# USP Layer Security

USP contains mechanisms for Authentication and Authorization, and Encryption.

## Authentication

Authentication of Endpoints is done using X.509 certificates as defined in [RFC 5280](https://tools.ietf.org/html/rfc5280) and [RFC 6818](https://tools.ietf.org/html/rfc6818). These certificates, at a minimum, need to be usable for [MTP security](/specification/mtp/#securing_mtps) with TLS or DTLS protocols.

In order to support various authentication models (e.g., trust Endpoint identity and associated certificate on first use; precise Endpoint identity is indicated in a certificate issued by a trusted Certificate Authority; trust that Endpoint is a member of a trusted domain as verified by a trusted Certificate Authority), this Working Text provides guidance based on conditions under which the Endpoint is operating, and on the Endpoint's policy for storing certificates of other Endpoints or just certificates of trusted CAs.

**R-SEC.0** - The Agent MUST have a Controller's certificate information prior to establishing a cryptographically protected connection.

TLS and DTLS both have handshake mechanisms that allow for exchange of certificate information.

Whether a Controller requires Agent certificates is left up to the Controller implementation.

### Authentication with RBAC (Role Based Access Control)

It is expected that Agents will have some sort of Access Control List (ACL) that will define different levels of authorization for interacting with the Agent's data model. This Working Text refers to different levels of authorization as "Roles". The Agent may be so simple as to only support a single Role that gives full access to its data model; or it may have just an "untrusted" Role and a "full access" Role. Or it may be significantly more complex with, for example, "untrusted" Role, different Roles for parents and children in a customer household, and a different Role for the service provider Controller. These Roles may be fully defined in the Agent's code, or Role definition may be allowed via the data model.

**R-SEC.1** - An Agent MUST confirm a Controller has the necessary permissions to perform the requested actions in a Message prior to performing that action.

## Trusted Certificate Authorities

An Endpoint can have a configured list of trusted Certificate Authority (CA) root certificates. The Agent policy may trust the CA to approve validated Controllers to have a specific default Role, or the policy may simply trust the CA to validate the Controller identity. The Controller policy may require the CA to validate Agent identity before the Controller tries to communicate with the Agent.

**R-SEC.2** - To validate against a root certificate, the Endpoint MUST contain one or more trusted root certificates that are either pre-loaded in the Endpoint or provided to the Endpoint by a secure means.

This secure means can accomplished through USP (see section below on Data Model Elements), or through a mechanism external to USP.

**R-SEC.3** - Where a CA is trusted to validate Controller identity, the Agent MUST ensure the URN form of the Controller Endpoint ID is in the Controller certificate subjectaltName with a type uniformResourceIdentifier attribute, and this matches the USP Header from\_id.

**R-SEC.4** - Where a CA is trusted to approve a Controller Role, the Agent MUST either ensure the URN form of the Controller Endpoint ID (that matches the USP Header from\_id) is in the Controller certificate subjectaltName with a type uniformResourceIdentifier attribute (with or without wildcards in the instance-id portion of the Endpoint ID), or perform the following validation:

* If the host portion of the Controller URL is a DNS name, this MUST be done according to the principles of [RFC 6125](https://tools.ietf.org/html/rfc6125), using the host portion of the Controller URL as the reference identifier.
* If the host portion of the Controller URL is an IP address, this MUST be done by comparing the IP address against any presented identifiers that are IP addresses.

*Note - the terms “reference identifier” and “presented identifier” are defined in* [*RFC 6125*](https://tools.ietf.org/html/rfc6125)*.* Note - wildcard certificates are permitted as described in [RFC 6125](https://tools.ietf.org/html/rfc6125).

**R-SEC.5** An Agent capable of obtaining absolute time SHOULD wait until it has accurate absolute time before contacting a Controller. If an Agent for any reason is unable to obtain absolute time, it can contact the Controller without waiting for accurate absolute time. If an Agent chooses to contact a Controller before it has accurate absolute time (or if it does not support absolute time), it MUST ignore those components of the Controller certificate that involve absolute time, e.g. not-valid-before and not-valid-after certificate restrictions.

## Self-Signed Certificates

**R-SEC.6** - An Endpoint that generates a self-signed certificate MUST place the URN form of its Endpoint ID in a certificate subjectaltName with a type uniformResourceIdentifier attribute.

