UPnP™ Security Ceremonies

Design Document

For UPnP™ Device Architecture 1.0 [1]

Date: October 3, 2003

This design document is being made available to UPnP Members pursuant to Section 2.1(c)(ii) of the UPnP Membership Agreement for review and comment by Members to the UPnP Steering Committee regarding the Steering Committee's consideration of the Proposed template as a Standardized service. Pursuant to Section 3.1 of the UPnP Membership Agreement, Member has limited rights to use or reproduce the Proposed template during the comment period and only in furtherance of this review and comment. All such use is subject to all of the provisions of the UPnP Membership Agreement.

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

Authors	Company
Carl Ellison	Intel Corporation

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

Table of Contents

List of Figures

1 Background

This white paper describes the ceremonies of which UPnP Security consists.

The term **ceremony** was coined by Jesse Walker of Intel Corporation. It refers to something that looks exactly like a network protocol. Network protocols, however, refer to messages among computers – or, more generally, among network nodes. The term "ceremony" refers to messages among computers, people and possibly the environment. However, the resemblance to a network protocol diagram and description is intentional. The same tools that one uses to design and analyze network protocols can be used to design and analyze ceremonies.

Ceremonies were created first for security protocols, like the UPnP Security protocol. This is because every security protocol needs human interactions. In a security protocol, there are decisions to be made (e.g., access control decisions). These decisions are made by network nodes, based on data held in those nodes (possibly augmented by data provided to them) and that data is generally called the **security policy**. That policy can not be built into the equipment. It is formulated in the mind of a human and must be communicated from that human to the network node(s). Therefore, at least for parts of a security protocol that specify security policy, a human is an essential component.

We have discovered over the years that carefully designed security protocols can fail miserably because of flaws in the human behavior or the human-computer interface. See for example the paper by Whitten[1]. Part of the reason for this failure is that the careful security design and analysis is often applied only to the security protocol (among network nodes) and not the full security ceremony.

When one analyzes the behavior of a ceremony, the human components need special attention. One can not assume that the human will behave like a computer. A human user is likely to do incomplete comparisons of values, for example. If there is some human step that is optional, then one can assume the human will not perform it. Some will and some won't, but for the purpose of security analysis, one must assume the worst case.

In UPnP Security, efforts were made to design and analyze the full ceremonies, not just the protocols. This can not be complete without an understanding of the details of the user interface and UPnP does not specify user interfaces. Therefore, the final security analysis of the ceremonies presented in this white paper needs to be done for each implementation.

2 Security Model

In UPnP™, there are **devices** and **control points**. Devices advertise themselves via a discovery protocol and offer services: collections of SOAP actions that the control points invoke.

UPnP Security concerns itself with the control protocol, SOAP. It secures SOAP control messages and their replies. This message security consists of:

1. identification
2. integrity
3. authentication
4. freshness
5. authorization and
6. secrecy

Control message security is depicted in the flowchart of Figure 1, below. Since UPnP specifies protocols rather than program structure, any UPnP implementation can use a different flowchart for its processing, provided the resulting code has the same functional behavior.

