SecurityConsole:1 Service Template

For UPnP™ Device Architecture 1.0 [1]
Status: Standardized DCP
Date: November 17*, 2003*

This Standardized DCP has been adopted as a Standardized DCP by the Steering Committee of the UPnP™ Forum, pursuant to Section 2.1(c)(ii) of the UPnP™ Forum Membership Agreement. UPnP™ Forum Members have rights and licenses defined by Section 3 of the UPnP™ Forum Membership Agreement to use and reproduce the Standardized DCP in UPnP™ Compliant Devices. All such use is subject to all of the provisions of the UPnP™ Forum Membership Agreement.

THE UPNP™ FORUM TAKES NO POSITION AS TO WHETHER ANY INTELLECTUAL PROPERTY RIGHTS EXIST IN THE STANDARDIZED DCPS. THE STANDARDIZED DCPS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". THE UPNP™ FORUM MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE STANDARDIZED DCPS, INCLUDING BUT NOT LIMITED TO ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, OF REASONABLE CARE OR WORKMANLIKE EFFORT, OR RESULTS OR OF LACK OF NEGLIGENCE.

Authors	Company
Carl Ellison	Intel Corporation

[1] UPnP™ is a service mark of the UPnP™ Implementers Corporation.

Contents

List of Tables

1. Overview and Scope

This service is offered by a Security Console (SC). The Security Console offers a user interface for administration of access control on security-aware UPnP devices. [See DeviceSecurity:1 for a description of the actions used in the creation and editing of Access Control Lists (ACLs) and in taking security ownership of Devices.] As a device the Security Console is self-owned. If it has any access controlled actions, then those are to be administered by the human user and not by some other Security Console. Therefore, a Security Console does not need to include a DeviceSecurity service. It does have a certificate cache, but it is an outgoing cache, rather than an incoming cache.

A network built of the user's own components with no connection to anything outside the user's personal domain and with no control points belonging to anyone other than the user ever attached to the network would not require the features of UPnP Security. Network isolation would already have achieved a level of physical security. We are concerned in UPnP Security with networks in which more than the user's own Control Points are present on the physical network and able to reach the user's Devices with control messages. These situations can include:

1. use of wireless, power-line networking or cable modem without a firewall, allowing an attacker to join the network without the user's knowledge or permission
2. shared infrastructure networks, such as a college dorm or a condominium building wired for Ethernet as one network segment serving more than one person's residence
3. households of multiple adults or teens, in which each individual wants to establish a private security domain, in addition to any domain of devices or control points shared among them, while using a shared network domain
4. connections to the Internet via devices or services that create single network segments of multiple subscribers as a side effect of offering network connectivity (such as some cable modems and some ISP connections)
5. households in which guests might bring mobile devices or control points into the network temporarily

In such networks of intentional or accidental sharing, one cannot rely on physical network security to protect devices or on discovery methods (e.g., multicast SSDP) to compile a list of "My Devices" or "My Control Points". This leaves it up to the user manually to select from physically accessible devices and control points, choosing those of interest to that user. One primary function of the SC is to enable the user to make that selection. This process requires two operations that were not anticipated in the original design of UPnP:

1. discovery of control points; and
2. naming of devices and control points on a per-user basis.

The actions provided in this service allow the SC to perform those two functions.

In addition, the sharing of devices across security domains sometimes calls for the use of authorization certificates, as described in sections 1.1.3 and 3.3. This service provides actions for the delivery of such certificates (or certificate chains) (see 2.5.3) and for the revocation (via renewal) of certificates (see 2.5.4).

1.1. Security Console Actions

When the Security Console interacts with a security-aware device, it does so through actions offered by that device. However, the Security Console must also interact with control points (CPs). Instead of

forcing CPs to become devices as well, in order to support these interactions, we define actions that a SC offers. The actions of this service fall in three functional categories:

1. discovery of control points
2. communication of dictionaries of local names
3. processing of certificates

1.1.1. Control Point Discovery

UPnP 1.0 includes a protocol, SSDP, for discovery of devices by control points. However, there is no protocol for discovery of control points by other control points or devices.