Self-signed certificates supplied by Controllers can only be meaningfully used in cases where a person is in a position to provide Authorization (what Role the Controller is trusted to have). Whether or not an Agent allows self-signed certificates from a Controller is a matter of Agent policy.

**R-SEC.7** - If an Agent allows Controllers to provide self-signed certificates, the Agent MUST assign such Controllers an "untrusted" Role on first use.

That is, the Agent will trust the certificate for purpose of encryption, but will heavily restrict what the Controller is authorized to do.

**R-SEC.8** - If an Agent allows Controllers to provide self-signed certificates, the Agent MUST have a means of allowing an external entity to change the Role of each such Controller.

Controller policy related to trust of Agent self-signed certificates is left to the Controller. Controllers may be designed to refuse self-signed certificates (thereby refusing to control the Agent), they may have a means of allowing a person to approve controlling the Agent via the Controller, or they may automatically accept the Agent.

**R-SEC.9** - An Endpoint that accepts self-signed certificates MUST maintain the association of accepted certificate public keys and Endpoint IDs.

Self-signed certificates require a “trust on first use” policy when using them to authenticate an Endpoint's identity. An external entity (a trusted Controller or user) can then authorize the authenticated Endpoint to have certain permissions. Subsequent to the first use, this same self-signed certificate can be trusted to establish the identity of that Endpoint. However, authentication of the Endpoint can only be subsequently trusted if the association of certificate to identity is remembered (i.e., it is known this is the same certificate that was used previously by that Endpoint). If it is not remembered, then every use is effectively a first use and would need to rely on an external entity to indicate permissions every time.

## Agent certificates

**R-SEC.10** - When certificates are used to authenticate the Agent to a Controller, the subjectaltName MUST contain either:

* the URN form of the Agent Endpoint ID with a type uniformResourceIdentifier attribute.
* the URN form of an Endpoint ID with a type uniformResourceIdentifier attribute, and with wildcards such that all Agent Endpoint IDs covered by the certificate fall within the wildcarded Endpoint ID.

**R-SEC.11** - If a single certificate is shared among multiple Agents, those Agents MUST include a wild-carded instance-id in the Endpoint ID in the subjectaltName with identical authority-scheme and authority-id.

**R-SEC.12** - If the Agent does not have a CA-issued certificate, it MUST support use of a self-signed certificate. See requirements for Endpoints using [self-signed certificates](#self-signed-certificates).

## Challenge Strings and Images

It is possible for the Agent to allow an external entity to change a Controller Role by means of a Challenge string or image. This Challenge string or image can take various forms, including having a user supply a passphrase or a PIN. Such a string could be printed on the Agent packaging, or supplied by means of a SMS to a phone number associated with the user account. These Challenge strings or images can be done using USP operations. Independent of how challenges are accomplished, following are some basic requirements related to Challenge strings and images.

**R-SEC.13** - The Agent MAY have factory-default Challenge value(s) (strings or images) in its configuration.

**R-SEC.14** - A factory-default Challenge value MUST be unique to the Agent. Re-using the same passphrase among multiple Agents is not permitted.

**R-SEC.15** - A factory-default Challenge value MUST NOT be derivable from information the Agent communicates about itself using any protocol at any layer.

**R-SEC.16** - The Agent MUST limit the number of tries for the Challenge value to be supplied successfully.

**R-SEC.17** - The Agent SHOULD have policy to lock out all use of Challenge values for some time, or indefinitely, if the number of tries limit is exceeded.

## Theory of operations

The following theory of operations relies on objects, parameters, events, and operations from the LocalAgent Object of the [Device:2 Data Model for TR-069 Devices and USP Agents](https://github.com/BroadbandForum/usp/tree/master/data-model).

### Data Model Elements

These data model elements play a role in reporting on and allowing control of trusted Controllers and the permissions they have to read and write parts of the Agent’s data model, and allowing an Agent to establish trust with a Controller.

* LocalAgent.Controller.{i}.AssignedRole parameter
* LocalAgent.Controller.{i}.InheritedRole parameter
* LocalAgent.Controller.{i}.Credential parameter

From component CoAP:

* Objects LocalAgent.MTP.{i}.CoAP., and LocalAgent.Controller.{i}.MTP.{i}.CoAP.
* Parameters EnableEncryption, ValidatePeerCertificate, and CheckPeerID.