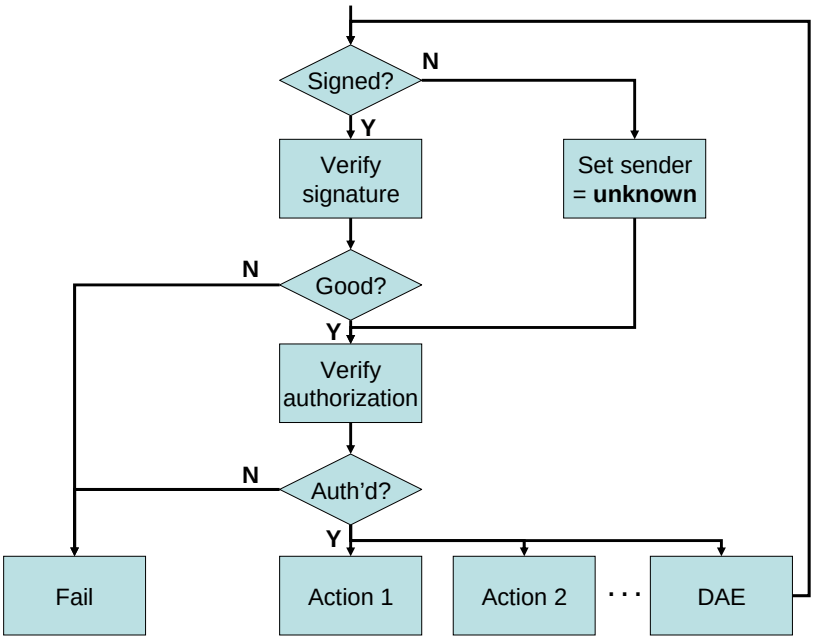

Figure 1: Message Security Flowchart

The sender of a message is identified by its SecurityID (the hash of its public key). An ID=**unknown** can be achieved in a number of ways, e.g., by using a quantity the size of a hash but of value = 0 or = the ASCII "unknown". This is an implementation choice that does not affect any other nodes and is therefore free to make.

If the message is signed by PK methods, then one computes the hash based on the public key that is provided in the <KeyInfo> field. If the message is Session-key signed, then the SecurityID can be available via indirection from the session key ID.

Control message security flow is as follows (referring to Figure 1):

1. A message arrives and is parsed. If there is a <SecurityInfo> element in the SOAP header, then this message is assumed to be signed.
2. If the message is not signed, then set its owner to "unknown" and proceed with testing for authorization. [Some actions are made available to unknown senders, at the discretion of the device working committee, device manufacturer or device owner.]
3. If the message is signed, then the device verifies its signature and its freshness. Verification of the signature verifies both integrity of the message and origin authentication. The sender is indicated by its SecurityID – the hash of its public key.
4. If the signature and freshness are good, then the owner is set to the SecurityID of the signer of the message. Otherwise, the message fails and an error return message is generated.
5. If the message is to be tested for authorization, then authorization information is gathered. This consists of the owner list, ACL, any cached certificates and any certificates provided in the message's <KeyInfo> element. The message's action name is retrieved and used to discover the abstract permission needed by the sender to perform this action. The authorization information of the sender is then tested to determine if that permission has been granted and is still in force.
6. If the sender is not authorized, then the message fails and an error return message is generated. If the sender is authorized, then the desired action is executed.
7. If the desired action is DecryptAndExecute (indicated in Figure 1 as "DAE"), then its ciphertext argument is deciphered and the resulting plaintext is recursively sent back to the start of the

process for processing. Usually, DecryptAndExecute will not be a signed message but the message inside its argument could be.

2.1 Security Policy Data

The message security processing described in Figure 1, above, relies on certain data that is collectively referred to as **security policy**. This data includes:

1. the device's owner list
2. the device's ACL
3. any cached certificates
4. any certificates provided by the sender (control point) as part of the message

This data originates in the mind of a human being (or set of human beings) that UPnP Security refers to as the device **owner**. This data must reside in the device at the time of security decision (Figure 1). It therefore must be communicated from the owner to the device.

To achieve that communication, UPnP Security defines two things:

1. the DeviceSecurity service, which specifies actions to permit the setting of that data
2. the Security Console (SC) – a component that is both a UPnP Device and Control Point, whose function is to provide the user interface for the (a) human owner of a device [The UPnP Device in the SC runs a UPnP service called SecurityConsole.]

An owner is indicated by the hash of the public key of a SC[2]. An **owner list** is a list of such hash values.

A device's **ACL** is a list of entries (possibly empty) that specify the following:

1. Subject: a SecurityID or name of a group of SecurityIDs – indicating the (single, group of) CP or SC to which rights are being granted by this ACL entry
2. Authorization: a list of permission elements, specifying what the Subject is being granted by this entry
3. May-not-delegate: an optional flag indicating that the Subject may not delegate any of the rights granted in this entry on to other CPs, SCs or groups
4. Validity: an optional set of elements that could include a not-before date/time and/or a not-after date/time, limiting the period of validity of the entry

There are two kinds of **certificate** defined in UPnP Security: **named group membership** and **authorization**.

A **named group membership certificate** contains the following entries:

1. Issuer: the SecurityID of the issuing SC
2. Name: the textual name of the group
3. Subject: a SecurityID or name (as in an ACL entry) of the individual key or group of keys being given membership in this group
4. Validity: an optional list of elements that could include a not-before date/time, a not-after date/time and/or a <Renew/> element (noting that this certificate was issued to be valid for an artificially short interval of time and can be renewed by sending it to the Issuer).