The Security Console needs to discover control points so that it can identify those that should receive access rights on devices in the local security domain. We achieve this discovery by reversing the logic of UPnP discovery. A security-aware control point will discover a SC that offers the PresentKey action and will then invoke that action to announce itself to the SC. Since a CP might act within multiple security domains, it should announce itself to every SC it detects. The mere act of announcing itself does not imply that it will receive any rights, since the assignment of rights is an expression of a user's decision. However, a CP cannot know ahead of time whether a particular SC will choose to grant it some rights and must therefore announce itself to all SCs.

1.1.2. Local Dictionary Communication

One primary function of the SC is to identify devices and control points in the user's local network. In at least one implementation of the SC, this process includes permitting the user to assign names of the user's own choosing (local names) to those devices and control points. Since devices and control points might be visible to (and therefore named within) different security domains operated by different users, a single device or control point could have different local names. Therefore, these names remain the property of the user (specifically the SC) rather than the named device or control point itself. Normally, they would reside within and not be released from the SC.

For example, consider two roommates, Joe and Sue, sharing a network in their Cambridge apartment. Each has a personal domain of UPnP devices and control points, but some components are shared between them. One shared device is Joe's archive of digital photos. Joe refers to it by the name "pix", while Sue names it "Joe's Snapshot Archive". Neither name fits the preferences of the other user; therefore, neither name is appropriate as the sole friendly name for the shared device. Meanwhile, the archive device is known on the network by a unique name such as `DE7Z-GVGK-QTYR-TWPO-YF54-GB4M-OGFH-XJYM` that neither user wants to deal with. The mapping from friendly name to unique name is the function of each user's user interface (the Security Console, in this case). That mapping is referred to here as a "local dictionary".

It is possible that this local dictionary of "My Devices and Control Points" might be useful to other components within the user's domain. For example, Joe might have two computers on the network, on one of which he named his personal devices, but on the second computer he would prefer just to import all names from the first computer, rather than go to all the work of manually assigning names again to each of his devices. To support such cases, we provide for access to that dictionary, via the GetNameList action, and we also provide for an event notification whenever that name list changes.

1.1.3. Certificate Processing

The Security Console is responsible for granting access rights to devices under its control. If a device is shared among multiple domains, there will be multiple Security Consoles that need to grant rights on that device. This sharing of the right to grant access can be achieved through co-ownership (see

GrantOwnership, in DeviceSecurity:1), but a co-owner has total access to a device and is, among other things, capable of removing all access rights of the first owner including its ownership status. If that is too much power to share with some other SC, that other SC can be granted permissions via the device's Access Control List, just like any control point. In that case, that SC will grant rights to CPs (or still other SCs) not by adding ACL entries, since it does not have the right to edit the ACL, but rather via authorization certificates. (See DeviceSecurity:1 for a definition of authorization certificates.)

It is possible that a Security Console that does have ownership of a device might also grant rights by certificate, for example if that device has too little storage for a detailed ACL or if the device is offline at the time the access right needs to be granted.

The authorization certificate is like an ACL entry, but it is digitally signed and includes an issuer and specification of the device(s) to which it applies. It will probably also include at least an expiration date and time.

There are two actions provided here to facilitate the processing of certificates:

1. **GetMyCertificates:** which serves as a post office mechanism to allow a control point or other security console to fetch certificates that have been issued to it by this SC (This action is backed up by an evented variable, PendingCPList, by which the CP or other SC can learn that there are certificates waiting.); and

2. **RenewCertificate:** by which a control point can request an updated copy of an expired (or soon to expire) certificate. For more details about renewal, see section 3, Theory of Operation.

