ISO/IEC 29341-13-10

Edition 1.0 2008-11

INTERNATIONAL STANDARD

Information technology – UPnP Device Architecture – Part 13-10: Device Security Device Control Protocol – Device Security Service



ISO/IEC 29341-13-10:2008(E)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2008 ISO/IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about ISO/IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub

The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ISO/IEC 29341-13-10

Edition 1.0 2008-11

INTERNATIONAL STANDARD

Information technology – UPnP Device Architecture – Part 13-10: Device Security Device Control Protocol – Device Security Service

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE X

ICS 35.200

ISBN 2-8318-1012-8

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

CONTENTS

FOREWORD 5
ORIGINAL UPNP DOCUMENTS (informative) 7
1. Overview and Scope 9
1.1. Acknowledgements 11
2. Service Modeling Definitions 12
2.1. Service Type 12
2.2. Namespaces 12
2.3. Referenced Specifications 12
2.4. MustUnderstand 12
2.5. State Variables 13
2.5.1. NumberOfOwners 13
2.5.2. LifetimeSequenceBase 13
2.5.3. TimeHint 14
2.5.4. TotalACLSize 14
2.5.5. FreeACLSize 14
2.5.6. TotalOwnerListSize 14
2.5.7. FreeOwnerListSize 14
2.5.8. TotalCertCacheSize 14
2.5.9. FreeCertCacheSize 14
2.5.10. A_ARG_TYPE_string 14
2.5.11. A_ARG_TYPE_base64 14
2.5.12. A_ARG_TYPE_int 14
2.5.13. A_ARG_TYPE_boolean 15
2.6. Eventing and Moderation 15
2.7. Actions 16
2.8. Cryptographic Notation for Selected Actions 17
2.9. Actions Invoked by Both CP and SC 17
2.9.1. GetPublicKeys 17
2.9.2. GetAlgorithmsAndProtocols 18
2.9.3. GetACLSizes 19
2.9.4. CacheCertificate 20
2.9.5. SetTimeHint 22
2.9.6. GetLifetimeSequenceBase 23
2.9.7. SetSessionKeys 24
2.9.8. ExpireSessionKeys 26
2.9.9. DecryptAndExecute 27
2.10. Actions Invoked by SC only 28
2.10.1. TakeOwnership 28
2.10.2. GetDefinedPermissions 30
2.10.3. GetDefinedProfiles 31
2.10.4. ReadACL 33
2.10.5. WriteACL 34
2.10.6. AddACLEntry 35
2.10.7. DeleteACLEntry 36
2.10.8. ReplaceACLEntry 37
2.10.9. FactorySecurityReset 38
2.10.10. GrantOwnership 39
2.10.11. RevokeOwnership 40
2.10.12. ListOwners 41
2.11. Relationships among Actions 43

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.11.1. Relationships among Actions invoked by Security Console 43
2.11.2. Relationships among Actions invoked by normal Control Point 43
2.11.3. ACLVersion 44
2.12. Common Error Codes 45
3. Supporting Information 46
3.1. Glossary 46
3.2. XML Strings as UPnP Arguments 46
3.3. BASE32 Encoding 47
3.4. Namespaces 47
4. Data Structures 48
4.1. Namespaces 48
4.2. Access Control List (ACL) Structure 48
4.2.1. Note on date and time format: ISO 8601 49
4.3. Owner List 49
4.4. Certificates 50
4.4.1. Authorization Certificate 50
4.4.2. Name Definition Certificate 51
4.5. Permission Language 52
4.5.1. <all> 52
4.5.2. <set> 52
4.5.3. <elt> 52
4.5.4. <prefix> 52
4.5.5. <range> 52
4.6. RSA Encryption Padding 53
4.6.1. SetSessionKeys 54
4.6.2. TakeOwnership 54
4.6.3. Counteracting attacks on PKCS#1 V 1.5 padding 54
4.6.4. Historical note about padding and padding attacks 55
4.7. Public Keys and their hashes 55
4.8. Symmetric cipher mode and padding 56
4.9. Canonical BASE64 Encoding 56
5. Theory of Operation 58
5.1. Access Control Lists and Certificates 58
5.1.1. ACL and Certificate Processing Model 59
5.2. Signature block format 59
5.2.1. Sequence Numbering 61
5.2.2. Hashing and Canonicalization 62
5.2.3. UPnP Certificate Transport 62
5.2.4. IDs for XML-Signature 63
5.2.5. Signature Processing Model 63
6. XML Service Description 64
Annex A (normative) **Device Security Schema 71**
Annex B (informative) **Security Ceremonies 79**
B.1 Background 79
B.2 Security Model 79
B.2.1 Security Policy Data 81
B.3 Secure Component Discovery 81
B.3.1 Discovery of Secured Devices 82
B.3.2 Discovery of a Secured CP or SC 84