An **authorization certificate** has the same elements as an ACL entry, but also an Issuer field and all the same Validity options as the name certificate. The authorization field is slightly different because it must indicate the device(s) to which the grant of authorization applies while the ACL can apply only to the one device in which it is resident.

[2] The phrase "the public key of a component" is shorthand for "the public key that corresponds to the private key unique to and kept protected within that component and used to sign or decrypt messages". In UPnP Security, Control Points and Security Consoles have signature private keys while Devices have decryption private keys.

3 Secure Component Discovery

In order to grant rights on some Device to a CP or SC, those components must first be discovered by the device owner's SC.

UPnP Discovery is not symmetric. That is, Control Points discover Devices, but not the other way around. UPnP Security, however, needs to discover both Devices and Control Points. Therefore, actions are defined for this purpose.

3.1 Discovery of Secured Devices

A secured Device can be discovered in the normal UPnP way. A secured device can be distinguished from an unsecured device because the secured device will include a DeviceSecurity service.

This normal discovery process via SSDP is not secure, however. To secure the discovery process there is a ceremony as shown in Figure 2, below.

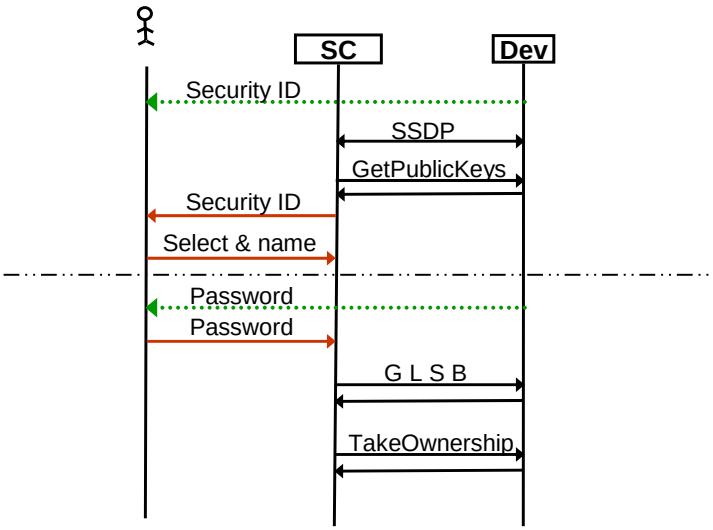

Figure 2: Device discovery ceremony and TakeOwnership

The discovery of security-enabled devices is more than the basic discovery that is performed by UPnP's SSDP. The purpose of this discovery process is to make sure that the user's SC (acting as the user's window into the security administration of the UPnP network) associates with the correct device(s) and that each device of the user's network associates with the correct SC. The security of the UPnP protocol is assured by industry standard practices, but it does no good to have a secure protocol between the wrong elements. Secure discovery addresses that latter problem.

With reference to Figure 2, the process for secure discovery is:

1. The user reads the target device's SecurityID from the device (from a label or display on the device or from a card shipped with the device).
2. The SC discovers the target device (possibly among others) via SSDP.
3. For each device offering DeviceSecurity, the SC gets the device's public key by a call to GetPublicKeys.
4. The SC computes each device's SecurityID from its public key and displays that to the user, for comparison with the SecurityID obtained in step 1.

5. The user selects the target device from a list of available devices (those devices with public keys not already named by that user at that SC) and names it. The SC then remembers that name for that device and no longer displays it as an available device.

If, in addition to discovering and naming the security-aware device, the user wants to take security ownership of that device, the process continues with:

6. The user reads the target device's initial password from the device (from a label or display on the device or from a card shipped with the device).
7. The user provides that password to the SC, which uses it to compute the values needed for a TakeOwnership call.
8. The SC executes the GetLifetimeSequenceBase call, to get the current LifetimeSequenceBase value to use in computing the arguments for the TakeOwnership call.
9. The SC executes the TakeOwnership call. [See section 4.1 for more details about TakeOwnership and its options.]

3.1.1 How to read the ceremony diagram

Figure 2 is a good sample of a full ceremony diagram.

Every time there is a connection between a computer component and a human user, one can not use a standard network layer. Humans don't have network interfaces.

