[MS-DTAG]:
Device Trust Agreement Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	First Release.
12/18/2009	0.1.1	Editorial	Revised and edited the technical content.
01/29/2010	0.2	Minor	Updated the technical content.

Contents

1 Introduction

This document specifies the Device Trust Agreement Protocol, which is henceforth referred to as "DTAG".

DTAG enables two UPnP endpoints to securely exchange certificates over an unsecure network and to establish a trust relationship by means of a simple, one-time shared secret.

DTAG is compliant with UPnP architecture and is implemented as a UPnP service [UPNPARCH1]. Therefore, this protocol does not have a specific WSDL declaration.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

certificate
endpoint
Hash-based Message Authentication Code (HMAC)
SHA-1 hash
SOAP
SOAP action
SOAP body
SOAP fault
SOAP message
Universal Plug and Play (UPnP)
universally unique identifier (UUID)
UTF-8
Web Services Description Language (WSDL)
XML
XML namespace
XML schema (XSD)

The following terms are specific to this document:

action: A command exposed by a **service** which takes one or more input or output arguments and which may have a return value. For more information, see [UPNPARCH1.1] sections 2 and 3.

authenticator: A large value (160 bits), which is generated from the **payload**, a shared secret, and a **nonce**; and which 1) reveals nothing of the **payload**, shared secret, or **nonce**; and 2) is impractical to generate from any other **payload**, shared secret, or **nonce**.

control point: A control point retrieves device and service descriptions, sends actions to **services**, polls for **service** state variables, and receives events from services.

device: A logical device and/or a container that may embed other logical devices and that embeds one or more services and advertises its presence on network(s). For more information, see [UPNPARCH1.1] sections 1 and 2.

nonce: A unique number for each transaction, typically implemented using a large cryptographically strong random value (160 bits), which is impractical to guess.

one-time password (OTP): A simple secret shared by two **endpoints** and delivered out-of-band by some means outside of the protocol (typically, via user input).

payload: A body of data (such as the **endpoint** ID and the **certificate**) that is to be delivered in a trustworthy fashion.

service: A logical functional unit that represents the smallest units of control and that exposes actions and models the state of a physical device with state variables. For more information, see [UPNPARCH1.1] section 3.

service description: A formal definition of a logical **service**, expressed in the **UPnP** Template language and written in **XML** syntax. A **service description** is specified by a **UPnP** vendor by filling in any placeholders in a **UPnP** Service Template (was SCPD). For more information, see [UPNPARCH1.1] section 2.

service type: Denoted by "urn:schemas-upnp-org:service:" followed by a unique name assigned by a **UPnP** forum working committee, a colon, and an integer version number. A **service type** has a one-to-one relationship with **UPnP** Service Templates. **UPnP** vendors may specify additional **services**; these are denoted by "urn:domain-name:service:" followed by a unique name assigned by the vendor, a colon, and a version number, where domain-name is a Vendor Domain Name. For more information, see [UPNPARCH1.1] section 2.

state variable: A single facet of a model of a physical **service** that is exposed by a **service** and which has a name, data type, optional default value, optional constraints values, and which may trigger events when its value changes. For more information, see [UPNPARCH1.1] sections 2 and 3.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", STD 63, RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[SOAP1.1] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D., "Simple Object Access Protocol (SOAP) 1.1", May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[UPNPARCH1] UPnP Forum, "UPnP Device Architecture 1.0", October 2008, http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[UPNPARCH1.1] UPnP Forum, "UPnP Device Architecture 1.1", October 2008, http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[WSASB] Gudgin, M., Hadley, M., and Rogers, T., "Web Services Addressing 1.0 - SOAP Binding", W3C Recommendation, May 2006, http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/

[WSDL] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., "Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[XMLNS] World Wide Web Consortium, "Namespaces in XML 1.0 (Second Edition)", August 2006, http://www.w3.org/TR/REC-xml-names/

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSCHEMA2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

1.3 Protocol Overview (Synopsis)

A common method for establishing a trust relationship between one **device** and another unknown device is for the devices to exchange and verify each other's **certificate**. However, if the devices are connected over an unsecure network, the success of this method is challenged by the fact that the exchanged information can be exposed to a third party or could even be tampered with. DTAG is designed to ensure the integrity of the **SOAP message** and to enable the establishment of a trust relationship between networked devices by means of a simple, one-time shared secret. The shared secret, called a **one-time password (OTP)**, is transferred in an out-of-band manner, such as through user interaction.

DTAG is implemented as a **UPnP service** consisting of four **actions** that are performed in the following order:

1. **Exchange**: The two **endpoints** exchange certificates and endpoint identifiers.
2. **Commit**, then **Validate**: The two endpoints perform a series of authentications based on the OTP, the OTP substrings, the endpoint identifiers, and the certificates.
3. **Confirm**: The two endpoints finalize the trust agreement process and store each other's certificate in secure storage.

Each action results in a pair of **SOAP** request and response messages in the network, as specified in [UPNPARCH1.1] section 3.1.1. The following diagram illustrates the flow of DTAG messages between the devices and **control points** until the trust agreement is established successfully.

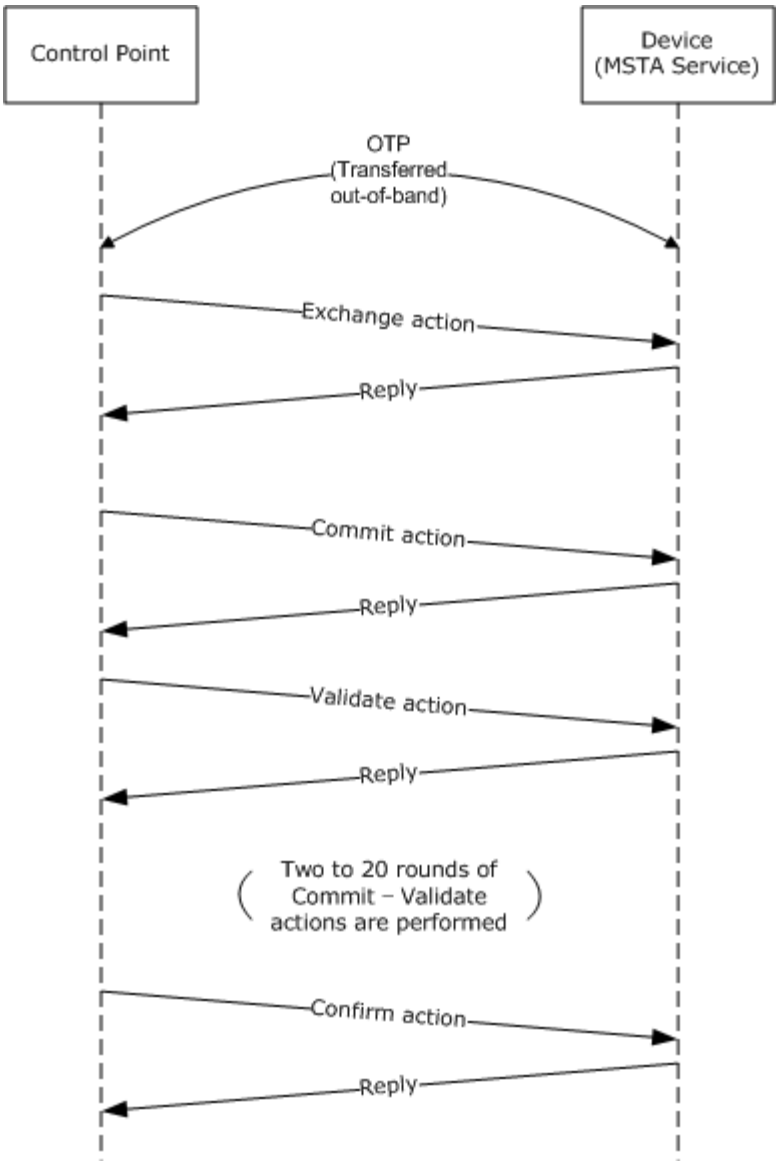

Figure 1: DTAG message sequence to establish trust agreement

1.4 Relationship to Other Protocols

DTAG is a UPnP service over SOAP/HTTP as shown in the following diagram:

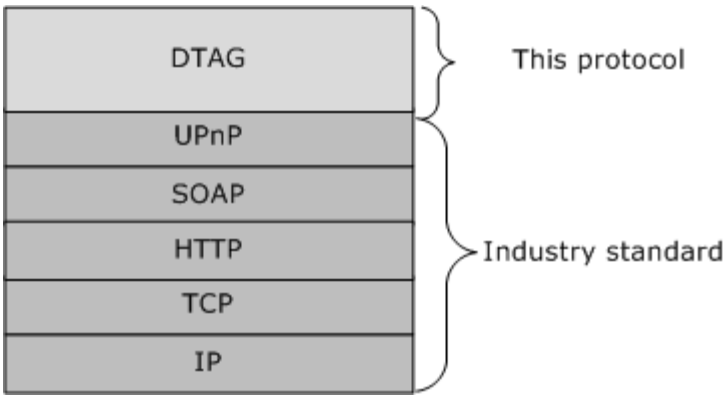

Figure 2: Relationship of DTAG to other protocols

DTAG is built on UPnP architecture version 1.0 [UPNPARCH1] and version 1.1 [UPNPARCH1.1]. For the purposes of this specification, descriptions of the **XML** and SOAP schema are provided via references to UPnP architecture version 1.1 [UPNPARCH1.1].

