                    <direction>out</direction>
                    <relatedStateVariable>AuthState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialState</name>
                    <direction>out</direction>
                    <relatedStateVariable>CredentialState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewDescription</name>
                    <direction>out</direction>
                    <relatedStateVariable>Description</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewMACAddress</name>
                    <direction>out</direction>
                    <relatedStateVariable>MACAddress</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialDuration</name>
                    <direction>out</direction>
                    <relatedStateVariable>CredentialDuration</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewLinkedIdentifier</name>
                    <direction>out</direction>
                    <relatedStateVariable>LinkedIdentifier</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
```

```
            <name>GetSpecificEntry</name>
            <argumentList>
                <argument>
                    <name>NewIdentifierKey</name>
                    <direction>in</direction>
                    <relatedStateVariable>Identifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewIdentifier</name>
                    <direction>out</direction>
                    <relatedStateVariable>Identifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewSecret</name>
                    <direction>out</direction>
                    <relatedStateVariable>Secret</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewSecretType</name>
                    <direction>out</direction>
                    <relatedStateVariable>SecretType</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewAuthType</name>
                    <direction>out</direction>
                    <relatedStateVariable>AuthType</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewAuthState</name>
                    <direction>out</direction>
                    <relatedStateVariable>AuthState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialState</name>
                    <direction>out</direction>
                    <relatedStateVariable>CredentialState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewDescription</name>
                    <direction>out</direction>
                    <relatedStateVariable>Description</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewMACAddress</name>
                    <direction>out</direction>
                    <relatedStateVariable>MACAddress</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialDuration</name>
                    <direction>out</direction>
                    <relatedStateVariable>CredentialDuration</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewLinkedIdentifier</name>
                    <direction>out</direction>
                    <relatedStateVariable>LinkedIdentifier</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>AddEntry</name>
            <argumentList>
                <argument>
                    <name>NewIdentifier</name>
                    <direction>in</direction>
                    <relatedStateVariable>Identifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewSecret</name>
                    <direction>in</direction>
                    <relatedStateVariable>Secret</relatedStateVariable>
            </argument>
            <argument>
                <name>NewSecretType</name>
                <direction>in</direction>
                <relatedStateVariable>SecretType</relatedStateVariable>
            </argument>
            <argument>
                <name>NewAuthType</name>
                <direction>in</direction>
                <relatedStateVariable>AuthType</relatedStateVariable>
            </argument>
            <argument>
                <name>NewAuthState</name>
                <direction>in</direction>
                <relatedStateVariable>AuthState</relatedStateVariable>
            </argument>
            <argument>
                <name>NewCredentialState</name>
                <direction>in</direction>
                <relatedStateVariable>CredentialState</relatedStateVariable>
            </argument>
            <argument>
                <name>NewDescription</name>
                <direction>in</direction>
                <relatedStateVariable>Description</relatedStateVariable>
            </argument>
            <argument>
                <name>NewMACAddress</name>
                <direction>in</direction>
                <relatedStateVariable>MACAddress</relatedStateVariable>
            </argument>
            <argument>
                <name>NewCredentialDuration</name>
                <direction>in</direction>
                <relatedStateVariable>CredentialDuration</relatedStateVariable>
            </argument>
            <argument>
                <name>NewLinkedIdentifier</name>
                <direction>in</direction>
                <relatedStateVariable>LinkedIdentifier</relatedStateVariable>
            </argument>
            <argument>
                <name>NewNumberOfEntries</name>
                <direction>out</direction>
                <relatedStateVariable>NumberOfEntries</relatedStateVariable>
            </argument>
        </argumentList>
    </action>
    <action>
        <name>UpdateEntry</name>
        <argumentList>
            <argument>