In the figure, there are three different colors for "network layers":

- Black: a normal network connection
- Red: a normal user interface
- Green dotted: a static user interface (reading a label or a print-out, maybe reading a display on the device, but copying down the information to bring over to another display (the one on the SC) for use there)

3.2 Discovery of a Secured CP or SC

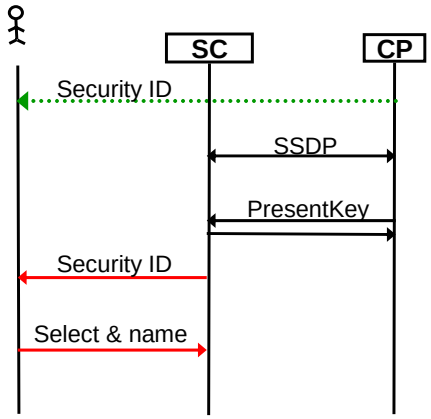

Figure 3: Discovery of CP and SC nodes

UPnP does not define a way to discover control points (CP). However, for security purposes, we must discover security-aware CPs (and other SCs, which act like CPs in this context). To remedy this lack, the SC is also a UPnP device that a security-aware CP discovers in the normal way (by SSDP). The full process is:

1. The user reads the Security ID from the target CP or SC (from a label or display on the CP or from a card shipped with the CP).
2. The CP discovers the user's SC (and any others) via SSDP.
3. The CP offers its public key to each SC it finds, via the PresentKey action.
4. The SC computes a Security ID from the CP's public key and displays that to the user.
5. The user compares the Security ID computed by the SC to the one obtained in step 1, selects the appropriate CP (or other SC), and names it. The user's SC remembers this name and no longer displays that CP as an unknown device in need of a name, even when it offers its public key again in the future via PresentKey.

4 Ownership

An SC is said to **own** a device if the SC's public key is listed in the device's owner list. An SC that owns a device is permitted to do everything on that device, including manipulate the owner list and the device's ACL.

DeviceSecurity defines five actions that allow manipulation of the device's owner list:

1. TakeOwnership
2. ListOwners
3. GrantOwnership
4. RevokeOwnership

5. FactorySecurityReset

4.1 TakeOwnership

TakeOwnership is available only if the device is unowned. Once a device has a non-empty owner list, TakeOwnership is no longer honored and instead owners must be added via GrantOwnership (section 4.3).

The default TakeOwnership process is described in section 3.1, above.

Alternate methods of taking ownership could be more convenient, if the manufacturer wants to provide extra hardware for that purpose. These methods have not been standardized as part of UPnP, but remain available for manufacturers to provide, if desired, in addition to the standardized mechanism.

For example, a device and SC could each have USB connectors and TakeOwnership could occur over a point-to-point cable dedicated to this function, as shown in Figure 4, below. The security of this method depends on the cable's connecting exactly two entities – the device and the SC.

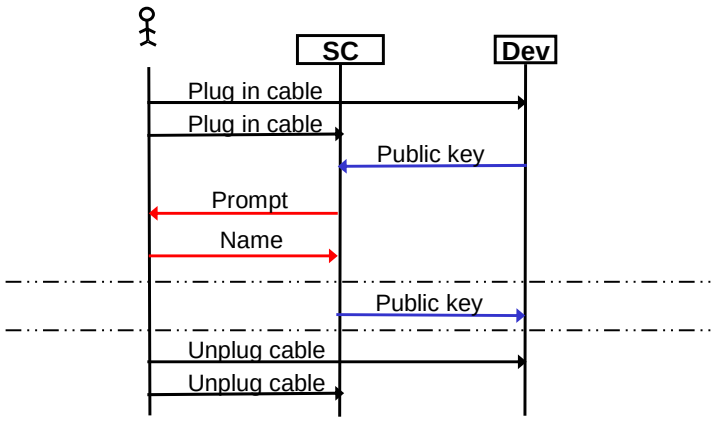

Figure 4: Taking ownership via private cable

This process has the following steps:

1. The user plugs a cable into both the device and the SC. This might be, for example, a USB cable with the device showing up on the PC that is running the SC as any USB device would.
2. Over that private cable, the device delivers its public key to the SC.
3. The user is prompted for a name for the device and a decision of whether to take ownership.
4. The user provides a name for that device.
5. If the user also wanted to take ownership, then the SC provides its public key to the device for immediate insertion into the owner list.
6. In either case, the user then unplugs the cable, since device discovery, naming and possibly ownership have been established. From then on, the device is accessed over the network.