1.5 Prerequisites/Preconditions

DTAG requires support for storing trusted certificates in a tamper-proof manner.

DTAG requires the support of a UPnP stack on device and control point. The device is required to have the **service description** for DTAG. The full UPnP service description of DTAG is provided in section 8. The device description is also required to include the information about the DTAG service, for which the **service type** is "mstrustagreement", the service identifier is "MSTA", and the version number is as specified in section 1.7.

Before DTAG can be used, all of the necessary, initial UPnP operations are required to be completed, including discovery of devices and publication of service/device descriptions as specified in [UPNPARCH1.1].

1.6 Applicability Statement

Use of DTAG is suitable when the UPnP device and control point are required to ensure the secure exchange of certificates over an unsecure network where the messages can be exposed to a third party or even tampered with.

1.7 Versioning and Capability Negotiation

This document specifies DTAG version 1. The version number is recommended to be included where DTAG service information is presented in a device description, as specified in [UPNPARCH1.1] section 2.3.

This protocol does not have a specific **WSDL** declaration.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

DTAG is implemented as a UPnP **service** and does not specify any transport details beyond what is specified by [UPNPARCH1] section 3.

2.2 Common Message Syntax

This section contains common definitions used by this protocol. The syntax of the definitions uses **XML schema** as defined in [XMLSCHEMA1] and [XMLSCHEMA2], and Web Services Description Language (WSDL) as defined in [WSDL].

2.2.1 Namespaces

This specification defines and references various **XML namespaces** using the mechanisms specified in [XMLNS]. Although this specification associates a specific XML namespace prefix for each XML namespace that is used, the choice of any particular XML namespace prefix is implementation-specific and not significant for interoperability.

Prefix	Namespace URI	Reference
s, SOAP-ENV	http://schemas.xmlsoap.org/soap/envelope/	[SOAP1.1]
m	urn:schemas-microsoft-com:service:mstrustagreement:1	
dt	urn:schemas-microsoft-com:datatypes	

2.2.2 Messages

The following table summarizes the set of common SOAP message definitions defined by this specification. SOAP message definitions that are specific to a particular operation are described with the operation.

Message	Description
UPnP Error	Sends a UPnP error message using a SOAP 1.1 UPnP profile, as specified in [UPNPARCH1.1] section 3.1.

2.2.2.1 UPnP Error

DTAG error messages MUST be expressed in XML using a SOAP 1.1 UPnP profile, as specified in [UPNPARCH1.1] section 3.1. For the purpose of this specification, this section specifies the **SOAP fault** message that is used to support UPnP error reporting.

All SOAP faults defined in this specification MUST be sent as described in [WSASB] section 6. For compatible UPnP error reporting, the values of the SOAP fault elements MUST be set as follows.

SOAP Fault Element	Value
<faultcode>	s:Client
<faultstring>	UPnPError

SOAP Fault Element	Value
<detail>	<UPnPError> element (section 2.2.3.1)

2.2.3 Elements

The following table summarizes the set of common XML schema element definitions defined by this specification. XML schema element definitions that are specific to a particular operation are described with the operation.

Element	Description
<UPnPError>	A wrapper used to support the UPnP error reporting format.
<HostID>	The unique identifier of the control point.
<Iteration>	The iteration number of the current **Validate** action.
<IterationsRequired>	The number of rounds of **Validate** actions.

2.2.3.1 UPnPError

DTAG error messages MUST be expressed in XML using a SOAP 1.1 UPnP profile, as specified in [UPNPARCH1.1] section 3.1. For this expression, the <UPnPError> element can be defined as follows and included as part of the <detail> element of the SOAP fault message, as specified in section 2.2.2.1.

```
<xs:element name="UPnPError">
    <xs:complexType>
        <xs:sequence>
            <xs: element name="ErrorCode" type="xs:integer"/>
            <xs: element name="ErrorDescription" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

The following table lists the possible values of the <ErrorCode> and <ErrorDescription> elements. If an action results in multiple errors, the most specific error MUST be returned.

ErrorCode	ErrorDescription	Explanation
401	Invalid Action	See the description of Control in [UPNPARCH1.1] section 3.
402	Invalid Args	Parameters are missing, extra, or are invalid for this action. See the description of Control in [UPNPARCH1.1] section 3.
403	Out of Sync	See the description of Control in [UPNPARCH1.1] section 3.
501	Action Failed	The service was not able to process this action, or the action is not allowed in the current state. See the description of Control in [UPNPARCH1.1] section 3.
801	Invalid Endpoint	The parameter, <HostID>, has an invalid format or is inconsistent with previous usage.

ErrorCode	ErrorDescription	Explanation
802	Invalid Certificate	The parameter, <HostCertificate>, has an invalid format or does not reference the <HostID>.
803	Invalid Nonce	The authentication process failed.

2.2.3.2 HostID

This element contains the **state variable** **_HostID**, described in section 3.1.1, which is the unique identifier information specific to the control point.

```
<xs:element name="HostID" type="A_ARG_TYPE_EndpointID"/>
```

2.2.3.3 Iteration

This element contains the state variable **Iter**, described in section 3.1.1, indicating the iteration number of the current **Validate** action.

```
<xs:element name="Iteration" type="A_ARG_TYPE_Iteration"/>
```

2.2.3.4 IterationsRequired

This element contains the state variable **N**, described in section 3.1.1, which is used to negotiate the number of rounds of **Validate** actions to complete the trust agreement process.

```
<xs:element name="IterationsRequired" type="A_ARG_TYPE_Rounds"/>
```

2.2.4 Complex Types

This specification does not define any common XML schema complex type definitions.

2.2.5 Simple Types

The following table summarizes the set of common XML schema simple type definitions defined by this specification. XML schema simple type definitions that are specific to a particular operation are described with the operation.

Simple type	Description
<A_ARG_TYPE_Rounds>	The number of rounds required for the **Validate** action.
<A_ARG_TYPE_Iteration>	The iteration number for the current **Validate** action.
<A_ARG_TYPE_EndpointID>	The **UUID** of the device (or the control point).
<A_ARG_TYPE_Authenticator>	A 20-octet authentication code.
<A_ARG_TYPE_Nonce>	An array of 20 octets (for a total of 160 bits) that contains

Simple type	Description
	cryptographically strong random values.
<A_ARG_TYPE_Certificate>	The certificate of the device (or the control point).

2.2.5.1 A_ARG_TYPE_Rounds

This type of element is used to negotiate the number of rounds required for **Validate** actions to be performed by the protocol.

```
<xs:simpleType name="A_ARG_TYPE_Rounds" >
    <xs:restriction base="xs:unsignedByte" >
        <xs:minInclusive value="2"/>
        <xs:maxInclusive value="20"/>
    </xs:restriction>
</xs:simpleType>
```

The number of rounds MUST be in the range of 2 to 20, inclusive.

2.2.5.2 A_ARG_TYPE_Iteration

This type of element is limited to the values between 1 and 20. These values correspond to each of the **N** times that the **Commit** and **Validate** actions are called.

```
<xs:simpleType name="A_ARG_TYPE_Iteration" >
    <xs:restriction base="xs:unsignedByte" >
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="20"/>
    </xs:restriction>
</xs:simpleType>
```

2.2.5.3 A_ARG_TYPE_EndpointID

This type of element is a string that uniquely identifies an endpoint. It has to remain stable for the lifetime of the device.

```
<xs:simpleType name="A_ARG_TYPE_EndpointID" >
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

2.2.5.4 A_ARG_TYPE_Authenticator

This type of element is the 160-bit (20-octet) result of the **HMAC-SHA-1** message authentication code [RFC2104], as specified in section 3.1.1, encoded as a Base64 string.

```
<xs:simpleType name="A_ARG_TYPE_Authenticator" >
    <xs:restriction base="xs:string"/>
```

```
</xs:simpleType>
```

2.2.5.5 A_ARG_TYPE_Nonce

This type of element is an array of 20 octets (160 bits) that contains cryptographically-strong random values, encoded as a Base64 string.

```
<xs:simpleType name="A_ARG_TYPE_Nonce" >
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

2.2.5.6 A_ARG_TYPE_Certificate

This type of element is a string and contains a certificate encoded as a Base64 string.