From component ControllerTrust:

* Object LocalAgent.ControllerTrust.
* Parameters UntrustedRole, BannedRole
* Commands RequestChallenge(), ChallengeResponse()
* Object LocalAgent.ControllerTrust.Role.{i}.
* Object LocalAgent.ControllerTrust.Credential.{i}.
* Object LocalAgent.ControllerTrust.Challenge.{i}.

For brevity, Device.LocalAgent is not placed in front of all further object references in this Security section. However, all objects references are under Device.LocalAgent. This section does not describe use of parameters under other top level components (e.g., it does not describe parameters under Device.RemoteAgent).

### Roles (Access Control)

Controller permissions are conveyed in the data model through Roles.

#### Role Definition

A Role is expressed in the data model through use of the ControllerTrust.Role.{i}. object. This object can have multiple entries with the same Role parameter value. All entries with the same Role value combine together to define the Role. Each entry in this object identifies the Role it applies to, the target (data model path that the related permissions apply to), permissions related to parameters, objects, instantiated objects, and commands identified by the Target parameter, and the relative order of precedence among entries for the same Role (the larger value of this parameter takes priority over an entry with a smaller value in the case of overlapping Target entries for the same Role).

The permissions of a Role for the specified Target entries are described by ParameterPermissions, ObjectPermissions, InstantiatedObjectPermissions, and CommandEventPermissions parameters. Each of these is expressed as a string of 4 characters where each character represents a permission ("r" for Read, "w" for Write, "x" for Execute", and "n" for Notify). The string is always in the same Order value (rwxn) and the lack of a permission is signified by a "-" character (e.g., r--n). How these permissions are applied to parameters, objects, and various Messages is described in the data model description of these parameters.

An Agent that wants to allow Controllers to define and modify Roles will implement the ControllerTrust.Role.{i}. object with all of the parameters listed in the data model. In order for a Controller to define or modify Role entries, it will need to be assigned a Role that gives it the necessary permission. Care should be taken to avoid defining this Role’s permissions such that an Controller with this Role can modify the Role such that it can no longer make future modifications to the ControllerTrust.Role.{i}. object.

A simple Agent that only wants to inform Controllers of pre-defined Roles (with no ability to modify or define additional Roles) can implement the ControllerTrust object with read-only data model definition for all entries and parameters. A simple Agent could even implement the object with read-only data model definition and just the Alias and Role parameters, with a single entry per Role; this could be sufficient in a case where the Role names convey enough information (e.g., there are only two pre-defined Roles named “Untrusted” and “FullAccess”).

An even simpler Agent might not implement ControllerTrust.Role.{i}. at all, if the Roles are sufficiently intuitive for users.

#### Special Roles

Two special Roles are identified by the UntrustedRole and BannedRole parameters under the ControllerTrust. object. An Agent can expose these parameters with read-only data model implementation if it simply wants to tell Controllers the names of these specific Roles.

The UntrustedRole is the Role the Agent will automatically assign to any Controller that has not been authorized for a different Role. Any Agent that has a means of allowing a Controller’s Role to be changed (by users through a Challenge string, by other Controllers through modification of Controller.{i}.AssignedRole, or through some other external means) and that allows “unknown” Controllers to attach will need to have an “untrusted” Role defined; even if the identity of this Role is not exposed to Controllers through implementation of the UntrustedRole parameter.

The BannedRole (if implemented) is assigned automatically by the Agent to Controllers whose certificates are invalid or have been revoked. If it is not implemented, the Agent can use the UntrustedRole for this, as well. It is also possible to simply implement policy for treatment of invalid or revoked certificates (e.g., refuse to connect), rather than associate them with a specific Role. This is left to the Agent policy implementation.

If the ControllerTrust.Role.{i}. object is implemented, it is good practice to include the UntrustedRole and BannedRole (if implemented) values in that table.

#### A Controller’s Role

A Controller’s assigned Roles can be conveyed by the Controller.{i}.AssignedRole parameter. This parameter is a list of all Role values assigned to the Controller through means other than ControllerTrust.Credential.{i}.Role. A Controller’s inherited Roles (those inherited from ControllerTrust.Credential.{i}.Role as described in the next section) need to be maintained separately from assigned Roles and can be conveyed by the Controller.{i}.InheritedRole parameter. Where multiple assigned and inherited Roles have overlapping Target entries, the resulting permission is the union of all assigned and inherited permissions. For example, if two Roles have the same Target with one Role assigning ParameterPermissions a value of r--- and the other Role assigning ParameterPermissions a value of ---n, the resulting permission will be r--n. This is done after determining which ControllerTrust.Role.{i}. entry to apply for each Role for specific Targets, in the case where a Role has overlapping Target entries for the same Role.