Although GetMyCertificates provides a communication mechanism for certificates, that does not preclude other communication mechanisms to be implemented by Security Console applications. For example, one might use e-mail, sneaker-net, some directory service or HTTP for this communication function. In a truly complex network with a large number of certificates, one might have an intelligent directory service that returns to a CP precisely the certificate chain it needs to access a particular action on a particular device. These are application design issues and out of scope of this protocol specification. GetMyCertificates stands as a common denominator, to insure interoperability (assuming components that share a network at least occasionally).

2. Service Modeling Definitions

2.1. Service Type

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

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

The shorthand SecurityConsole:1 is used herein to refer to this service type.

2.2. Namespaces

The XML in this document should be read as if the following namespace definitions were in effect.

xmlns="urn:schemas-upnp-org**:**service:DeviceSecurity:1"

xmlns:us="urn:schemas-upnp-org**:**service:DeviceSecurity:1"

xmlns:sc="urn:schemas-upnp-org**:**service:SecurityConsole:1"

xmlns:ds=" http://www.w3.org/2000/09/xmldsig#"

2.3. State Variables

SecurityConsole:1 defines two state variables: PendingCPList and NameListVersion.

Table 1: State variable

Variable Name	Req. or Opt. [1]	Data Type	Default Value
PendingCPList	O	string	<CPList></CPList>
NameListVersion	O	string	
A_ARG_TYPE_string	R	string	
A_ARG_TYPE_base64	R	bin.base64	

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

2.3.1. PendingCPList

The PendingCPList is the string encoding of an XML document giving the list of Control Points (specifically the hashes of those Control Point keys) that have certificates waiting to be fetched via GetMyCertificates. This variable is optional: not needed if there is no certificate processing done by this Security Console. For example, the XML document might be as follows. We use white space here for readability, but since this structure is for computer-to-computer communication it need have no white space.

```
<CPList>
  <hash>
    <algorithm>SHA1</algorithm>
    <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value>
  </hash>
```

```
  <hash>
    <algorithm>SHA1</algorithm>
    <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value>
  </hash>
</CPList>
```

2.3.2. NameListVersion

The NameListVersion variable is modified whenever a change is made to the SC's name definitions. Subscription to this variable allows a slaved SC to know when to ask for a new name definition list. The variable value itself has no meaning. Its purpose is merely to notify a subscribed SC that there is a modified name list to be fetched. This variable is optional: not needed if there is no GetNameList action implemented. The variable could, for example, be a counter that gets incremented or a BASE64 encoding of the hash of the name list.

2.3.3. A_ARG_TYPE_string

This is a dummy state variable, for being a related variable to indicate that an argument is a string, possibly escaped XML.

2.3.4. A_ARG_TYPE_base64

This is a dummy state variable, for being a related variable to indicate that an argument is a BASE64 encoding of a (usually binary) byte string.

2.4. Eventing and Moderation

Table 2: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
PendingCPList	*yes*	*N/A*	*N/A*	*N/A*	*N/A*
NameListVersion	*yes*	*N/A*	*N/A*	*N/A*	*N/A*
Non-standard state variables implemented by an UPnP vendor go here.	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

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

2.5. Actions

As the table below summarizes, SecurityConsole:1 defines actions used to communicate with control points or other security consoles. These provide for discovery of control points, communication of the set of names of devices in a personal domain and processing of authorization certificates. Somcatcat

GetNameList	O
GetMyCertificates	O
RenewCertificate	O
Non-standard actions implemented by an UPnP vendor go here.	X

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

2.5.1. PresentKey

PresentKey accepts an offered key hash from a control point on the network, in order to do "discovery" of control points without forcing them to become UPnP Devices and announce their existence by SSDP. Other Security Consoles are expected to announce themselves to Security Consoles via PresentKey as well, since from the point of view of a Security Console, another SC is just a CP.