```
<xs:simpleType name="A_ARG_TYPE_Certificate" >
    <xs:restriction base="xs:string"/>
</xs:simpleType>
```

2.2.6 Attributes

This specification does not define any common XML schema attribute definitions.

2.2.7 Groups

This specification does not define any common XML schema group definitions.

2.2.8 Attribute Groups

This specification does not define any common XML schema attribute group definitions.

3 Protocol Details

The operations of the device and control point are almost symmetric because they examine each other using the same types of information.

3.1 Common Details

This section describes protocol details that are common between the device and control point.

The device and control point start the trust agreement process when a one-time password (OTP) is made available to the two endpoints. Throughout the trust agreement process, the device and control point MUST synchronize the state to perform each action.

The following diagram provides an overview of the state machine common to the device and control point.

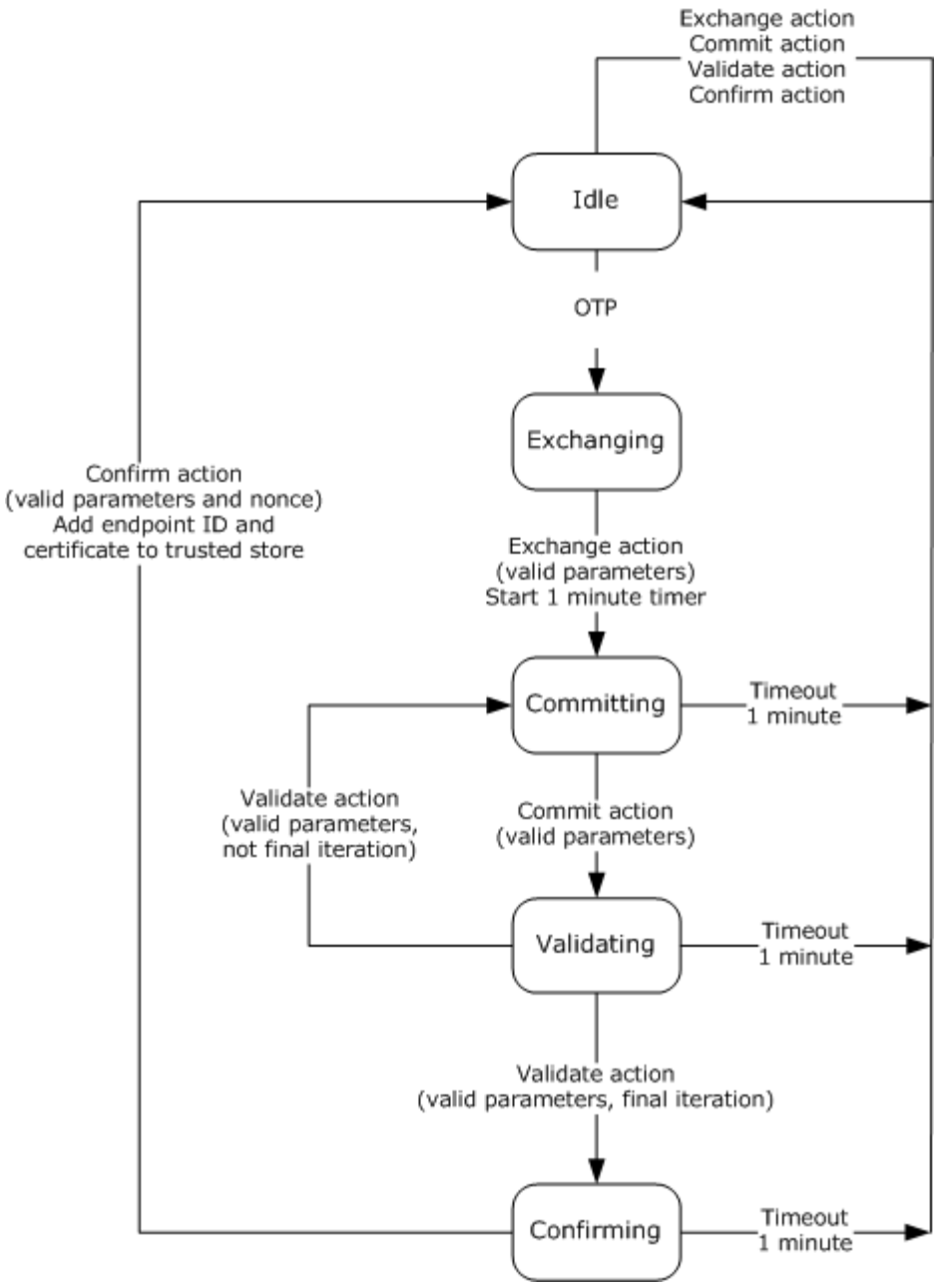

Figure 3: DTAG message sequence to establish trust agreement

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

TrustState: The current setting of the service's state machine. The following states are specified for this state variable.

TrustState	State	Description
0	*Idle*	The trust agreement process is not started. The device and control point wait for a one-time password (OTP) event.
1	*Exchanging*	The device and control point exchange certificates and endpoint identifiers, along with the authentication code based on the entire **OTP** string. This authentication code will be examined in the last *Confirming* state. The **Exchange** action is processed in this state.
2	*Committing*	The device and control point exchange the authentication code based on **OTP** substrings. The **Commit** action and timeout event are processed in this state.
3	*Validating*	The device and control point validate the authentication code exchanged on the previous **Commit** action. The **Validate** action and timeout event are processed in this state.
4	*Confirming*	The device and control point finalize the validation of the authentication code obtained in the **Exchange** action. The **Confirm** action and timeout event are processed in this state.

N: The number of rounds required for the **Commit**-**Validate** actions that will be perfomed by the protocol. The value of this state variable is selected at run-time.

Iter: The current iteration number at which **Commit**-**Validate** actions are performed. This state variable is only valid up to **N**.

OTP: The one-time password (OTP).

$\mathbf{OTP_{Iter}}$: The substring **OTP** for the indicated iteration.

The **OTP** and its substrings are obtained by the following rule.

The **OTP** is divided up into **N** substrings. These substrings are denoted as OTP_1, OTP_2, ... OTP_n. The rule for generating the substring **OTPs** from the **OTP** is as follows:

- Individual characters in an **OTP** are not broken up.
- The number of characters in the **OTP** MUST be greater than or equal to the number of rounds specified in the state variable **N**.
- If *L* is the number of characters in the **OTP**, then each substring will be either *L div* **N** or *L div* **N**+1 characters long. The last *L mod* **N** substrings will have *L div* **N**+1 characters. All of the other substrings will have *L div* **N** characters.
- The characters of the **OTP** are broken up in order into their substrings.

For example, if the value of **N** is 4 and the value of the **OTP** is "ThatCat", then the first substring, OTP1 would be "T", the second, OTP2 would be "ha", the third, OTP3 would be "tC", and the fourth, OTP4 would be "at".

_DeviceCertificate: The certificate of the device that is associated with the **_DeviceID** state variable and which MUST remain stable for the lifetime of the device.

_DeviceConfirmAuthenticator: The authentication code made by the device for the **Exchange** and **Confirm** actions.

_DeviceConfirmNonce: A 20-octet **nonce** made by the device for the **Exchange** and **Confirm** actions.

_DeviceID: The UUID of the device.

_DeviceValidateAuthenticatorIter: The authentication code of the device for the indicated iteration of **Commit**-**Validate** actions.

_DeviceValidateNonceIter: A 20-octet nonce of the device for the indicated iteration of **Commit**-**Validate** actions.

_HostCertificate: The certificate of the control point that is associated with the **_HostID** state variable and which MUST remain stable for the lifetime of the control point.

_HostConfirmAuthenticator: The authentication code of the control point for the **Exchange** and **Confirm** actions.

_HostConfirmNonce: A 20-octet nonce of the control point for the **Exchange** and **Confirm** actions.

_HostID: The UUID of the control point.

_HostValidateAuthenticatorIter: The authentication code of the control point for the indicated iteration of the **Commit**-**Validate** action.

_HostValidateNonceIter: A 20-octet nonce of the control point for the indicated iteration of the **Commit**-**Validate** action.

The **_DeviceValidateAuthenticatorIter**, **_DeviceConfirmAuthenticator**, **_HostValidateAuthenticatorIter**, and **_HostConfirmAuthenticator** are the 160-bit (20-octet) result of the HMAC-SHA-1 message authentication code [RFC2104]. The HMAC-SHA-1 function takes two parameters, a 20-octet key and some variable-length text, and returns a 20-octet message authentication code.

The HMAC-SHA-1 function key is a nonce.

The HMAC-SHA-1 function text is the **UTF-8** representation [RFC3629] of the concatenation of the following items in the order presented:

- **N** (or **Iter**) , encoded as a Base64 string
- An **OTP** string (or $\mathbf{OTP}_{\mathbf{Iter}}$ substring)
- The endpoint identifier
- A certificate, encoded as a Base64 string

Therefore, the HMAC-SHA-1 results are denoted in this specification as:

_DeviceConfirmAuthenticator

= **HMAC**(**_DeviceConfirmNonce**, UTF-8(**N** + **OTP** + **_DeviceID** + **_DeviceCertificate**)

_HostConfirmAuthenticator

= HMAC(**_HostConfirmNonce**, UTF-8(**N** + **OTP** + **_HostID** + **_HostCertificate**)

_DeviceValidateAuthenticator$_{\textbf{Iter}}$

= HMAC(**_DeviceValidateNonce**$_{\textbf{Iter}}$, UTF-8(**Iter**$_{\textbf{Iter}}$ + **OTP**$_{\textbf{Iter}}$ + **_DeviceID** + **_DeviceCertificate**)

_HostValidateAuthenticator$_{\textbf{Iter}}$

= HMAC(**_HostValidateNonce**$_{\textbf{Iter}}$, UTF-8(**Iter**$_{\textbf{Iter}}$ + **OTP**$_{\textbf{Iter}}$ + **_HostID** + **_HostCertificate**)

3.1.2 Timers

None.

3.1.3 Initialization

Before startup, the device and control point keep the **TrustState** state variable set to 0 (*Idle*). In this state, any of the service's actions MUST NOT be called, and invoking any one of them MUST return an error.

3.1.4 Message Processing Events and Sequencing Rules

None.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

3.1.6.1 One-time Password (OTP) Event

An **OTP** event outside of the scope of this specification (for example, human interaction) triggers the start of the trust agreement process. When the trigger event occurs, the service MUST initiate the process as follows:

1. The device and control point MUST terminate any ongoing DTAG process, discarding all locally saved **OTPs**, nonces, endpoint identifiers, and certificates.
2. The device and control point MUST acquire and locally save the endpoint identifier (in other words, **_DeviceID** and **_HostID**, respectively).
3. The device and control point MUST acquire and locally save the certificate (in other words, **_DeviceCertificate** and **_HostCertificate**, respectively).
4. The device and control point MUST acquire and locally save the **OTP**, and generate the substrings, as described in section 3.1.1.
5. The device and control point MUST change **TrustState** from 0 (*Idle*) to 1 (*Exchanging*), as described in section 3.1.1.

3.2 Device Details

In addition to the protocol details specified in section 3.1, the following details are also applied to the device.

3.2.1 Abstract Data Model

None.

3.2.2 Timers

The device and control point each have a 1 minute timer "TimeOut" for the maximum interval allowed for the transition between actions.

3.2.3 Initialization

None.

3.2.4 Message Processing Events and Sequencing Rules

On each action, the control point sends a request message to the device, and the device returns a response or error message to the control point, as specified in [UPNPARCH1.1] section 3.1.

3.2.4.1 Exchange Action

In order to perform the **Exchange** action, the control point MUST attach an <Exchange> body to the DTAG SOAP message that contains the <HostID>, <HostCertificate>, <IterationsRequired>, and <HostConfirmAuthenticator> elements. This action is supported only when **TrustState** is 1 (*Exchanging*).

On this action, the following checks MUST be performed:

1. The <HostID>, <HostCertificate>, <IterationsRequired>, and <HostConfirmAuthenticator> elements MUST be syntactically validated.
2. The <HostCertificate> (**_HostCertificate**) MAY be validated as per any vendor-defined rules.
3. The <IterationsRequired> (**N**) MAY additionally be checked per vendor-defined rules.

If successful, the device:

1. MUST locally save the values of **_HostID**, **_HostCertificate**, and **_HostConfirmAuthenticator**.
2. MUST generate and locally save **_DeviceConfirmNonce**.
3. MUST change **TrustState** from 1 (*Exchanging*) to 2 (*Committing*).
4. MUST start a one-minute timer.
5. MUST set the following elements and return with status 200 (success):
   - The <DeviceID>, the **UUID** of the enclosing UPnP device, as specified in A_ARG_Type_EndpointID (section 2.2.5.3), and as acquired in section 3.1.6.1.
   - The <DeviceCertificate>, as specified in A_ARG_TYPE_Certificate (section 2.2.5.6), and as acquired in section 3.1.6.1.
   - The <DeviceConfirmAuthenticator>, an HMAC as specified in section 3.1.1, calculated as:

     Base64 (HMAC(**_DeviceConfirmNonce**, UTF-8 (**N** + **OTP** + **_DeviceID** + **_DeviceCertificate**)).

If this action fails, the device MUST send a SOAP fault message to the control point, as specified in section 2.2.2.1.

3.2.4.1.1 Messages

Message	Description
Exchange	Contains the request for the **Exchange** action.
Exchange Response	Contains the response of the **Exchange** action.

3.2.4.1.1.1 Exchange Message

The HTTP header MUST specify the SOAPACTION for the **Exchange** action as follows:

SOAPACTION: "urn:schemas-microsoft-com:service: mstrustagreement:1#Exchange"

Where "urn:schemas-microsoft-com:service: mstrustagreement:1" is the service type that comes from the device description, as specified in section 7, and "#Exchange" is the **SOAP action**.

The following XML session shows the <HostId>, <HostCertificate>, <IterationsRequired>, and <HostConfirmAuthenticator> elements in a SOAP Exchange message.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:Exchange xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <HostID xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
        Control point identifier
      </HostID>
      <HostCertificate xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
        Host Certificate payload
      </HostCertificate>
      <IterationsRequired xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="ui1">
        Number of iterations required
      </IterationsRequired>
      <HostConfirmAuthenticator xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="string">
        HostConfirmAuthenticator payload
      </HostConfirmAuthenticator>
    </m:Exchange>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HostID: The <HostID> element, as specified in section 2.2.3.2.

HostCertificate: The <HostCertificate> element, as specified in section 3.2.4.1.2.2.

IterationsRequired: The <IterationsRequired> element, as specified in section 2.2.3.4.

HostConfirmAuthenticator: The <HostConfirmAuthenticator> element, as specified in section 3.2.4.1.2.4.

3.2.4.1.1.2 Exchange Response Message

The device MUST reply with an ExchangeResponse SOAP response message, which contains the <DeviceID>, <DeviceCertificate>, and <DeviceConfirmAuthenticator> elements.