4.1.1 Note on Figure 4

Figure 4 uses a color code, as the one exception to the general expectation set in section 3.1.1. The connection between SC and Dev in this case is not over the normal network link, but rather over a private point-to-point cable. That link is colored blue in the figure.

4.2 ListOwners

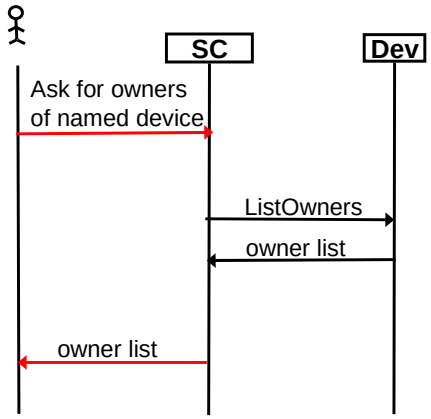

Figure 5: ListOwners

The ListOwners action is invoked by the SC as the user's client and agent. The flow is as shown in Figure 5 and assumes that the device whose owners are being listed is already known to the user, having been discovered and named, as shown in Figure 2.

It is assumed that the SC will display the owner list to the user using names that the user has already assigned to the various SCs that are in the owner list, assuming these have been discovered and named (as per Figure 3). If there are owners listed in the owner list that have not been named, the SC can display those as raw SecurityIDs.

4.3 GrantOwnership

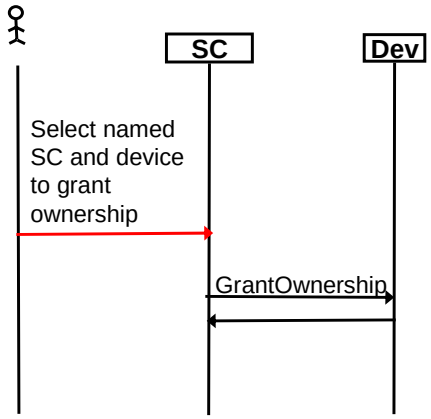

Figure 6: GrantOwnership

Having named some other SC (see Figure 3) and some target device (see Figure 2) of which the user's SC is an owner, the user selects that other SC and device for a grant of ownership and then the user's SC performs a GrantOwnership action.

4.4 RevokeOwnership

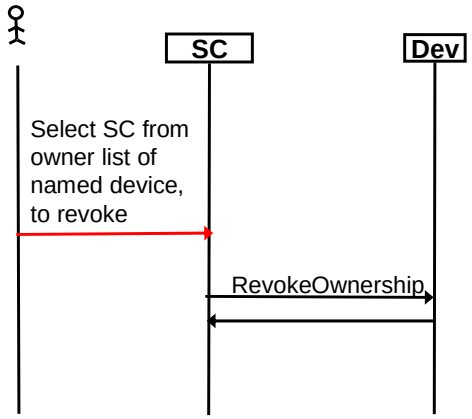

Figure 7: RevokeOwnership

From a display of the owner list of a device of which the user's SC has ownership, the user selects some SC whose ownership is to be revoked. That SC can be listed by name or SecurityID. The RevokeOwnership action will not allow the user's SC to revoke its own ownership.

4.5 FactorySecurityReset

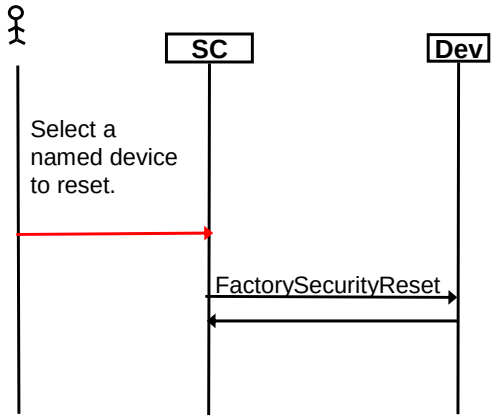

Figure 8: FactorySecurityReset

FactorySecurityReset can be performed only by an owner of the device. It returns the device to unowned state, with all security policy erased. Because this could result in an annoying loss of information, if applied inappropriately, the user's SC may ask for extra confirmation before permitting this action. However, from a security point of view, FactorySecurityReset is relatively innocuous.