2.5.1.1. Arguments

Argument(s)	Direction	relatedStateVariable
HashAlgorithm	IN	A_ARG_TYPE_string
Key	IN	A_ARG_TYPE_string
PreferredName	IN	A_ARG_TYPE_string
IconDesc	IN	A_ARG_TYPE_string

A Control Point (CP) or Security Console (SC) is identified by its public key. The hash algorithm (SHA1 for now, with others possible later) and key are given in the first two parameters. The key is encoded as an XML structure, properly escaped for transmission. This structure should be as described in DeviceSecurity:1, in the section entitled "Public Keys and their Hashes". It is hashed on receipt, using the indicated hash algorithm, and that hash value is stored. It is also presented to the user under the guise of a "Security ID" (in BASE32 encoding) for comparison to the Security ID shipped with or displayed by the CP or SC calling PresentKey. Using that Security ID, the user assigns a name to the key and therefore the CP or SC.

The PreferredName argument is a descriptive, friendly name for the calling CP or SC. It is available for the SC to use until the SC's user chooses a personal name for that caller. Note: in any large network with no physical security, it is easy to discover multiple callers with the same friendly name, either by popularity of some control point or by deliberate attack. Therefore, it is important that the process by which a Security Console accepts a presented key into the category of "My Control Points" (or whatever it would be called) should include examination of the full hash of the key. For presentation of this value to users, we have defined a BASE32 mapping, as described in section 3.6, below.

The CP can also offer its own icon for display, but the IconDesc is allowed to be empty. If it is non-empty, it should be an (escaped) XML structure of the form:

```
<icon>
  <mimetype>image/format</mimetype>
  <width>horizontal pixels</width>
  <height>vertical pixels</height>
  <depth>color depth</depth>
  <url>URL to icon</url>
</icon>
```

2.5.1.2. Effect on State

Unless the offered key hash is already known, it is added to the pool of CP key hashes waiting to be named. If the offered key is already known, there is no action. Naming of key hashes from that pool is a manual operation that may occur sometime after the completion of this action.

2.5.1.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.5.2. GetNameList

2.5.2.1. Arguments

Argument(s)	Direction	relatedStateVariable
Names	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

The action returns an XML element, encoded as a string as per section 3.7 below, containing all of the names defined by the SC. Group names, defined by certificate (as described in DeviceSecurity:1), are not listed since they are available by certificate.

The data structure returned in the Names argument is digitally signed by the originating Security Console.

If the user's network is to be protected from inventory-taking, then this action should be access controlled – with the ACL controlling it edited manually by the user who operates this Security Console.

This XML element is of the form of a list of name definitions, with the whole list signed by the Security Console signature key. For example, a name list of one device and one control point (including its signature) might be:

```
<SignedNameList>
  <Names us:Id="NameList">
    <Device>
      <name>Joe's Snapshot Archive</name>
      <hash>
        <algorithm>SHA1</algorithm>
        <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value>
      </hash>
    </Device>
    <CP>
      <name>Joe's PC</name>
      <hash>
        <algorithm>SHA1</algorithm>
        <value>CC0FQNQuS2S5S22aQnFdmST4tnw=</value>
      </hash>
    </CP>
  </Names>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="minimal"/>
```

```
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
      <Reference URI="#NameList">
        <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>SiGg1/kFmfx7aQ4XWq56rdUQfyo=</DigestValue>
      </Reference>
    </SignedInfo>
<SignatureValue>Tx3dGYKl8UWjx00Q+fE0aYKlMcr2UTO96shC/duR9xYkFY2za5UEVrf8
o22mBEq7LQg3LQF9L5EpLpChtXZEgQ==</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
<Modulus>tPK7xYLJqm77saltSus77darlxIHHWNajVEdxlwV7YmlnUyp/plhKltFr1jXzoz
XfPWc3ZwN6JfpdbyDwlJ74Q==</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</SignedNameList>
```

Note that the digest and signature values in the example above were not calculated from this example, so they will fail to verify. Note also that we use white space to make the XML more readable, while we expect real XML on the wire to use no unnecessary white space, since this structure is for communication between two machines.