```
<s:Envelope xmlns:s=http://schemas.xmlsoap.org/soap/envelope/
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ExchangeResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceID>
        Device identifier
      </DeviceID>
      <DeviceCertificate>
        Device Certificate payload
      </DeviceCertificate>
      <DeviceConfirmAuthenticator>
        DeviceConfirmAuthenticator payload
      </DeviceConfirmAuthenticator>
    </u:ExchangeResponse>
  </s:Body>
</s:Envelope>
```

DeviceID: The <DeviceID> element, as specified in section 3.2.4.1.2.1.

DeviceCertificate: The <DeviceCertificate> element, as specified in section 3.2.4.1.2.3.

DeviceConfirmAuthenticator: The <DeviceConfirmAuthenticator> element, as specified in section 3.2.4.1.2.5.

3.2.4.1.2 Elements

The following table summarizes the XML Schema element definitions that are specific to this operation.

Element	Description
<DeviceID>	A string that contains the **_DeviceID**, as described in section 3.1.1.
<HostCertificate>	A Base64 encoded string that contains the **_HostCertificate**, as described in section 3.1.1.
<DeviceCertificate>	A Base64 encoded string that contains the **_DeviceCertificate**, as described in section 3.1.1.
<HostConfirmAuthenticator>	A Base64 encoded string that contains the **_HostConfirmAuthenticator**, as described in section 3.1.1.
<DeviceConfirmAuthenticator>	A Base64 encoded string that contains the **_DeviceConfirmAuthenticator**, as described in section 3.1.1.

3.2.4.1.2.1 DeviceID

This element provides the unique identifier information specific to the device(**_DeviceID**). This element is contained in the **SOAP body** of the ExchangeResponse message.

```
<xs:element name="DeviceID" type="A_ARG_TYPE_EndipointID"/>
```

3.2.4.1.2.2 HostCertificate

This element provides the control point certificate (**_HostCertificate**). This element is contained in the SOAP body of the ExchangeResponse message and is encoded as a Base64 string.

```
<xs:element name="HostCertificate" type="A_ARG_TYPE_Certificate"/>
```

3.2.4.1.2.3 DeviceCertificate

This element provides the certificate of the device associated with the <DeviceID> (**_DeviceCertificate**) encoded as a Base64 string. This element is contained in the SOAP body of the ExchangeResponse message and is encoded as a Base64 string.

```
<xs:element name="HostCertificate" type="A_ARG_TYPE_Certificate"/>
```

3.2.4.1.2.4 HostConfirmAuthenticator

This element provides the 160-bit (20-octet) authentication code for the control point (**_HostConfirmAuthenticator**). This element is contained in the SOAP body of the Exchange message that and is encoded as a Base64 string.

```
<xs:element name="HostConfirmAuthenticator" type="A_ARG_TYPE_Authenticator"/>
```

3.2.4.1.2.5 DeviceConfirmAuthenticator

This element provides the 160-bit (20-octet) authentication code made by the device (**_DeviceConfirmAuthenticator**), specified in section 3.2.4.1. This element is contained in the SOAP body of the ExchangeResponse message and is encoded as a Base64 string.