5 Session Keys

For all secured actions on a device, except TakeOwnership and SetSessionKeys, the SC (or a CP) needs session keys. Session keys are acquired with a straight protocol, as shown in Figure 9, below.

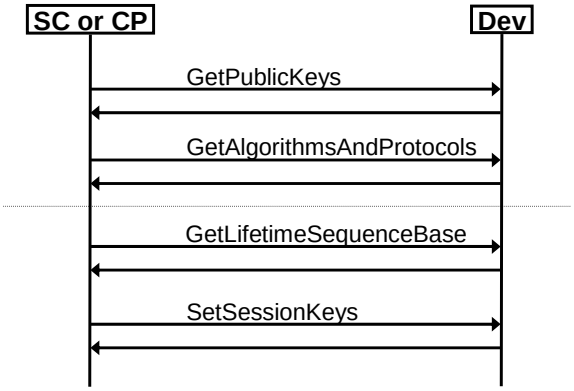

Figure 9: Setting Session Keys

The protocol for setting session keys consists of 4 steps, the first two of which gather data that is relatively constant and can therefore be cached.

1. The CP gets the device's public key.
2. The CP gets the device's list of supported algorithms and protocols. There may be optional protocols offered by the device that the CP wishes to use and for which session keys must be established.
3. The CP gets the device's LifetimeSequenceBase. This value changes over the life of the device and is used to provide freshness for any public-key-authenticated actions (TakeOwnership and SetSessionKeys).
4. The CP generates session keys that it wants to use and sends those to the device, encrypted using the device's public key, in a SetSessionKeys action call.

6 ACL Editing

The main purpose of security ownership (see section 4) is to establish who is permitted to edit a device's ACL. The ACL is the root of the device's security policy and must be established by a human user (the security owner of the device).

The owner list is a special case ACL – in which all entries are ACL subjects who receive all permissions on the device.

A normal ACL grants some set of permissions on a device, typically not total permissions, to a selected CP, SC or named group of those.

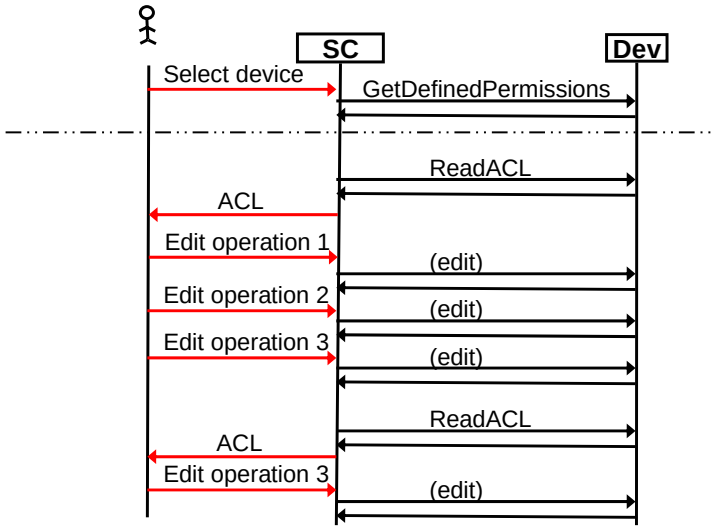

Figure 10: ACL Editing

ACL editing is a user-driven operation, so we can not prescribe a full ceremony for it. Figure 10 gives a sample of what an involved editing session might entail. This ceremony assumes that the SC has already established session keys with the device. It also assumes that the user has named all CPs or SCs to which it cares to grant access on this device and that any named CP or SC groups have been created already.

1. This session starts with the user selecting a device by name. The device selected must be one this SC already owns.
2. The SC asks the device for its set of defined permissions (so that the SC can give the user tool tips and other documentation about the permissions). This information is cacheable, so the SC is not required to ask for it every time.
3. The SC must read the device's ACL. An ACL might be cached but if there is more than one device owner, then the ACL could have been changed by some other SC. It is therefore safest to read the ACL before an editing session. If the SC is going to add entries to the ACL, it may also need to call GetACLSizes, to make sure that there is room for the ACL to grow.
4. The SC presents the ACL to the user.
5. The user gives an editing operation (1).
6. The SC performs that edit operation via one of the editing actions: WriteACL, DeleteACLEntry, ReplaceACLEntry or AddACLEntry.
7. The user gives a second editing operation (2).
8. The SC performs it.
9. The user gives a third editing operation (3).
10. The SC attempts to perform it, but gets an error message that the ACLVersion is out of date.
11. The SC re-reads the ACL, because it was changed from what the SC had been dead-reckoning through the editing commands.
12. The SC presents the updated ACL to the user.
13. The user gives the third editing operation (3) again.
14. The SC performs that third editing operation.