For example, Given the following ControllerTrust.Role.{i}. entries:

i=1, Role = “A”, Targets = “Device.LocalAgent.”, Order = 1, ParameterPermissions = “r---“  
 i=2, Role = “A”, Targets = “Device.LocalAgent.Controller.”, Order = 101, ParameterPermissions = “r-xn“  
 i=3, Role = “B”, Targets = “Device.LocalAgent.”, Order = 1, ParameterPermissions = “r---“  
 i=4, Role = “B”, Targets = “Device.LocalAgent.Controller.”, Order = 101, ParameterPermissions = “rw-n“

and Device.LocalAgent.Controller.1.AssignedRole = “A, B”

When determining permissions for the Device.LocalAgent.Controller. table, the Agent will first determine that for A, both i=1 and i=2 apply to the table, and i=2 takes precedence over i=1 (101 > 1). For B, both i=3 and i=4 apply to the table and i=4 takes precedence over i=3 (101 > 1). The union of i=2 and i=4 is “r-xn” + “rw-n” = “rwxn”.

It is strongly recommended that Agents implement the Controller object with the AssignedRole parameter (with at least read-only data model definition) and InheritedRole parameter (if allowed Roles can come from a trusted CA), so users can see what Controllers have access to the Agent and their permissions. This will help users identify rogue Controllers that may have gained access to the Agent.

#### Role Associated with a Credential or Challenge

The ControllerTrust.Credential.{i}.InheritedRole parameter value is inherited by Controllers whose credentials have been validated using the credentials in the same entry of the ControllerTrust.Credential.{i}. table. Whenever ControllerTrust.Credential.{i}. is used to validate a certificate, the Agent writes the current value of the associated ControllerTrust.Credential.{i}.Role into the Controller.{i}.InheritedRole parameter. For more information on use of this table for assigning Controller Roles and validating credentials, see the sections below.

The ControllerTrust.Challenge.{i}.Role parameter is a default Role that is assigned to Controllers that send a successful ChallengeResponse() command. For more information on use of challenges for assigning Controller Roles, see the sections below.

### Assigning Controller Roles

As mentioned above, the Controller.{i}.AssignedRole parameter can be used to expose the Controller’s assigned Role via the data model.

*Note: Even if it is not exposed through the data model, the Agent is expected to maintain knowledge of the permissions assigned to each known Controller.*

Controllers can be assigned Roles through a variety of methods, depending on the data model elements an Agent implements and the Agent’s coded policy. Note that it is possible for an Agent to maintain trusted credentials with associated permissions (as described by the ControllerTrust.Credential.{i}. object) and various default permission definitions (as identified by the UntrustedRole and BannedRole parameters) without exposing these through the data model. If the data is maintained but not exposed, the same methods can still be used.

* Another Controller (with appropriate permission) can insert a Controller (including the AssignedRole parameter value) into the Controller.{i}. table, or can modify the AssignedRole parameter of an existing Controller.{i}. entry. The InheritedRole value cannot be modified by another Controller.
* If credentials in an entry in a ControllerTrust.Credential.{i}.Credential parameter with an associated ControllerTrust.Credential.{i}.Role parameter are used to successfully validate the certificate presented by the Controller, the Controller inherits the Role from the associated ControllerTrust.Credential.{i}.Role. The Agent writes this value to the Controller.{i}.InheritedRole parameter.
* A Controller whose associated certificate is revoked or invalidated by a CA will be assigned the role in BannedRole, if this parameter or policy is implemented. In this case, the value of BannedRole must be the only value in Controller.{i}.AssignedRole (all other entries are removed) and Controller.{i}.InheritedRole must be empty (all entries are removed).