```
<xs:element name="HostConfirmAuthenticator" type="A_ARG_TYPE_Authenticator"/>
```

3.2.4.2 Commit Action

In order to perform the **Commit** action, the control point MUST attach a <Commit> body to the DTAG SOAP message that contains the <HostID>, <Iteration>, and <HostValidateAuthenticator> elements. This action is only supported when **TrustState** is 2 (*Committing*).

On this action, the following checks MUST be performed:

1. The <HostID>, <Iteration>, and <HostValidateAuthenticator> (**_HostValidateAuthenticator**$_{\text{Iter}}$) elements MUST be syntactically validated.

2. The <HostID> MUST match the value of the **_HostID** obtained in the **Exchange** action.

If successful, the service:

1. MUST change **TrustState** from 2 (*Committing*) to 3 (*Validating*).
2. MUST generate **_DeviceValidateNonce$_{Iter}$**.
3. MUST set the following element and return with status 200 (success).
   - The <DeviceValidateAuthenticator>, an HMAC as specified in section 3.1.1, calculated as:

     Base64(HMAC(**_DeviceValidateNonce$_{Iter}$**, UTF-8(**Iter** + **OTP$_{Iter}$** + **_DeviceID** + **_DeviceCertificate**)).

If this action fails, the device MUST send a SOAP fault message to the control point, as specified in section 2.2.2.1.

3.2.4.2.1 Messages

Message	Description
Commit	Contains the request for the **Commit** action.
Commit Response	Contains the response of the **Commit** action.

3.2.4.2.1.1 Commit Message

The HTTP header MUST specify the SOAPACTION for the **Commit** action as follows:

SOAPACTION: "urn:schemas-microsoft-com:service: mstrustagreement:1#Commit"

Where "urn:schemas-microsoft-com:service: mstrustagreement:1" is the service type which comes from the device description as specified in section 7 and "#Commit" is the SOAP action.

The following XML session shows the <HostId>, <Iteration>, and <HostValidateAuthenticator> elements in a SOAP Commit message.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:Commit xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <HostID xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
        Host identifier
      </HostID>
      <Iteration xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="ui1">
        Current iteration
      </Iteration>
      <HostValidateAuthenticator xmlns:dt="urn:schemas-microsoft-com:datatypes"
dt:dt="string">
        HostValidateAuthenicator payload
      </HostValidateAuthenticator>
    </m:Commit>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HostID: The <HostID> element, as specified in section 2.2.3.2.

Iteration: The <Iteration> element, as specified in section 2.2.3.3.

HostValidateAuthenticator: The <HostValidateAuthenticator> element, as specified in section 3.2.4.2.2.1.

3.2.4.2.1.2 Commit Response Message

The server MUST reply with a CommitResponse SOAP response message that containis the <DeviceValidateAuthenticator> element.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:CommitResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceValidateAuthenticator>
        DeviceValidateAuthenticator payload
      </DeviceValidateAuthenticator>
    </u:CommitResponse>
  </s:Body>
</s:Envelope>
```

DeviceValidateAuthenticator: The <DeviceValidateAuthenticator> element, as specified in section 3.2.4.2.2.2.

3.2.4.2.2 Elements

The following table summarizes the XML Schema element definitions that are specific to this operation.

Element	Description
<HostValidateAuthenticator>	A Base64 encoded string that contains the **_HostValidateAuthenticator$_{Iter}$**, as described in section 3.1.1.
<DeviceValidateAuthenticator>	A Base64 encoded string that contains the **_DeviceValidateAuthenticator$_{Iter}$**, as described in section 3.1.1.

3.2.4.2.2.1 HostValidateAuthenticator

This element provides the 160-bit (20-octet) authentication code for the control point (**_HostValidateAuthenticatorIter**). This element is contained in the SOAP body of the Commit message and is encoded as a Base64 string.

```
<xs:element name="HostValidateAuthenticator" type="A_ARG_TYPE_Authenticator"/>
```

3.2.4.2.2.2 DeviceValidateAuthenticator

This element provides the 160-bit (20-octet) authentication code for the device (**_DeviceValidateAuthenticatorIter**), as specified in section 3.2.4.2. This element is contained in the SOAP body of the CommitResponse message and is encoded as a Base64 string.

```
<xs:element name="DeviceValidateAuthenticator" type="A_ARG_TYPE_Authenticator"/>
```

3.2.4.3 Validate Action

In order to perform the **Validate** action, the control point MUST attach a <Validate> body to the DTAG SOAP message that contains the <HostID>, <Iteration>, and <HostValidateNonce> elements. This action is only valid if **TrustState** is 3 (*Validating*).

On this action, the following checks MUST be performed:

1. The <HostID>, <Iteration>, and <HostValidateNonce> (**_HostValidateNonceIter**) elements MUST be syntactically validated.
2. The <HostID> MUST match the value of the **_HostID** obtained in the **Exchange** action.
3. The <Iteration> number MUST be equal to the device's current iteration number, **Iter**.
4. The value of HMAC(**_HostValidateNonceIter**, UTF-8(**Iter** + **OTP_{Iter}** + **_HostID** + **_HostCertificate**)) calculated as specified in section 3.1.1, MUST match the **$\_HostValidateAuthenticator_{Iter}$** obtained in the **Commit** action.

If successful, the service:

1. MUST increment the iteration number, **Iter**.
2. MUST change **TrustState** from 3 (*Validating*) to 4 (*Confirming*), if this is the last iteration, or to 2 (*Committing*) if this is not the last iteration.
3. MUST set the following element and return with status 200 (success).
   - <DeviceValidateNonce>, a Base64 encoded string of **$\_DeviceValidateNonce_{Iter}$**, which is the 20-octet random number acquired in section 3.2.4.2.

If this action fails, the device MUST send a SOAP fault message to the control point, as specified in section 2.2.2.1.

3.2.4.3.1 Messages

Message	Description
Validate	Contains the request for the **Validate** action.
Validate Response	Contains the response of the **Validate** action.

3.2.4.3.1.1 Validate Message

The HTTP header MUST specify the SOAPACTION for the **Validate** action as follows:

SOAPACTION: "urn:schemas-microsoft-com:service: mstrustagreement:1#Validate"

Where "urn:schemas-microsoft-com:service: mstrustagreement:1" is the service type, which comes from the device description, as specified in section 7 and "#Validate" is the SOAP action.

The following XML session shows the <HostID>, <Iteration>, and <HostValidateNonce> elements in a SOAP Validate message.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
<SOAP-ENV:Body>
  <m:Validate xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
    <HostID xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
      Host identifier
    </HostID>
    <Iteration xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="ui1">
      Current iteration
    </Iteration>
    <HostValidateNonce xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
      HostValidateNonce payload
    </HostValidateNonce>
  </m:Validate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HostID: The <HostID> element, as specified in section 2.2.3.2.

Iteration: The <Iteration> element, as specified in section 2.2.3.3.

HostValidateNonce: The <HostValidateNonce> element, as specified in section 3.2.4.3.2.1.

3.2.4.3.1.2 Validate Response Message

The server MUST reply with a ValidateResponse SOAP response message that contains the <DeviceValidateNonce> element.

```
<s:Envelope.. xmlns:s=http://schemas.xmlsoap.org/soap/envelope/
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ValidateResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceValidateNonce>
        DeviceValidateNonce payload
      </DeviceValidateNonce>
    </u:ValidateResponse>
  </s:Body>
</s:Envelope>
```

DeviceValidateNonce: The <DeviceValidateNonce> element, as specified in section 3.2.4.3.2.2.

3.2.4.3.2 Elements

The following table summarizes the XML Schema element definitions that are specific to this operation.

Element	Description
<HostValidateNonce>	A Base64 encoded string that contains the **_HostValidateNonce$_{Iter}$**, as described in section 3.1.1.
<DeviceValidateNonce>	A Base64 encoded string that contains the **_DeviceValidateNonce$_{Iter}$**, as described in section 3.1.1.

3.2.4.3.2.1 HostValidateNonce

This element provides the **nonce** of the control point for the indicated iteration of the **Commit-Validate** action (**_HostValidateNonce$_{Iter}$**). This element is contained in the SOAP body of the Validate message and is encoded as a Base64 string.

```
<xs:element name="HostValidateNonce" type="A_ARG_TYPE_Nonce"/>
```

3.2.4.3.2.2 DeviceValidateNonce

This element provides the nonce of the device for the indicated iteration of the **Commit-Validate** action (**_DeviceValidateNonce$_{Iter}$**). This element is contained in the SOAP body of the ValidateResponse message and is encoded as a Base64 string.

```
<xs:element name="DeviceValidateNonce" type="A_ARG_TYPE_Nonce"/>
```

3.2.4.4 Confirm Action

In order to perform the **Confirm** action, the control point MUST attach a <Confirm> body to the DTAG SOAP message that contains the <HostID>, <IterationsRequired>, and <HostConfirmNonce> elements. This action is only valid if **TrustState** is 4 (*Confirming*).

On this action, the following checks MUST be performed:

The <HostID>, <HostConfirmNonce>, and <IterationsRequired> (**N**) MUST be syntactically validated.

The <HostID> MUST match the value of the **_HostID** obtained in the **Exchange** action.

The value of HMAC(**_HostConfirmNonce**, UTF-8 (**N** + **OTP** + **_HostID** + **_HostCertificate**)), as specified in section 3.1.1, MUST match the **_HostConfirmAuthenticator** acquired in the **Exchange** action.

If successful, then trust has been established, and the service:

MUST store **_HostID** and **_HostCertificate** in a tamper-proof, persistent store.

MUST change **TrustState** from 4 (*Confirming*) to 0 (*Idle*).

MUST set the following element and return with status 200 (success):

<DeviceConfirmNonce>, a Base64 encoded string of **_DeviceConfirmNonce**, which is the 20 octet random number acquired in section 3.1.6.1.

If this action fails, the device MUST send a SOAP fault message to the control point, as specified in section 3.1.1.

3.2.4.4.1 Messages

Message	Description
Confirm	Contains the request for the **Confirm** action.

Message	Description
Confirm Response	Contains the response of the **Confirm** action.

3.2.4.4.1.1 Confirm Message

The HTTP header MUST specify the SOAPACTION for the **Confirm** action as follows:

SOAPACTION: "urn:schemas-microsoft-com:service: mstrustagreement:1#Confirm"

Where "urn:schemas-microsoft-com:service: mstrustagreement:1" is the service type that comes from the device description, as specified in section 7, and "#Confirm" is the SOAP action.