7 Certificate caching

There are two purposes for a certificate in UPnP Security:

1. to define named groups of control points (or security consoles)
2. to grant an authorization to some CP, SC or named group of those when it is not possible to edit the ACL, because:
    a. there is too little room in the ACL for a new entry, or
    b. the SC does not have permission to edit the ACL

To be effective, certificates must be at the device on which they grant authorization. They can be delivered either via the device's CacheCertificate action or in the <KeyInfo> field of the XML-Signature of the incoming message to which they are to apply.

Certificate caching on the device is a straight-forward protocol, as shown in Figure 11, below.

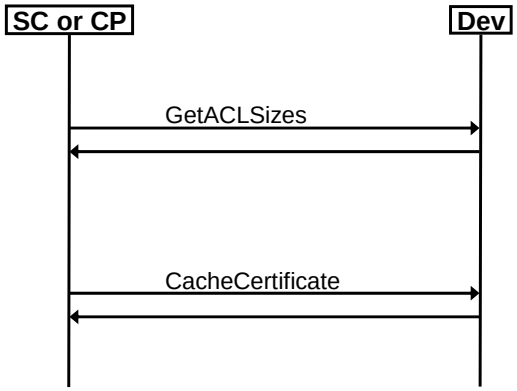

Figure 11: Certificate caching on the device

If the certificate is delivered from the CP, in the message or by caching, the CP must get it from the SC that generated it. The CP does not offer any services, so it can not offer a CacheCertificate action. Instead, the CP pulls certificate from the SC, as shown in Figure 12, below.

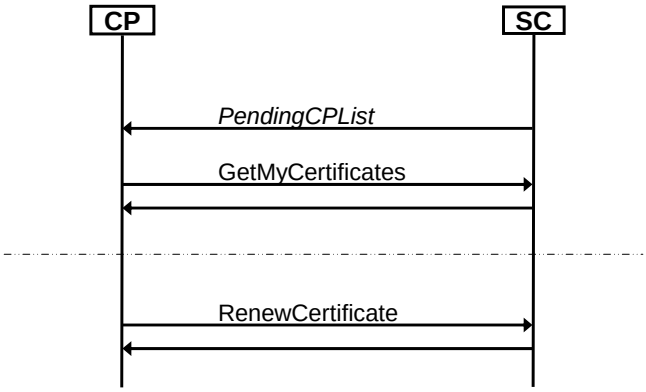

Figure 12: Delivering certificates to the CP

A control point that is capable of holding certificates gets those from any security console that happens to generate one. These grants of authority or group memberships can come from any SC, not just a single one, so the CP must look for these certificates from every SC it can see on the network.

The CP could poll for certificates by periodically issuing GetMyCertificates calls to all these SCs. However, in the interest of efficiency, each SC offers an evented variable, PendingCPList, that lists the ID of each CP that has certificates waiting to be fetched. Assuming the CP makes use of that variable, the process for getting certificates, as shown in Figure 12, is:

1. The CP receives a copy of PendingCPList from some SC. It scans that list, looking for its own ID. If it finds it, then it proceeds with this process.
2. The CP calls the GetMyCertificates action, getting the set of all certificate the SC has available for that CP.
3. At some time in the future, the CP may notice that it has a cached certificate that is renewable and is due to expire soon. In that case, assuming the SC capable of renewing it is online, the CP sends the expiring certificate body to the SC in a RenewCertificate action and gets a newly signed certificate with new expiration date in return – assuming the human owner of that SC has not decided to revoke that certificate. If the human user has revoked the certificate, the SC will return an error in response to the attempt to renew that certificate.

8 INDEX

9 REFERENCES

[1] Alma Whitten and J.D. Tygar, “Why Johnny Can’t Encrypt”, USENIX Security Symposium. http://www-2.cs.cmu.edu/~alma/johnny.pdf