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.4 Ownership 84
B.4.1 TakeOwnership 85
B.4.2 ListOwners 86
B.4.3 GrantOwnership 87
B.4.4 RevokeOwnership 87
B.4.5 FactorySecurityReset 88
B.5 Session Keys 88
B.6 ACL Editing 89
B.7 Certificate caching 90
B.8 References 92

LIST OF TABLES

Table 1: State variable 13
Table 2: Event Moderation 15
Table 3: Actions invoked by both Control Point and Security Console 16
Table 4: Actions invoked by a Security Console only 16

LIST OF FIGURES

Figure B.1: Message Security Flowchart 80
Figure B.2: Device discovery ceremony and TakeOwnership 82
Figure B.3: Discovery of CP and SC nodes 84
Figure B.4: Taking ownership via private cable 85
Figure B.5: ListOwners 86
Figure B.6: GrantOwnership 87
Figure B.7: RevokeOwnership 87
Figure B.8: FactorySecurityReset 88
Figure B.9: Setting Session Keys 89
Figure B.10: ACL Editing 90
Figure B.11: Certificate caching on the device 91
Figure B.12: Delivering certificates to the CP 92

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

INFORMATION TECHNOLOGY – UPNP DEVICE ARCHITECTURE –

Part 13-10: Device Security Device Control Protocol – Device Security Service

FOREWORD

1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.

4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.

6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.

7) All users should ensure that they have the latest edition of this publication.

8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.

9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

IEC and ISO draw attention to the fact that it is claimed that compliance with this document may involve the use of patents as indicated below.

ISO and IEC take no position concerning the evidence, validity and scope of the putative patent rights. The holders of the putative patent rights have assured IEC and ISO that they are willing to negotiate free licences or licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of the putative patent rights are registered with IEC and ISO.

Intel Corporation has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Intel Corporation
Standards Licensing Department
5200 NE Elam Young Parkway
MS: JFS-98
USA – Hillsboro, Oregon 97124

Microsoft Corporation has informed IEC and ISO that it has patent applications or granted patents as listed below:

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6101499 / US; 6687755 / US; 6910068 / US; 7130895 / US; 6725281 / US; 7089307 / US; 7069312 / US; 10/783 524 /US

Information may be obtained from:

Microsoft Corporation
One Microsoft Way
USA – Redmond WA 98052

Philips International B.V. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Philips International B.V. – IP&S
High Tech campus, building 44 3A21
NL – 5656 Eindhoven

NXP B.V. (NL) has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

NXP B.V. (NL)
High Tech campus 60
NL – 5656 AG Eindhoven

Matsushita Electric Industrial Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Matsushita Electric Industrial Co. Ltd.
1-3-7 Shiromi, Chuoh-ku
JP – Osaka 540-6139

Hewlett Packard Company has informed IEC and ISO that it has patent applications or granted patents as listed below:

5 956 487 / US; 6 170 007 / US; 6 139 177 / US; 6 529 936 / US; 6 470 339 / US; 6 571 388 / US; 6 205 466 / US

Information may be obtained from:

Hewlett Packard Company
1501 Page Mill Road
USA – Palo Alto, CA 94304

Samsung Electronics Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Digital Media Business, Samsung Electronics Co. Ltd.
416 Maetan-3 Dong, Yeongtang-Gu,
KR – Suwon City 443-742

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC and ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29341-13-10 was prepared by UPnP Implementers Corporation and adopted, under the PAS procedure, by joint technical committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

The list of all currently available parts of the ISO/IEC 29341 series, under the general title *Universal plug and play (UPnP) architecture*, can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies, and the voting results may be obtained from the address given on the second title page.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ORIGINAL UPNP DOCUMENTS
(informative)

Reference may be made in this document to original UPnP documents. These references are retained in order to maintain consistency between the specifications as published by ISO/IEC and by UPnP Implementers Corporation. The following table indicates the original UPnP document titles and the corresponding part of ISO/IEC 29341:

UPnP Document Title	**ISO/IEC 29341 Part**
UPnP Device Architecture 1.0	ISO/IEC 29341-1
UPnP Basic:1 Device	ISO/IEC 29341-2
UPnP AV Architecture:1	ISO/IEC 29341-3-1
UPnP MediaRenderer:1 Device	ISO/IEC 29341-3-2
UPnP MediaServer:1 Device	ISO/IEC 29341-3-3
UPnP AVTransport:1 Service	ISO/IEC 29341-3-10
UPnP ConnectionManager:1 Service	ISO/IEC 29341-3-11
UPnP ContentDirectory:1 Service	ISO/IEC 29341-3-12
UPnP RenderingControl:1 Service	ISO/IEC 29341-3-13
UPnP MediaRenderer:2 Device	ISO/IEC 29341-4-2
UPnP MediaServer:2 Device	ISO/IEC 29341-4-3
UPnP AV Datastructure Template:1	ISO/IEC 29341-4-4
UPnP AVTransport:2 Service	ISO/IEC 29341-4-10
UPnP ConnectionManager:2 Service	ISO/IEC 29341-4-11
UPnP ContentDirectory:2 Service	ISO/IEC 29341-4-12
UPnP RenderingControl:2 Service	ISO/IEC 29341-4-13
UPnP ScheduledRecording:1	ISO/IEC 29341-4-14
UPnP DigitalSecurityCamera:1 Device	ISO/IEC 29341-5-1
UPnP DigitalSecurityCameraMotionImage:1 Service	ISO/IEC 29341-5-10
UPnP DigitalSecurityCameraSettings:1 Service	ISO/IEC 29341-5-11
UPnP DigitalSecurityCameraStillImage:1 Service	ISO/IEC 29341-5-12
UPnP HVAC_System:1 Device	ISO/IEC 29341-6-1
UPnP HVAC_ZoneThermostat:1 Device	ISO/IEC 29341-6-2
UPnP ControlValve:1 Service	ISO/IEC 29341-6-10
UPnP HVAC_FanOperatingMode:1 Service	ISO/IEC 29341-6-11
UPnP FanSpeed:1 Service	ISO/IEC 29341-6-12
UPnP HouseStatus:1 Service	ISO/IEC 29341-6-13
UPnP HVAC_SetpointSchedule:1 Service	ISO/IEC 29341-6-14
UPnP TemperatureSensor:1 Service	ISO/IEC 29341-6-15
UPnP TemperatureSetpoint:1 Service	ISO/IEC 29341-6-16
UPnP HVAC_UserOperatingMode:1 Service	ISO/IEC 29341-6-17
UPnP BinaryLight:1 Device	ISO/IEC 29341-7-1
UPnP DimmableLight:1 Device	ISO/IEC 29341-7-2
UPnP Dimming:1 Service	ISO/IEC 29341-7-10
UPnP SwitchPower:1 Service	ISO/IEC 29341-7-11
UPnP InternetGatewayDevice:1 Device	ISO/IEC 29341-8-1
UPnP LANDevice:1 Device	ISO/IEC 29341-8-2
UPnP WANDevice:1 Device	ISO/IEC 29341-8-3
UPnP WANConnectionDevice:1 Device	ISO/IEC 29341-8-4
UPnP WLANAccessPointDevice:1 Device	ISO/IEC 29341-8-5
UPnP LANHostConfigManagement:1 Service	ISO/IEC 29341-8-10
UPnP Layer3Forwarding:1 Service	ISO/IEC 29341-8-11
UPnP LinkAuthentication:1 Service	ISO/IEC 29341-8-12
UPnP RadiusClient:1 Service	ISO/IEC 29341-8-13
UPnP WANCableLinkConfig:1 Service	ISO/IEC 29341-8-14
UPnP WANCommonInterfaceConfig:1 Service	ISO/IEC 29341-8-15
UPnP WANDSLLinkConfig:1 Service	ISO/IEC 29341-8-16
UPnP WANEthernetLinkConfig:1 Service	ISO/IEC 29341-8-17
UPnP WANIPConnection:1 Service	ISO/IEC 29341-8-18
UPnP WANPOTSLinkConfig:1 Service	ISO/IEC 29341-8-19
UPnP WANPPPConnection:1 Service	ISO/IEC 29341-8-20
UPnP WLANConfiguration:1 Service	ISO/IEC 29341-8-21
UPnP Printer:1 Device	ISO/IEC 29341-9-1
UPnP Scanner:1.0 Device	ISO/IEC 29341-9-2
UPnP ExternalActivity:1 Service	ISO/IEC 29341-9-10
UPnP Feeder:1.0 Service	ISO/IEC 29341-9-11
UPnP PrintBasic:1 Service	ISO/IEC 29341-9-12
UPnP Scan:1 Service	ISO/IEC 29341-9-13
UPnP QoS Architecture:1.0	ISO/IEC 29341-10-1
UPnP QosDevice:1 Service	ISO/IEC 29341-10-10
UPnP QosManager:1 Service	ISO/IEC 29341-10-11
UPnP QosPolicyHolder:1 Service	ISO/IEC 29341-10-12
UPnP QoS Architecture:2	ISO/IEC 29341-11-1
UPnP QOS v2 Schema Files	ISO/IEC 29341-11-2