The following XML session shows the <HostID>, <IterationsRequired>, and <HostConfirmNonce> in a SOAP Confirm message.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:Confirm xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <HostID xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
        Host identifier
      </HostID>
      <IterationsRequired xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="ui1">
        Number of iterations requested
      </IterationsRequired>
      <HostConfirmNonce xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="string">
        HostConfirmationNonce payload
      </HostConfirmNonce>
    </m:Confirm>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

HostID: The <HostID> element, as specified in section 2.2.3.2.

IterationsRequired: The <IterationsRequired> element, as specified in section 2.2.3.4.

HostConfirmNonce: The <HostConfirmNonce> element, as specified in section 3.2.4.4.2.1.

3.2.4.4.1.2 Confirm Response Message

The server MUST reply with a ConfirmResponse SOAP response message that contains the <DeviceConfirmNonce> element.

```
<s:Envelope xmlns:s=http://schemas.xmlsoap.org/soap/envelope/
  ..s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ConfirmResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceConfirmNonce>
        DeviceConfirmNonce payload
      </DeviceConfirmNonce>
    </u:ConfirmResponse>
  </s:Body>
```

```
</s:Envelope>
```

DeviceConfirmNonce: The <DeviceConfirmNonce> element, as specified in section 3.2.4.4.2.2

3.2.4.4.2 Elements

The following table summarizes the XML Schema element definitions that are specific to this operation.

Element	Description
<HostConfirmNonce>	A Base64 encoded string that contains the **_HostConfirmNonce**,as described in section 3.1.1.
<DeviceConfirmNonce>	A Base64 encoded string that contains the **_DeviceConfirmNonce**, as described in section 3.1.1.

3.2.4.4.2.1 HostConfirmNonce

This element provides the nonce of the control point for the indicated iteration of the **Exchange** and **Confirm** actions (**_HostConfirmNonce**). This element is contained in the SOAP body of the Confirm message and is encoded as a Base64 string.

```
<xs:element name="HostConfirmNonce" type="A_ARG_TYPE_Nonce"/>
```

3.2.4.4.2.2 DeviceConfirmNonce

This element provides the nonce of the device for the indicated iteration of the **Exchange** and **Confirm** actions (**_DeviceConfirmNonce**). This element is contained in the SOAP body of the ConfirmResponse message and is encoded as a Base64 string.

```
<xs:element name="DeviceConfirmNonce" type="A_ARG_TYPE_Nonce"/>
```

3.2.5 Timer Events

After sending each response message, if the device does not receive the message of the next action within one minute, the device MUST stop DTAG and reset **TrustState** to 0 (*Idle*).

3.2.6 Other Local Events

None.

3.3 Control Point (Host) Details

In addition to the protocol details specified in section 3.1, the following details are applied to the control point.

3.3.1 Abstract Data Model

None.

3.3.2 Timers

None.

3.3.3 Initialization

None.

3.3.4 Message Processing Events and Sequencing Rules

None.

3.3.4.1 Exchange Response

This response is supported only when **TrustState** is 1 (*Exchanging*). On this response, the following checks MUST be performed:

1. The <DeviceID>, <DeviceCertificate>, and <DeviceConfirmAuthenticator> elements MUST be syntactically validated.
2. The <DeviceCertificate> (**_DeviceCertificate**) MAY be validated as per any vendor-defined rules.

If successful, the control point:

1. MUST locally save the values of **_DeviceID**, **_DeviceCertificate**, and **_DeviceConfirmAuthenticator**.
2. MUST set **Iter** to 1.
3. MUST generate $\textbf{\_HostValidateNonce}_{\textbf{Iter}}$.
4. MUST change **TrustState** from 1 (*Exchanging*) to 2 (*Committing*).
5. MUST set the following elements and send them in a Commit Message (section 3.2.4.2.1.1):
   - <HostID>, as acquired in section 3.1.6.1.
   - <Iteration>, as of the current **Iter** value.
   - <HostValidateAuthenticator>, an HMAC, as specified in section 3.1.1, calculated as:

     Base64(HMAC($\textbf{\_HostValidateNonce}_{\textbf{Iter}}$, UTF-8(**Iter** + $\textbf{OTP}_{\textbf{Iter}}$ + **_HostID** + **_HostCertificate**)).

If this action fails, the control point MUST change **TrustState** to 0 (*Idle*), cancel the DTAG protocol, and report an error to the control point user of this protocol.

3.3.4.2 Commit Response

This response is supported only when **TrustState** is 2 (*Committing*). On this response, the following checks MUST be performed:

- The <DeviceValidateAuthenticator> element MUST be syntactically validated.

If successful, the service:

1. MUST change **TrustState** from 2 (*Committing*) to 3 (*Validating*).
2. MUST set the following element and send them in a Validate Message (section 3.2.4.3.1.1):
   - <HostID>, as acquired in section 3.1.6.1.
   - <Iteration>, as the current **Iter** value.
   - <HostValidateNonce>, for the current **Iter** value.

If this action fails, the control point MUST change **TrustState** to 0 (*Idle*), cancel the DTAG protocol, and report an error to the control point user of this protocol.

3.3.4.3 Validate Response

This action is supported only when **TrustState** is 3 (*Validating*). On this action, the following checks MUST be performed:

1. The <DeviceValidateNonce> (**_DeviceValidateNonce**$_{\textbf{Iter}}$) element MUST be syntactically validated.
2. The <Iteration> number MUST be equal to the device's current iteration number, **Iter**.
3. The value of HMAC(**_DevicetValidateNonce**$_{\textbf{Iter}}$, UTF-8(**Iter** + **OTP**$_{\textbf{Iter}}$ + **_DeviceID** + **_DeviceCertificate**)), calculated as specified in section 3.1.1, MUST match the **_DeviceValidateAuthenticator**$_{\textbf{Iter}}$ obtained in the **Commit** response.

If successful, and this is *not* the last iteration, the service:

1. MUST increment the iteration number, **Iter**.
2. MUST change **TrustState** from 3 (*Validating*) to 2 (*Committing*).
3. MUST set the following elements and send them in a Commit Message (section 3.2.4.2.1.1):
   - <HostID>, as acquired in section 3.1.6.1.
   - <Iteration>, as the new **Iter** value.
   - <HostValidateAuthenticator>, an HMAC, as specified in section 3.1.1, calculated as:

     Base64(HMAC(**_HostValidateNonce**$_{\textbf{Iter}}$, UTF-8 (**Iter** + **OTP**$_{\textbf{Iter}}$ + **_HostID** + **_HostCertificate**)).

If successful and this is the last iteration, the service:

1. MUST increment the iteration number, **Iter**.
2. MUST change **TrustState** from 3 (*Validating*) to 4 (*Committing*).
3. MUST set the following elements and send them in a Confirm Message (section 3.2.4.4.1.1):
   - <HostID>, as acquired in section 3.1.6.1.
   - <IterationsRequired>, as acquired in section 3.1.6.1.

- <HostConfirmNonce>, as used in section 3.2.4.4.2.1.

If this action fails, the control point MUST change **TrustState** to 0 (*Idle*), cancel the DTAG protocol, and report an error to the control point user of this protocol.

3.3.4.4 Confirm Response

This action is supported only when **TrustState** is 4 (*Confirming*). On this action, the following checks MUST be performed:

1. The <DeviceConfirmNonce> (**_DeviceConfirmNonce**) element MUST be syntactically validated.
2. The value of HMAC(**_DeviceConfirmNonce**, UTF-8(**N** + **OTP** + **_DeviceID** + **_DeviceCertificate**)), calculated as specified in section 3.1.1, MUST match the **_DeviceValidateAuthenticator** obtained in the **Exchange** response.

If successful then trust has been established, and the service:

1. MUST store **_DeviceID** and **_DeviceCertificate** in a tamper-proof, persistent store.
2. MUST change **TrustState** from 4 (*Confirming*) to 0 (*Idle*).
3. MUST report the success to the control point user of this protocol.

If this action fails, the control point MUST change **TrustState** to 0 (*Idle*), cancel the DTAG protocol, and report an error to the control point user of this protocol.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

3.3.6.1 One-time Password (OTP) Event

In addition to the local events specified in section 3.1.6.1, the control point:

1. MUST generate **_HostConfirmNonce**.
2. MUST set the following elements and send them in an Exchange Message (section 3.2.4.1.1.1):
   - <HostID>, as acquired in section 3.1.6.1.
   - <HostCertificate>, as acquired in section 3.1.6.1.
   - <HostConfirmAuthenticator>, an HMAC as specified in section 3.1.1, calculated as:

     Base64(HMAC(**_HostConfirmNonce**, UTF-8 (**N**+ **OTP**+ **_HostID**+ **_HostCertificate**)).

4 Protocol Examples

4.1 Trust Channel Establishment

This subsection demonstrates the entire sequence of DTAG messages for a successful exchange of certificates between the device and the control point.