The caller needs to verify the signature before displaying or otherwise relying on those names.

2.5.2.2. Effect on State

None.

2.5.2.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
701	Not authorized	Authorization failure (action probably not signed by trusted control point public key).
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.5.3. GetMyCertificates

GetMyCertificates retrieves and returns the full set of certificates being held for the indicated CP, if any. The purpose is to provide for communications of certificates, not for a directory of all certificates ever issued. It is the responsibility of the CP to store the certificates that empower it, whether internally or via some network directory or backup service. At the discretion of the Security Console application developer, other delivery mechanisms may be used, but this one is provided as a common denominator among components from different manufacturers. It is also up to the discretion of the Security Console developer whether and how to back up all working data of that SC, possibly including the set of all certificates it has generated. It is up to either the Security Console developer or the human user operating the SC to determine when a given certificate will no longer be returned to the caller by this action. That is, the

developer or end user decides when to flush a given certificate from the cache of certificates held for a given CP.

2.5.3.1. Arguments

Argument(s)	Direction	relatedStateVariable
HashAlgorithm	IN	A_ARG_TYPE_string
Hash	IN	A_ARG_TYPE_base64
Certificates	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

The argument Hash is the hash of the key of the CP whose certificates are being fetched. HashAlgorithm gives the algorithm used in that hash, currently SHA1. The hash is BASE64 encoded, for example:

```
Gd48BqQzAMPn4FkWnFslMMdxSG4=
```

The Certificates argument is an escaped XML document containing the returned set of certificates, in the following format:

```
<Sequence>{<cert>…</cert><ds:Signature>…</ds:Signature>}*</Sequence>
```

where `<cert>` is defined in DeviceSecurity:1.

2.5.3.2. Effect on State

When a CP gets its own certificates, its hash is removed from the PendingCPList. That hash is added back to the PendingCPList when a new CP certificate is added or an old one is changed. A CP's retrieval of its own certificates is established only if the CP signed the call to GetMyCertificates. If GetMyCertificates is called by some other CP (or SC) or by an anonymous caller, then the CP's hash is not removed from the PendingCPList.

How GetMyCertificates is implemented is up to the Security Console developer. For example, one might maintain what looks like a local copy of an ACL and note, internally, for each ACL entry whether it is an actual ACL entry or a certificate. When an entry changes in that local ACL ghost copy, if it is a real ACL entry, the SC can call ReplaceACLEntry, and if it is a certificate entry, the SC can add the subject CP to the PendingCPList. In such an implementation, some certificate entries would be enabled by a chain of certificates allowing the SC to grant some set of privileges. In that case, this chain of empowering certificates would be referred to by the ghost ACL entry and GetMyCertificates would return the entire empowering chain, not just the final certificate.

More complex certificate chain discovery can be done by a service yet to be defined, if we discover installations that require complex certificate chains, large named groups of CPs, etc. We do not anticipate such installations at this time.

2.5.3.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
732	No certificates	There were no certificates at this SC for this caller.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.5.4. RenewCertificate

We assume that the user of a Security Console will appear to be editing ACL entries empowering various control points or other security consoles, whether those authorizations happen by ACL editing or certificate issuance. In the case that the authorization is by certificate, it becomes necessary to model the act of deleting a specific entry. Rather than use elaborate certificate revocation mechanisms, we use the simple renewal mechanism. A certificate is issued with a short lifetime. The actual lifetime used is established by the Security Console that issues that certificate, perhaps with user input. However, the certificate is marked as being subject to renewal. The lifetime used is not a true expiration date for the certificate, but rather a length of time after which the certificate needs to be synchronized with the SC's image of what the ACL would have been, had authorization been performed by ACL editing.