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

UPnP Document Title	ISO/IEC 29341 Part
UPnP QosDevice:2 Service	ISO/IEC 29341-11-10
UPnP QosManager:2 Service	ISO/IEC 29341-11-11
UPnP QosPolicyHolder:2 Service	ISO/IEC 29341-11-12
UPnP RemoteUIClientDevice:1 Device	ISO/IEC 29341-12-1
UPnP RemoteUIServerDevice:1 Device	ISO/IEC 29341-12-2
UPnP RemoteUIClient:1 Service	ISO/IEC 29341-12-10
UPnP RemoteUIServer:1 Service	ISO/IEC 29341-12-11
UPnP DeviceSecurity:1 Service	ISO/IEC 29341-13-10
UPnP SecurityConsole:1 Service	ISO/IEC 29341-13-11

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

1. Overview and Scope

The Device Security service provides the services necessary for strong authentication, authorization, replay prevention and privacy of UPnP SOAP actions. Under this architecture, a Device enforces its own access control but its access control policy is established and maintained by an administrative application, the Security Console (see SecurityConsole:1), which uses some of the actions provided as part of this service. Nothing prevents a device with the proper user interface capabilities from providing its own administration interface, although presumably it is a valuable ability to administer access from one location for an entire household network. In what follows, the term "Security Console" refers to any Control Point that chooses to exercise the administrative functions defined here in DeviceSecurity.

DeviceSecurity implements access control for itself and for other Services in the same Device (or embedded Device). There are two classes of access control grant defined here: ownership and normal permission. Each security-aware Device has an ownership list capable of holding at least one entry. Any Security Console listed as an owner has full rights to the Device, specifically to all actions including the DeviceSecurity actions that specify other access control. In addition to the owner list, the Device usually has an access control list (ACL) maintained by DeviceSecurity. Entries in the ACL grant a Security Console or other Control Point symbolic permissions that, in turn, grant access to sets of actions. Those permissions typically grant less than the full access that ownership grants. A Security Console, at the option of the Device vendor, might also be granted the permission to delegate rights to others without having to be a full owner of the device or to define named groups of Control Points to be granted access as a group in a single operation. These last two capabilities depend on the implementation of certificate processing by the Device and that is an optional feature of DeviceSecurity.

A device implementing DeviceSecurity will need to support the basic cryptographic algorithms used in this service:

1. AES 128-bit, for symmetric bulk encryption, labeled "AES-128-CBC", with blocks padded as described in section 4.8 below.

2. SHA1 HMAC, for symmetric signatures and for TakeOwnership, labeled "SHA1-HMAC"

3. RSA 1024-bit, for identification and for establishing secure sessions, and possibly for other operations, labeled "RSA". In the current version of this DeviceSecurity specification, there is no reason for a device to have a public-key algorithm signature key. Therefore a device will offer a public confidentiality key but does not need to offer a signature key at this time. [In particular, actions that are authorized by public-key algorithm signatures do not have digitally signed responses.] The algorithm identifier "RSA" when used for encryption in this service implies (RSA, PKCS#1) and when used for signing implies (RSA, SHA-1, PKCS#1).

Other algorithms may be added to this list, if these develop flaws and need to be supplanted. For example, if another hash algorithm were to be added, one would need to identify it and also identify RSA using it for signing. So, for example, if one adds the hash algorithm FOOBLA, one would need to add RSA-FOOBLA and FOOBLA-HMAC to the algorithm list.