4.1.1 Exchange Action Message

Upon the receipt of an out-of-band one-time password (OTP), the control point sends the Exchange SOAP request message to the device. The following example demonstrates an Exchange message where the <HostCertificate> and <HostConfirmAuthenticator> elements are encoded in a MIME base64 scheme and the requested iteration for the **Commit**-**Validate** action is four:

```
<?xml version="1.0"?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <m:Exchange xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
        <HostID xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
          uuid:dd5b556f-765f-4eed-8db9-ed8b5b79d036
        </HostID>
        <HostCertificate xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
          AAABAANiMIIDXjCCAkagAwIBAgIQHcJCKsL2arVHUTvyMIkNMTANBgkqhkiG9w0BAQUQa/
        </HostCertificate>
        <IterationsRequired xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="ui1">
          4
        </IterationsRequired>
        <HostConfirmAuthenticator xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
          u7S1cI9Kqdgt1uUCHNoeWMbkbbI=
        </HostConfirmAuthenticator>
      </m:Exchange>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.1.2 Exchange Response Message

If the Exchange message is verified without error, the device sends an ExchangeResponse SOAP response message to the control point. The following example demonstrates an ExchangeResponse message where the <DeviceCertificate> and <DeviceConfirmAuthenticator> elements are encoded in a MIME base64 scheme:

```
<s:Envelope xmlns:s=http://schemas.xmlsoap.org/soap/envelope/
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ExchangeResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceID>
        uuid:50995244-0612-43E0-8D59-56AFC0D2D49C
      </DeviceID>
      <DeviceCertificate>
        AAAEAAVcMIIFWDCCBECgAwIBAgIKd4AEJwAGAAAGDzANBgkqhkiG9w0BAQUFADCBlTELMA
      </DeviceCertificate>
      <DeviceConfirmAuthenticator>
        KNTZRl/E/qaBxEri688nkMtzw3g=
      </DeviceConfirmAuthenticator>
```

```
    </u:ExchangeResponse>
  </s:Body>
</s:Envelope>
```

4.1.3 Commit Action Message

If the ExchangeResponse message is verified without error, the control point sends a Commit SOAP request message to the device. The following example demonstrates a Commit message where the <HostValidateAuthenticator> element is encoded in a MIME base64 scheme:

```
<?xml version="1.0"?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
      <m:Commit xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
        <HostID xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
          uuid:dd5b556f-765f-4eed-8db9-ed8b5b79d036
        </HostID>
        <Iteration xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="ui1">
          1
        </Iteration>
        <HostValidateAuthenticator xmlns:dt="urn:schemas-upnp-org:service-1-0"
dt:dt="string">
          7of/dDyMM2nXs055JXpn4hoABwQ=
        </HostValidateAuthenticator>
      </m:Commit>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

4.1.4 Commit Response Message

If the Commit message is verified without error, the device returns a CommitResponse SOAP response message to the control point. The following example demonstrates a CommitResponse message where the <DeviceValidateAuthenticator> element is encoded in a MIME base64 format:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:CommitResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceValidateAuthenticator>
        kn0ylcPCkk811LhOFvrZlV4SfaM=
      </DeviceValidateAuthenticator>
    </u:CommitResponse>
  </s:Body>
</s:Envelope>
```

4.1.5 Validate Action Message

If the CommitResponse message is verified without error, the control point sends a Validate SOAP request message to the device. The following example demonstrates a Validate message where the <HostValidateNonce> element is encoded in a MIME base64 scheme:

```
<?xml version="1.0"?>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:Validate xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <HostID xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
        uuid:dd5b556f-765f-4eed-8db9-ed8b5b79d036
      </HostID>
      <Iteration xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="ui1">
        1
      </Iteration>
      <HostValidateNonce xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
        v9DcGzl+I+j2Pt/LgA6l2oaB86s=
      </HostValidateNonce>
    </m:Validate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.1.6 Validate Response Message

If the Validate message is verified without error, the device returns a ValidateResponse SOAP response message to the control point. The following example demonstrates a CommitResponse message where the <DeviceValidateNonce> is encoded in a MIME base64 format:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ValidateResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceValidateNonce>
        Fhn2k/oWWnq/eXeBr7ZObFfWon0=
      </DeviceValidateNonce>
    </u:ValidateResponse>
  </s:Body>
</s:Envelope>
```

Because the requested iterations are four at the previous **Exchange** action, as described in section 4.1.1, the **Commit** and **Validate** actions will be repeated four times.

4.1.7 Confirm Action Message

If the ValidateResponse message is verified without error, the control point sends a Confirm SOAP request message to the device. The following example demonstrates a Confirm message where the <HostConfirmNonce> element is encoded in a MIME base64 scheme:

```
<?xml version="1.0"?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <S:Body>
      <m:Confirm xmlns:m="urn:schemas-microsoft-com:service:mstrustagreement:1">
        <HostID xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
          uuid:dd5b556f-765f-4eed-8db9-ed8b5b79d036
        </HostID>
        <IterationsRequired xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="ui1">
          4
        </IterationsRequired>
        <HostConfirmNonce xmlns:dt="urn:schemas-upnp-org:service-1-0" dt:dt="string">
```

```
          cTUsZ3d7bHMxeTrL0+KNmSNMOAk=
        </HostConfirmNonce>
      </m:Confirm>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

4.1.8 Confirm Response Message

If the Confirm message is verified without error, the device returns a ConfirmResponse SOAP response message to the control point. The following example demonstrates a ConfirmResponse message where the <DeviceConfirmNonce> element is encoded in a MIME base64 scheme:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:ConfirmResponse xmlns:u="urn:schemas-microsoft-com:service:mstrustagreement:1">
      <DeviceConfirmNonce>
        LLZYoA31XRz56EU+GXHYUd3jo2Y=
      </DeviceConfirmNonce>
    </u:ConfirmResponse>
  </s:Body>
</s:Envelope>
```

After the control point verifies the response message, and if there is no error, DTAG is completed and a trust relationship is established between the control point and the device.

4.2 Error Message

If an error occurs while the device processes any request message, the device returns an error message instead of the response message to the control point. The following example demonstrates an error message on the **Validate** action, which indicates error code 803 (invalid nonce):

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>803</errorCode>
          <errorDescription>Invalid Nonce</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

5 Security

5.1 Security Considerations for Implementers

In general, DTAG provides protection at the strength of the one-time password (OTP), where the OTP is required to be:

- Cryptographically random and difficult to guess.
- Transported to the endpoints in an out-of-band manner, such as through user interaction, the details of which are not described in this specification. For this purpose, the OTP can be relatively short enough for the user to remember.
- Generated anew each time DTAG is started or restarted.
- The number of OTP characters is required to be equal to or greater than the number of iterations.

The number of validate rounds (**N**) is required to be at least 2, with a minimum of 4 recommended.

5.2 Index of Security Parameters

Security parameter	Section
One-time Password (OTP)	3.1.6.1
N	3.1.1

6 Appendix A: Full WSDL

This protocol does not contain a WSDL. For UPnP, the equivalent to WSDL are the UPnP device and service descriptions. Please see the UPnP device description in section 7 and the full UPnP service description in section 8.

7 Appendix B: UPnP Device Description

The following is a sample service information of DTAG, which the device description has to include as part of the service list for the device.

The default namespace, "urn:schemas-upnp-org:device-1-0", is specified in [UPNPARCH1] sections 2.1 and 2.6.

```
<?xml version='1.0'?>
<root xmlns="urn:schemas-upnp-org:device-1-0"
         xmlns:pnpx="http://schemas.microsoft.com/windows/pnpx/2005/11" >
 <specVersion>
      <major>1</major>
      <minor>0</minor>
    </specVersion>
 <device>
<pnpx:X_deviceCategory>MediaDevices</pnpx:X_deviceCategory>
  <deviceType>urn:schemas-microsoft-com:device:MediaCenterExtenderMFD:1</deviceType>
  <friendlyName>Xbox 360 Media Center Extender</friendlyName>
  <manufacturer>Microsoft Corporation</manufacturer>
  <manufacturerURL>http://www.xbox.com/</manufacturerURL>
  <modelDescription>Xbox 360 Media Center Extender</modelDescription>
  <modelName>Xbox 360</modelName>
  <modelNumber></modelNumber>
  <modelURL>http://go.microsoft.com/fwlink/?LinkID=53081</modelURL>
  <serialNumber></serialNumber>
  <UDN>uuid:10000000-0000-0000-0200-00125A702E78</UDN>
  <UPC></UPC>
  <iconList>
        <icon>
          <mimetype>image/jpeg</mimetype>
          <width>48</width>
          <height>48</height>
          <depth>24</depth>
          <url>/IconSM.jpg</url>
        </icon>
        <icon>
          <mimetype>image/jpeg</mimetype>
          <width>120</width>
          <height>120</height>
          <depth>24</depth>
          <url>/IconLRG.jpg</url>
        </icon>
        <icon>
          <mimetype>image/png</mimetype>
          <width>48</width>
          <height>48</height>
          <depth>24</depth>
          <url>/IconSM.png</url>
        </icon>
        <icon>
          <mimetype>image/png</mimetype>
          <width>120</width>
          <height>120</height>
          <depth>24</depth>
          <url>/IconLRG.png</url>
        </icon>
        <icon>
```