                <name>NewIdentifier</name>
                <direction>in</direction>
                <relatedStateVariable>Identifier</relatedStateVariable>
            </argument>
            <argument>
                <name>NewSecret</name>
                <direction>in</direction>
                <relatedStateVariable>Secret</relatedStateVariable>
            </argument>
            <argument>
                <name>NewSecretType</name>
                <direction>in</direction>
                <relatedStateVariable>SecretType</relatedStateVariable>
            </argument>
            <argument>
                <name>NewAuthType</name>
                <direction>in</direction>
                <relatedStateVariable>AuthType</relatedStateVariable>
            </argument>
```

```
                <argument>
                    <name>NewAuthState</name>
                    <direction>in</direction>
                    <relatedStateVariable>AuthState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialState</name>
                    <direction>in</direction>
                    <relatedStateVariable>CredentialState</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewDescription</name>
                    <direction>in</direction>
                    <relatedStateVariable>Description</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewMACAddress</name>
                    <direction>in</direction>
                    <relatedStateVariable>MACAddress</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewCredentialDuration</name>
                    <direction>in</direction>
                    <relatedStateVariable>CredentialDuration</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewLinkedIdentifier</name>
                    <direction>in</direction>
                    <relatedStateVariable>LinkedIdentifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewNumberOfEntries</name>
                    <direction>out</direction>
                    <relatedStateVariable>NumberOfEntries</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>DeleteEntry</name>
            <argumentList>
                <argument>
                    <name>NewIdentifier</name>
                    <direction>in</direction>
                    <relatedStateVariable>Identifier</relatedStateVariable>
                </argument>
                <argument>
                    <name>NewNumberOfEntries</name>
                    <direction>out</direction>
                    <relatedStateVariable>NumberOfEntries</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>GetNumberOfEntries</name>
            <argumentList>
                <argument>
                    <name>NewNumberOfEntries</name>
                    <direction>out</direction>
                    <relatedStateVariable>NumberOfEntries</relatedStateVariable>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>FactoryDefaultReset</name>
        </action>
        <action>
            <name>ResetAuthentication</name>
        </action>
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="no">
        <name>NumberOfEntries</name>
        <dataType>ui2</dataType>
        <defaultValue>0</defaultValue>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>Identifier</name>
        <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>Secret</name>
        <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>SecretType</name>
        <dataType>string</dataType>
        <allowedValueList>
            <allowedValue>TextPassword</allowedValue>
            <allowedValue>X509Certificate</allowedValue>
            <allowedValue>PublicKey</allowedValue>
            <allowedValue>PublicKeyHash160</allowedValue>
        </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>AuthType</name>
        <dataType>string</dataType>
        <allowedValueList>
            <allowedValue>SharedSecret</allowedValue>
            <allowedValue>ValidateCredentials</allowedValue>
        </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>AuthState</name>
        <dataType>string</dataType>
        <defaultValue>Unconfigured</defaultValue>
        <allowedValueList>
            <allowedValue>Unconfigured</allowedValue>
            <allowedValue>Failed</allowedValue>
            <allowedValue>Succeeded</allowedValue>
        </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>CredentialState</name>
        <dataType>string</dataType>
        <defaultValue>Unconfigured</defaultValue>
        <allowedValueList>
            <allowedValue>Unconfigured</allowedValue>
            <allowedValue>Pending</allowedValue>
            <allowedValue>Accepted</allowedValue>
            <allowedValue>Denied</allowedValue>
        </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>Description</name>
        <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>MACAddress</name>
        <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>CredentialDuration</name>
        <dataType>ui4</dataType>
        <defaultValue>0</defaultValue>
    </stateVariable>
    <stateVariable sendEvents="yes">
        <name>LastChange</name>
        <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
        <name>LinkedIdentifier</name>
        <dataType>string</dataType>
        </stateVariable>
        <stateVariable sendEvents="yes">
            <name>LastError</name>
            <dataType>string</dataType>
        </stateVariable>
    </serviceStateTable>
</scpd>
```

4. Test

No semantic tests have been defined for this service.