The logic of operation, from the point of view of the security-aware device, is as follows:

1. If a device has a display or printing capability and also has a source of randomness, then it is preferable for the device to generate a new password and new public key pair on any power-up when it is in factory reset state. It would then display or print the generated password and the hash of the new public key. For devices without those abilities, the password and public key pair will have been created by the manufacturer and will remain constant over the lifetime of the device. In that case, the manufacturer will have printed the password and the Security ID (hash of the device's public key) on a card or a label attached to the device case, or both. The password does not need to be longer than about 6 upper case alphanumeric characters, provided it was generated randomly for that device and is used by TakeOwnership shortly after the device is plugged into the network. [Note: passwords or keys are not to be used in common across a set of devices. Such usage would be a security flaw.]

2. The device announces itself via SSDP, perhaps giving real details or perhaps describing itself only as "Security Aware Device", if the device wants to prevent inventory by an unauthorized control point.

3. A Security Console (SC), presumably that of the device's owner, calls the GetPublicKeys, GetLifetimeSequenceBase and TakeOwnership actions, supplying the device's password to prove

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

authorization to take control of the device. As a result of a successful TakeOwnership action, that Security Console is listed as the device's Owner. An Owner is a control point that is empowered to edit the device's Access Control List (ACL). Subsequent TakeOwnership attempts MUST be ignored. If the device generated its password dynamically, then the password used in this TakeOwnership action should not be valid again.

4. The owning SC establishes a set of session keys by calling GetLifetimeSequenceBase and SetSessionKeys. These session keys will be used for digitally signing future action messages using XML-Signature with symmetric key signature.

5. The owning SC will then be free to use the GetACLSizes action to discover whether access permissions can be granted by ACL entries directly or need to be encoded as certificates, because the device has too little room to store an ACL.

6. The owning SC will probably also invoke GetDefinedPermissions in order to learn what permissions it might grant to chosen Control Points.

7. If the SC grants access by certificate, that operation happens without invoking any actions on the device. Otherwise, the SC grants access to desired Control Points (CPs) by way of AddACLEntry. It also reads the current ACL contents, for display to its human operator, by using ReadACL. The actions WriteACL, ReplaceACLEntry and DeleteACLEntry are also available for ACL editing. These actions will be omitted from DeviceSecurity in those implementations that do not provide any memory for an ACL.

8. A CP that wants to conduct a control session with a security aware device may call GetAlgorithmsAndProtocols, in order to confirm interoperability, and will then call SetSessionKeys, to establish a secure session with the device. These sessions do not hold network connections open and could be very long-lived, depending on the storage capacity of the Device and the CP.

9. Once a CP has established a session with the device, it invokes actions by sending normal action messages, digitally signed using XML-Signature with a symmetric signature key (e.g., HMAC) established during SetSessionKeys and with a sequence number initialized by that action. The sequence numbers during a session must be monotonically increasing but need not be sequential.

10. If a CP needs an action message to be confidential, it uses the DecryptAndExecute action, one argument of which is the ciphertext of an encrypted action. The reply from that action is then encrypted and returned in the reply to the DecryptAndExecute action.

11. A session may be ended intentionally, by ExpireSessionKeys, or may time out at the device's discretion.

12. In the event that the device's owner wants to share ownership, either with another person or (for fault-tolerance) with another SC operated by him- or her-self, a current owning SC can invoke GrantOwnership. This assumes that GetACLSizes shows that there is room in the device to record the additional owner. Ownership can be revoked via RevokeOwnership. Current owners can be listed via the action ListOwners.

13. Should a device be sold to someone else, it can be reinitialized via the FactorySecurityReset action. It is strongly recommended that a device also include some physical means for achieving the same end, although that means should not necessarily be convenient (e.g., might require opening the case). The physical Reset mechanism is to cover the case when a Security Console takes ownership of the device, has not granted any access or any co-ownership, and then dies irretrievably, leaving the device owned by an SC that can never again be used.