```
            <mimetype>image/png</mimetype>
            <width>152</width>
            <height>152</height>
            <depth>24</depth>
            <url>/IconMCE.png</url>
          </icon>
        </iconList>
    <serviceList>
          <service>
            <serviceType>urn:schemas-microsoft-com:service:NULL:1</serviceType>
            <serviceId>urn:microsoft-com:serviceId:NULL</serviceId>
            <SCPDURL>/XD/NULL.xml</SCPDURL>
            <controlURL>/UD/?0</controlURL>
            <eventSubURL/>
          </service>
        </serviceList>
    <deviceList>
        <device xmlns:mcx="http://schemas.microsoft.com/windows/mcx/2007/06"
  xmlns:nss="urn:schemas-microsoft-com:WMPNSS-1-0">
      <pnpx:X_compatibleId>MICROSOFT_MCX_0001</pnpx:X_compatibleId>
      <pnpx:X_deviceCategory>MediaDevices</pnpx:X_deviceCategory>
      <mcx:pakVersion>dv2.0.0</mcx:pakVersion>
      <mcx:supportedHostVersions>pc2.0.0</mcx:supportedHostVersions>
      <nss:X_magicPacketSendSupported>1</nss:X_magicPacketSendSupported>
      <deviceType>urn:schemas-microsoft-com:device:MediaCenterExtender:1</deviceType>
      <friendlyName>Xbox 360 Media Center Extender</friendlyName>
      <manufacturer>Microsoft Corporation</manufacturer>
      <manufacturerURL>http://www.microsoft.com/</manufacturerURL>
      <modelDescription>Xbox 360 Media Center Extender</modelDescription>
      <modelName>Xbox 360</modelName>
      <modelNumber></modelNumber>
      <modelURL>http://go.microsoft.com/fwlink/?LinkID=53081</modelURL>
      <serialNumber></serialNumber>
      <UDN>uuid:20000000-0000-0000-0200-00125A702E78</UDN>
      <UPC></UPC>
      <iconList>
          <icon>
            <mimetype>image/jpeg</mimetype>
            <width>48</width>
            <height>48</height>
            <depth>24</depth>
            <url>/IconSM.jpg</url>
          </icon>
          <icon>
            <mimetype>image/jpeg</mimetype>
            <width>120</width>
            <height>120</height>
            <depth>24</depth>
            <url>/IconLRG.jpg</url>
          </icon>
          <icon>
            <mimetype>image/png</mimetype>
            <width>48</width>
            <height>48</height>
            <depth>24</depth>
            <url>/IconSM.png</url>
          </icon>
          <icon>
            <mimetype>image/png</mimetype>
```

```
            <width>120</width>
            <height>120</height>
            <depth>24</depth>
            <url>/IconLRG.png</url>
          </icon>
          <icon>
            <mimetype>image/png</mimetype>
            <width>152</width>
            <height>152</height>
            <depth>24</depth>
            <url>/IconMCE.png</url>
          </icon>
        </iconList>
      <serviceList>
          <service>
           <serviceType>urn:schemas-microsoft-com:service:mstrustagreement:1</serviceType>
            <serviceId>urn:microsoft-com:serviceId:MSTA</serviceId>
            <SCPDURL>/XD/mstrustagreement.xml</SCPDURL>
            <controlURL>/UD/?1</controlURL>
            <eventSubURL />
          </service>
        </serviceList>
    </device>
   </deviceList>
  </device>
 </root>
```

8 Appendix C: Full UPnP Service Description

The following is a sample service description of DTAG, which the device has to publish as a prerequisite before DTAG can take any action, as described in section 1.5.

The default namespace, "urn:schemas-upnp-org:service-1-0", is specified in [UPNPARCH1] sections 2.3 and 2.7.

```
<?xml version="1.0" ?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
<specVersion>
      <major>1</major>
  <minor>0</minor>
  </specVersion>
<actionList>
<action>
      <name>Exchange</name>
<argumentList>
    <argument>
  <name>HostID</name>
  <direction>in</direction>
  <relatedStateVariable>A_ARG_TYPE_EndpointID</relatedStateVariable>
    </argument>
    <argument>
        <name>HostCertificate</name>
        <direction>in</direction>
        <relatedStateVariable>A_ARG_TYPE_Certificate</relatedStateVariable>
    </argument>
    <argument>
        <name>IterationsRequired</name>
        <direction>in</direction>
        <relatedStateVariable>A_ARG_TYPE_Rounds</relatedStateVariable>
    </argument>
        <argument>
        <name>HostConfirmAuthenticator</name>
        <direction>in</direction>
        <relatedStateVariable>A_ARG_TYPE_Authenticator</relatedStateVariable>
      </argument>
        <argument>
        <name>DeviceID</name>
        <direction>out</direction>
        <relatedStateVariable>A_ARG_TYPE_EndpointID</relatedStateVariable>
      </argument>
        <argument>
        <name>DeviceCertificate</name>
        <direction>out</direction>
        <relatedStateVariable>A_ARG_TYPE_Certificate</relatedStateVariable>
        </argument>
        <argument>
        <name>DeviceConfirmAuthenticator</name>
        <direction>out</direction>
        <relatedStateVariable>A_ARG_TYPE_Authenticator</relatedStateVariable>
        </argument>
  </argumentList>
  </action>
    <action>
    <name>Commit</name>
```

```
  <argumentList>
      <argument>
      <name>HostID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_EndpointID</relatedStateVariable>
      </argument>
      <argument>
      <name>Iteration</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_Iteration</relatedStateVariable>
    </argument>
      <argument>
      <name>HostValidateAuthenticator</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_Authenticator</relatedStateVariable>
    </argument>
      <argument>
      <name>DeviceValidateAuthenticator</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_Authenticator</relatedStateVariable>
      </argument>
</argumentList>
</action>
  <action>
  <name>Validate</name>
  <argumentList>
     <argument>
        <name>HostID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_EndpointID</relatedStateVariable>
   </argument>
     <argument>
      <name>Iteration</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_Iteration</relatedStateVariable>
     </argument>
     <argument>
      <name>HostValidateNonce</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_Nonce</relatedStateVariable>
     </argument>
     <argument>
      <name>DeviceValidateNonce</name>
      <direction>out</direction>
      <relatedStateVariable>A_ARG_TYPE_Nonce</relatedStateVariable>
     </argument>
</argumentList>
</action>
  <action>
  <name>Confirm</name>
  <argumentList>
     <argument>
      <name>HostID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_EndpointID</relatedStateVariable>
   </argument>
     <argument>
      <name>IterationsRequired</name>
      <direction>in</direction>
```

```
          <relatedStateVariable>A_ARG_TYPE_Rounds</relatedStateVariable>
        </argument>
        <argument>
          <name>HostConfirmNonce</name>
          <direction>in</direction>
          <relatedStateVariable>A_ARG_TYPE_Nonce</relatedStateVariable>
      </argument>
        <argument>
          <name>DeviceConfirmNonce</name>
          <direction>out</direction>
          <relatedStateVariable>A_ARG_TYPE_Nonce</relatedStateVariable>
      </argument>
      </argumentList>
    </action>
  </actionList>
    <serviceStateTable>
      <stateVariable sendEvents="no">
         <name>TrustState</name>
           <dataType>ui1</dataType>
        <allowedValueRange>
          <minimum>0</minimum>
            <maximum>4</maximum>
     </allowedValueRange>
    </stateVariable>
      <stateVariable sendEvents="no">
           <name>A_ARG_TYPE_Rounds</name>
        <dataType>ui1</dataType>
        <allowedValueRange>
           <minimum>2</minimum>
           <maximum>20</maximum>
     </allowedValueRange>
    </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Iteration</name>
           <dataType>ui1</dataType>
        <allowedValueRange>
           <minimum>1</minimum>
           <maximum>20</maximum>
      </allowedValueRange>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_EndpointID</name>
           <dataType>string</dataType>
    </stateVariable>
      <stateVariable sendEvents="no">
           <name>A_ARG_TYPE_Authenticator</name>
           <dataType>string</dataType>
    </stateVariable>
      <stateVariable sendEvents="no">
           <name>A_ARG_TYPE_Nonce</name>
           <dataType>string</dataType>
    </stateVariable>
      <stateVariable sendEvents="no">
           <name>A_ARG_TYPE_Certificate</name>
           <dataType>string</dataType>
    </stateVariable>
  </serviceStateTable>
  </scpd>
```

9 Appendix D: Product Behavior

The information in this specification is applicable to the following Microsoft products:

- Windows Vista® operating system
- Windows® 7 operating system

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. The new behavior also applies to subsequent service packs of the product unless otherwise specified.

Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that product does not follow the prescription.

10 Change Tracking

This section identifies changes made to [MS-DTAG] protocol documentation between December 2009 and January 2010 releases. Changes are classed as major, minor, or editorial.

Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- A protocol is deprecated.
- The removal of a document from the documentation set.
- Changes made for template compliance.

Minor changes do not affect protocol interoperability or implementation. Examples are updates to fix technical accuracy or ambiguity at the sentence, paragraph, or table level.

Editorial changes apply to grammatical, formatting, and style issues.

No changes means that the document is identical to its last release.

Major and minor changes can be described further using the following revision types:

- New content added.
- Content update.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.

- Content removed for template compliance.
- Obsolete document removed.

Editorial changes always have the revision type "Editorially updated."

Some important terms used in revision type descriptions are defined as follows:

Protocol syntax refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.

Protocol revision refers to changes made to a protocol that affect the bits that are sent over the wire.

Changes are listed in the following table. If you need further information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Revision Type
1 Introduction	Removed citation [MS-DTAG].	N	Content update.

11 Index

T

U

V

W