In the case of a Controller that has not previously been assigned a Role or who has been assigned the value of UntrustedRole:

* If the Controller’s certificate is validated by credentials in a ControllerTrust.Credential.{i}.Credential parameter but there is no associated ControllerTrust.Credential.{i}.Role parameter (or the value is empty), then the Controller is assigned the role in UntrustedRole (written to the Controller.{i}.AssignedRole parameter). Note that assigning UntrustedRole means there needs to be some implemented way to elevate the Controller’s Role, either by another Controller manipulating the Role, implementing Challenges, or some non-USP method.
* If the Controller’s certificate is self-signed or is validated by credentials not in ControllerTrust.Credential.{i}., the Agent policy may be to assign the role in UntrustedRole. This policy can be influenced by the MTP.{i}.<MTP>.ValidatePeerCertificate parameter, if implemented, as described in the next section.
* If the Agent implements the RequestChallenge() and ChallengeResponse() commands, a Controller assigned the role in UntrustedRole can have permission to read one or more ControllerTrust.Challenge.{i}.Alias and Description values and issue the commands. Roles with more extensive permissions can have permission to read additional ControllerTrust.Challenge.{i}.Alias and Description values. A successful Challenge results in the Controller being assigned the associated Role value.

### Controller Certificates and Certificate Validation

When an Agent is presented with a Controller’s certificate, the Agent will always attempt to validate the certificate to whatever extent possible. The scenarios below describe how data model elements can be used to drive the policy for handling various successful and unsuccessful attempts at validation.

Note that it is possible for an Agent to maintain policy of the type described by the UntrustedRole, BannedRole, and MTP.{i}.<MTP>.ValidatePeerCertificate parameters, and the information described by ControllerTrust.Credential.{i}. and Controller.{i}.Credential without exposing these through the data model. If the policy concepts and data are maintained but not exposed, the same methods can still be used. It is also possible for an Agent to have policy that is not described by any defined data model element.

1. If the certificate presented by the Controller is self-signed then:
   1. If the certificate Endpoint ID is in subjectaltName but is not in Controller.{i}.EndpointID, and MTP.{i}.<MTP>.ValidatePeerCertificate is false, the Agent creates a Controller.{i}. entry and assigns Controller.{i}.AssignedRole the role in UntrustedRole. The Agent stores the certificate information in Controller.{i}.Credential.
   2. If the certificate is not in any Controller.{i}.Credential entry and either does not include the Controller Endpoint ID or MTP.{i}.<MTP>.ValidatePeerCertificate is true, the Agent refuses to establish an encrypted connection with the Controller and does not store the certificate information.
   3. If the certificate Endpoint ID is in subjectaltName and is in Controller.{i}.EndpointID with this certificate referenced by that same table entry’s Controller.{i}.Credential, the Agent considers the certificate valid for purpose of confirming Controller identity, and allows the Controller use of its Controller.{i}.AssignedRole.
   4. If the certificate Endpoint ID is in subjectaltName and is in Controller.{i}.EndpointID but this certificate is not referenced by the same table entry’s Controller.{i}.Credential, the Agent considers the certificate invalid and will not establish an encrypted connection.
2. If the certificate indicates it has a chain of trust leading to a Certificate Authority (CA), and the CA indicates the certificate is not valid or has been revoked:
   1. If there is no Controller.{i}. entry for the Endpoint in the USP Header, the Agent may create an entry and assign Controller.{i}.AssignedRole the role in BannedRole.
   2. If there is a Controller.{i}. entry for the Endpoint ID in the USP Header, the Agent will make no changes to the stored information related to that Endpoint ID, and will refuse to establish an encrypted connection.
3. If the certificate indicates it has a chain of trust leading to a Certificate Authority (CA) and the CA indicates the certificate is valid, but the CA is not in ControllerTrust.Credential.{i}., the certificate is treated like a self-signed certificate.
4. If the certificate indicates it has a chain of trust, but the CA is unreachable or the Agent does not attempt to validate the certificate (e.g., no Internet access or CA not responding for some reason or Agent policy is not to validate every time):
   1. If the Controller certificate is in Controller.{i}.Credential and includes the correct Controller Endpoint ID and MTP.{i}.<MTP>.ValidatePeerCertificate is false, the Agent will consider the certificate valid for purpose of confirming Controller identity, and allow the Controller use of Controller.{i}.AssignedRole and Controller.{i}.InheritedRole.
   2. If the Controller certificate is in Controller.{i}.Credential and either does not include the correct Controller Endpoint ID or MTP.{i}.<MTP>.ValidatePeerCertificate is true, the Agent refuses to establish an encrypted connection with the Controller.
   3. If the Controller certificate is not in Controller.{i}.Credential, the Agent will treat it like a self-signed certificate.
5. If the certificate has a chain of trust, the CA indicates the certificate is valid, the CA is in ControllerTrust.Credential.{i}., the CA is reachable, and the certificate includes the Controller Endpoint ID:
   1. If the Controller (for Controller.{i}.EndpointID) has a Controller.{i}.AssignedRole, the Agent considers the certificate valid for purpose of confirming Controller identity (even if the certificate is not the same as Controller.{i}.Credential), and allows the Controller use of its Controller.{i}.AssignedRole. This includes the case where Controller.{i}.AssignedRole is the role in BannedRole.
   2. If the CA has a non-empty ControllerTrust.Credential.{i}.Role, the Agent replaces the current Controller.{i}.InheritedRole (for Controller.{i}.EndpointID) with this value.
   3. If the Controller has no Controller.{i}. entry (for Controller.{i}.EndpointID) or empty Controller.{i}.AssignedRole, and there is no ControllerTrust.Credential.{i}.Role, the Agent assigns the Controller.{i}.AssignedRole the role listed in UntrustedRole.
6. If the certificate has a chain of trust, the CA indicates the certificate is valid, the CA is in ControllerTrust.Credential.{i}., the CA is reachable, and the certificate does not include the exact Controller Endpoint ID, but does include the Controller domain, with or without wildcard and/or includes a wildcarded Controller Endpoint ID:
   1. If the CA has a non-empty ControllerTrust.Credential.{i}.Role, the Agent applies the ControllerTrust.Credential.{i}.Role to the Controller.{i}.InheritedRole. in either the existing entry with Controller.{i}.EndpointID same as the USP Header from\_id or in a new Controller.{i}. entry created for the from\_id (if there was no existing entry). Any Controller.{i}.AssignedRole associated with the Controller Endpoint ID is ignored, because the identity of the Controller has not been validated.
   2. If the ControllerTrust.Credential.{i}.Role is empty, the Agent will not establish an encrypted connection.