14. It is conceivable that a device manufacturer might want to define some UPnP maintenance actions to which it alone retains authority. To do this under UPnP DeviceSecurity, the manufacturer needs to create a separate device with its own instance of DeviceSecurity and take ownership of that sub-device at time of manufacture. Simply initializing the normal ACL with access permissions granted to the manufacturer's key(s) does not allow the manufacturer to retain control, since a device owner can delete any entries in the normal ACL. This separate sub-device would not have its ACL affected by FactorySecurityReset.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Enhancements

15. A device may, if it chooses, define named sets of permissions, called Profiles, and a Security Console may read those definitions via GetDefinedProfiles. These Profiles could be role names, for example, like Parent, Child, or Administrator. The Profile names can be used in the user interface provided by the Security Console.

16. If a CP has certificates to present to the device, in order to gain access, and if the device shows via GetACLSizes that it has certificate cache memory available, the CP may send those certificates to the device via the CacheCertificate action. Such certificates would then be available on the device over multiple subsequent actions. Implementation of certificate caches is up to the device, but it would make sense to validate certificates at the time they are cached. It might also make sense to derive implied ACL entries from validated certificates and current ACL entries, and store those derived entries at the time a certificate is cached.

1.1. Acknowledgements

The authors and the chair of the UPnP Security Working Committee would like to acknowledge significant contributions made by other UPnP members to help complete this work. Markus Wischy of Siemens was one of the initiators of this effort, and he made many important contributions along the way, including work on securing device discovery and leading one of the sample implementation development teams. Andrew Fiddian-Green of Siemens provided detailed technical feedback on the specification and single-handedly developed one of the sample implementations. Sony Electronics, LG Electronics, GlobespanVirata, and Atinav contributed sample implementations and provided feedback on the specification. Microsoft contributed to the specification and also enhanced the certification test tool to enable testing of secure devices.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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**:*DeviceSecurity:1*

The shorthand DeviceSecurity: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:ds=" http://www.w3.org/2000/09/xmldsig#"

2.3. Referenced Specifications

Unless explicitly stated otherwise herein, implementation of the mandatory provisions of any standard referenced by this specification shall be mandatory for compliance with this specification.

This specification references the UPnP Device Architecture version 1.0 and XML-Signature.

- http://www.upnp.org/download/UPnPDA10_20000613.htm
- http://www.w3.org/Signature/

2.4. MustUnderstand

If mustUnderstand="1" is present on the <us:SecurityInfo> header, the header must be processed in accordance with this specification or an error must be generated. This attribute is not expected to be used, except possibly in some future implementations in which <us:SecurityInfo> is directed at an intermediate agent.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.5. State Variables

DeviceSecurity:1 defines state variables given in the table below. In addition to the state represented by these variables, there is other persistent state (such as ACL, ACLVersion and owner list). Note that all of the security-relevant data for a device (ACL, ACLVersion, LifetimeSequenceBase, owner list and maybe the certificate cache) are expected to be in non-volatile memory so that they would survive power failures. Success of modifications to those data should not be reported until after successful update of that non-volatile memory. ACLVersion can be computed from the ACL itself (e.g., as the hash of the ACL) at the discretion of the manufacturer and in those cases need not be stored in non-volatile memory.

Note: The QueryStateVariable feature is not intended to be supported in the DeviceSecurity service.

Table 1: State variable

Variable Name	Req. or Opt. [1]	data type	Allowed Value	Default Value
NumberOfOwners	R	i4	>= 0	0
LifetimeSequenceBase	R	string		
TimeHint	O	string		
TotalACLSize	R	i4	>= 0	
FreeACLSize	R	i4	>= 0	
TotalOwnerListSize	R	i4	> 0	
FreeOwnerListSize	R	i4	>= 0	
TotalCertCacheSize	R	i4	>= 0	
FreeCertCacheSize	R	i4	>= 0	
A_ARG_TYPE_string	R	string		
A_ARG_TYPE_base64	R	bin.base64		
A_ARG_TYPE_int	R	i4		
A_ARG_TYPE_boolean	R	boolean		

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

2.5.1. NumberOfOwners

A device must maintain a variable which is the count of the number of currently registered owners.

2.5.2. LifetimeSequenceBase

This variable is a non-repeating value that is used to prevent replay for TakeOwnership and SetSessionKeys. Note that if the manufacturer desires to use a sequence number for the LifetimeSequenceBase, this number would have to be held in non-volatile memory where write operations can be expensive and can also be limited in total number performed over the lifetime of the part. Since the value need only be non-repeating, one can for example increment the non-volatile value by some quantity, N, and then keep the old LifetimeSequenceBase value in volatile memory, to be incremented by 1 until it reaches the non-volatile value.