A certificate that is subject to renewal will have an additional element, `<renew/>` in its `<valid>` element:

```
<valid>
  <not-before> … </not-before>
  <not-after> … </not-after>
  <renew></renew>
</valid>
```

The `<renew/>` element indicates that the certificate in question can be renewed. To renew it, one sends the old certificate body to the issuer's RenewCertificate action and, if the authorization has not been deleted, the issuer SC generates and returns a new certificate with validity period starting at the present time and ending after the **renewal interval** from the present time. [See section 3.5 for a definition of renewal interval.] How much ahead of time a CP chooses to renew an existing certificate is not specified here. For example, the CP can renew a certificate when it is half-way through its lifetime, assuming the issuing SC is available. As long as the SC is expected to be online and in operation at least once in half the renewal interval, that algorithm would allow service to continue uninterrupted.

2.5.4.1. Arguments

Argument(s)	Direction	relatedStateVariable
OldCertificate	IN	A_ARG_TYPE_string
NewCertificate	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

Note: the XML in this document is formatted for reading. We expect that real XML on the wire will not use any extra white space.

The OldCertificate argument is an XML element such as:

```
<cert>
  <issuer>
    <hash>
      <algorithm>SHA1</algorithm>
      <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value>
    </hash>
  </issuer>
  <subject>
    <hash>
      <algorithm>SHA1</algorithm>
      <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value>
    </hash>
  </subject>
```

```
  <may-not-delegate/>
  <tag>
    <device>
      <hash>
        <algorithm>SHA1</algorithm>
        <value>2jmj7l5rSw0yVb/vlWAYkK/YBwk=</value>
      </hash>
    </device>
    <access><p1/><p2/></access>
  </tag>
  <valid>
    <not-before>2001-09-01T17:00:00Z</not-before>
    <not-after>2001-10-01T17:00:00Z</not-after>
    <renew></renew>
  </valid>
</cert>
```

The old certificate needs to have been issued by the SC being called. The `<Signature>` normally associated with a certificate is not to be present in the OldCertificate. The us:Id attribute in <cert> is therefore not needed.

On output, if the indicated authorization is still in force, a new `<cert>` is returned, with new dates and a new, valid `<Signature>` element, using the XML form:

`<Sequence><cert>…</cert><ds:Signature>…</ds:Signature></Sequence>`

2.5.4.2. Effect on State

There is no effect on visible state. Depending on the Security Console developer, there may be a record kept of the last time a given certificate was renewed. Alternatively, one might keep a ghost ACL, as described in section 2.5.3.2 and include in a renewable certificate entry the length of time that any issued certificate should live, so that the certificate actually generated would be set to expire at that length of time after the time of the RenewCertificate call. See section 3.5 for more information about the certificate renewal process.

2.5.4.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
733	Revoked	Certificate has been revoked.
734	Not issued here	Certificate provided was not issued by this SC
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.6. Relationships between Actions

The actions presented here are independent, except indirectly in that a certificate must be delivered to someone before it would be renewed and that a CP or SC must be known before a certificate can be given to it.

2.7. Common Error Codes

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

errorCode	errorDescription	Description
401	Invalid Action	See UPnP Device Architecture section on Control.
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
800-899	*TBD*	*(Specified by UPnP vendor.)*

3. Theory of Operation

From the point of view of UPnP, the Security Console is a user application that is both a device and a control point. Its function is to give the user an interface by which to administer access control on the user's own devices. As part of that function, the Security Conso

3.2.1. Hardware alternatives

If the UPnP Security Working Committee were free to specify that every UPnP component were to have an extra hardware port, such as a USB connector, then one could devise physically secured hardware channels for introduction of components and secure transfer of their keys to a Security Console, without requiring the user ever to see the globally unique ID, much less verify it.

If some manufacturer chooses to build components with such a second channel, for security uses, and deliver a security console capable of using those second channels, then the user would no doubt welcome the ease of use. This document in general and the PresentKey mechanism in particular are not meant to preclude some manufacturer from providing such a channel. However, PresentKey is required, rather than optional, so that an SC can deal with Control Points that are not so equipped or that cannot be brought into range of the SC for the physical second channel to be used.