### Encryption

It is recommended that Agents implement the ability to encrypt all MTPs (via TLS 1.2 or DTLS 1.2 insert references, as appropriate), enable it by default, and not implement the ability to disable it.

### Challenges

An Agent can implement the ability to provide Controllers with challenges via USP, in order to be trusted with certain Roles. It is also possible to use non-USP methods to issue challenges, such as HTTP digest authentication with prompts for login and password.

To use the USP mechanism, the RequestChallenge() and ChallengeResponse() commands and ControllerTrust.Challenge.{i}. object with at least the Alias and Description parameters must be implemented. The other functionality implied by the other ControllerTrust.Challenge.{i}. needs to be implemented, but does not have to be exposed through the data model.

A Controller that sends a Get message on Device.ControllerTrust.Challenge.{i}. will receive all entries and parameters that are allowed for its current assigned Role. In the simplest case, this will be a single entry and only Alias and Description will be supplied for that entry. It is important to restrict visibility to all other implemented parameters to highly trusted Roles.

The Controller can display the value of Description to the user and allow the user to indicate they want to request the described challenge. If multiple entries were returned, the user can be asked to select which challenge they want to request, based on the description. An example of a description might be “Request administrative privileges” or “Request guest privilege”.

When the user indicates to the Controller which challenge they want, the Controller sends RequestChallenge() with the Alias associated with the selected Description. The Agent replies with the associated Instruction, InstructionType, ValueType and an auto-generated ChallengeID. The Controller presents the value of Instruction to the user (in a manner appropriate for InstructionType). Examples of an instruction might be “Enter passphrase printed on bottom of device” or “Enter PIN sent to registered email address”. The user enters a string per the instructions, and the Controller sends this value together with the ChallengeID in ChallengeResponse().

If the returned value matches Value, the Agent gives a successful response - otherwise it returns an unsuccessful response. If successful, the ControllerTrust.Challenge.{i}.Role replaces an UntrustedRole in Controller.{i}.AssignedRole or is appended to any other Controller.{i}.AssignedRole value.

The number of times a ControllerTrust.Challenge.{i}. entry can be consecutively failed (across all Controllers, without intermediate success) is defined by Retries. Once the number of failed consecutive attempts equals Retries, the ControllerTrust.Challenge.{i}. cannot be retried until after LockoutPeriod has expired.

Type values other than Passphrase can be used and defined to trigger custom mechanisms, such as requests for emailed or SMS-provided PINs.

[<-- Messages](/specification/messages/) [Extensions -->](/specification/extensions/)