3.3. Certificates

The most straight-forward method of granting access permission is through modification of the device's Access Control List (ACL). Actions that enable that modification are described in the DeviceSecurity:1 service definition.

In some cases, it is not possible or not desirable to grant permission by modifying an ACL. These include:

- when the Security Console (SC) that is granting that permission (that is, is in communication with the CP that is being granted the permission), does not have permission to edit the ACL of the device
- when the granting SC is not in communication with the target device
- when the device does not have enough memory to hold any more ACL entries

In these cases, access is granted by authorization certificate. An authorization certificate can be thought of as a digitally signed ACL entry. It is defined more formally in DeviceSecurity:1.

For ease of administration, a Security Console might define named groups of control points. Such named groups are defined via group-membership Name Certificates, defined in DeviceSecurity:1.

3.4. Certificate Delivery

A certificate (or a chain of such certificates) must be available at the device at the time of any request authorized by that certificate, in order to prove the right to perform the requested action. This certificate must somehow be communicated from the SC that generates it to the device that applies it. UPnP supports two methods:

- The certificate can be cached by the target device, directly.
- The certificate can be held by the Control Point (or Security Console) to which the permission is being granted.

If the certificate is to be communicated to a device, then it can be written to that component via a SOAP CacheCertificate action, as defined in DeviceSecurity:1. A Control Point, however, does not offer SOAP actions. To overcome this lack, when the granting SC has a certificate to communicate to a CP, it caches that certificate itself, advertises the existence of that certificate (via the PendingCPList evented variable) and waits for the CP to fetch its pending certificates via the GetMyCertificates action. The same method is available to be used to communicate a certificate to another SC (since to a security console, an SC looks just like a CP).

3.5. Certificate Renewal

When access is granted by modification of an ACL, one is free to delete an ACL entry. We imagine that a user, operating a Security Console, may be given the impression of editing an ACL whether authorizations are granted by ACL edit or by issuance of authorization certificates.

If the user wants to delete one of these authorizations and it had been issued by certificate, there is a problem. The certificate is not under the control of the user's SC. It is in the hands of the CP that was granted the authorization involved. There could have been an unlimited number of copies made of the certificate, so deleting one copy of the certificate does not delete the authorization the way deletion of an ACL entry does.

In general, there are three ways to effectively delete a certificate:

1. to have it expire, by way of its `<not-after>` date and time;
2. to have a certificate validator keep an up to date revocation list; or
3. to have the certificate validator do an online test (as in OCSP[2]) with every validation.

Of these methods, UPnP Security has chosen to implement #1, as the simplest. Method #2, a revocation list (or CRL), is very complex and has been generally discounted in the industry. Method #3 works, but requires a certificate processing service that is available on the network at all times and incurs the overhead of an online test with each access controlled action call. In effect, this would double the network traffic for all secured actions.

No matter which method of certificate "deletion" one chooses, one must first answer the question:

How long am I willing to let someone else act on information I know to be false?

That length of time is here called the **renewal interval**.

The knee-jerk answer is "zero time", but that is not an option. Even in the OCSP case, it takes time to communicate the result from the validation server to the relying party and during the time the response message is in transit, the server may learn that the message just sent was false. This assumes, of course, that the validation server learns instantly that some certificate is to be revoked. That information, however, is in the head of some human being and that human being may not be in communication with the validation server for some length of time after learning that a certificate needs to be revoked. If the network containing the validation server and the user of that service (the secured device) happens to be partitioned, momentarily, or the validation server is down for some reason, the time #3 takes could be considerable (even days). The online test cannot proceed until both machines are in communication with each other. If we were to design in such an online test, then the home network could become inoperative until the validation server is brought up.

Complicating the choice of renewal interval is the fact that home UPnP devices are not all going to have calendar clocks. Such devices would have to rely on the SetTimeHint action of DeviceSecurity:1 and we cannot predict how many days would elapse between invocations of that action to update the device's concept of the current date and time.

A user should probably choose this renewal interval, but that assumes that the user understands all the implications and can make a proper decision. That decision is a matter for application / GUI designers and not in the scope of this spec. Too small an interval would lead to periods of unavailability of secured devices. Too long an interval would lead to noticeable periods in which a revocation had not taken effect.

---

[2] OCSP: Online Certificate Status Protocol (see the IETF RFC database)

Once that renewal interval is chosen, however, its use is clear.

One issues certificates with a limited lifetime – specifically with a lifetime equal to the renewal interval. This is the maximum length of time that it would take the SC operator to have a change of mind take complete effect. However, it is not desirable to force the SC operator to re-issue a certificate every renewal interval. Therefore, the certificate is issued with the <renew/> element in its <valid> field, indicating that it can be renewed automatically. The SC that issued it would have to be online for the renewal to happen, but a CP is free to ask for renewal before the old certificate expires.

A true computation of renewal interval might require formal risk analysis. It is unlikely that a home user would engage in that, although a manufacturer might and might express the results in a Security Console's code.

3.6. BASE32 Encoding

For display of a Security ID (CP or SC key hash) to the user, in order to minimize confusion, we have chosen BASE32 encoding. A 160-bit hash value is represented as a sequence of 32 5-bit quantities, with the left-most 5-bits being the 5 most significant bits of the 160-bit quantity, etc. The 5-bit quantity is encoded using 32 characters: A..Z, 2..5, 7, 9, in that order, so that 0 becomes "A", 1 becomes "B", 31 becomes "9". The resulting string of letters and numbers will resemble a product registration key, with which the user is expected to be familiar, and omits the digits 0, 1, 6 and 8 which can be confused with O, I, G and B. These can be printed as a sequence of 8 groups of 4 characters each, separated by dashes. In some cases, e.g., in a summary listing of devices or control points, one might use only the left-most group of 4 characters, which should be enough to resolve most ambiguities.

For example, the SHA-1 hash value (in hex):

```
193d9354 ca84f119 d9eec17b c3078c71 8a7ba70c
```

would be (in BASE32):

```
DE7Z-GVGK-QTYR-TWPO-YF54-GB4M-OGFH-XJYM
```

and might be truncated to: `DE7Z` or `DE7Z-GVGK` for resolving ambiguities (e.g., in a list of discovered devices), while the full security ID might be used while verifying the correctness of a control point key.

3.7. XML Strings as UPnP Arguments

The UPnP 1.0 schemas for SOAP as a transport protocol for calling UPnP actions with their respective arguments do not permit arguments that are themselves XML. Some of the security related actions described in this document require the arguments themselves to be XML strings. These XML argument strings are embedded in the surrounding SOAP XML. To ensure that embedded XML argument strings do not "break" the surrounding SOAP XML, it is necessary that the embedded XML is "escaped" as follows:

- The '<' character is encoded as '<'
- The '>' character is encoded as '>'
- The '&' character is encoded as '&'

4. XML Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion> <!-- UPnP version 1.x -->
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>PresentKey</name>
      <argumentList>
        <argument>
          <name>HashAlgorithm</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>Key</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>PreferredName</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>IconDesc</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetNameList</name>
      <argumentList>
        <argument>
          <name>Names</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetMyCertificates</name>
      <argumentList>
        <argument>
          <name>HashAlgorithm</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>Hash</name>
```

```
          <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>Certificates</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>RenewCertificate</name>
      <argumentList>
        <argument>
          <name>OldCertificate</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>NewCertificate</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>PendingCPList</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="yes">
      <name>NameListVersion</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_string</name>
      <dataType>string</dataType>
    </stateVariable>
    <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_base64</name>
      <dataType>bin.base64</dataType>
    </stateVariable>
  </serviceStateTable>
</scpd>
```