There is a minor security advantage to having LifetimeSequenceBase be unpredictable. A manufacturer is free to make it so, for example by hashing some sequence counter with a secret value (e.g., the device's private key) or, if there is enough CPU power, by digitally signing a sequence counter and using that signature (or its hash) as the LifetimeSequenceBase.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.5.3. TimeHint

This variable is for devices that have no internal source of time but want to be able to handle certificates and ACL entries that have expiration times. A trusted Security Console or Control Point is able to reset this variable on occasion, to give the device a rough sense of the date and time.

The format of a time hint is as given in section 4.2.1, e.g., 2002-01-26T15:32:41Z.

2.5.4. TotalACLSize

This variable should be the sum of the current FreeACLSize and the number of entries already in the ACL. For devices that have chosen not to store an ACL locally, this can be 0. If TotalACLSize is 0, then the device should not offer the ACL editing actions.

2.5.5. FreeACLSize

This variable should be the number of ACL entries of average size (for that device, as a function of the number and size of permissions) that the device has remaining memory to store.

2.5.6. TotalOwnerListSize

This variable is the number of owner slots in the device's owner list. It must be at least 1. Every security aware device must be able to accept one owner. Multiple owners are optional but the space to record 3 or more owners is recommended, for convenience and fault tolerance.

2.5.7. FreeOwnerListSize

This variable is the number of owner slots in the device's owner list that are still available to be filled.

2.5.8. TotalCertCacheSize

This variable should be the number of certificates currently cached in the device plus the value of FreeCertCacheSize.

2.5.9. FreeCertCacheSize

This variable should be the estimated number of certificates of average size that would fit in the memory remaining for certificate cache on the device. A device that has no memory for certificate cache would set this and TotalCertCacheSize to 0.

2.5.10. A_ARG_TYPE_string

This is an arbitrary string parameter, not related to any real state variable.

2.5.11. A_ARG_TYPE_base64

This is an arbitrary string parameter, not related to any real state variable, carrying the BASE64 representation of a binary byte string or, in some cases, an XML document (so that it doesn't need to be escaped on a character by character basis). Refer to RFC1521 for a definition of BASE64. As defined in that spec (primarily for e-mail transmissions) BASE64 encodings are assumed to be limited in length, with a newline inserted as needed to make sure no line is over 76 characters in length. However, UPnP is not attempting to transmit data by e-mail and need not break lines to that length. In particular, one can generate BASE64 with no white space. One can use BASE64 with no line breaks in normal UPnP messages and must use BASE64 without added white space when building canonical public key encodings (see sections 4.7 and 4.9).

2.5.12. A_ARG_TYPE_int

This is an arbitrary 32-bit integer parameter, not related to any real state variable.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.5.13. A_ARG_TYPE_boolean

This is an arbitrary Boolean parameter, not related to any real state variable.

2.6. Eventing and Moderation

Table 2: Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
NumberOfOwners	*Yes*	*No*			*1*
LifetimeSequenceBase	*Yes*	*No*			
TimeHint	*No*	*N/A*			
TotalACLSize	*No*	*N/A*			
FreeACLSize	*Yes*	*No*			
TotalOwnerListSize	*No*	*N/A*			
FreeOwnerListSize	*Yes*	*No*			*1*
TotalCertCacheSize	*No*	*N/A*			
FreeCertCacheSize	*Yes*	*No*			
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).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.7. Actions

The DeviceSecurity service is to be added to a UPnP device when its actions need security. The actions of DeviceSecurity fall into two categories: those used only to maintain data defined and used by DeviceSecurity itself and those that can be used to interact with the device to which DeviceSecurity has been added. Similarly, Control Points that access a secured Device are characterized in two categories: **Security Consoles (SC)**, for those whose only purpose is to manipulate the data that is private to the DeviceSecurity service and **Control Points (CP)**, for those whose purpose is to interact with the device being secured. In the tables below, actions are grouped accordingly.

Table 3: Actions invoked by both Control Point and Security Console

Name	Req. or Opt. [1]	Authoriz.
GetPublicKeys	R	N
GetAlgorithmsAndProtocols	R	N
GetACLSizes	R	O
CacheCertificate	O	O
SetTimeHint	O	R
GetLifetimeSequenceBase	R	N
SetSessionKeys	R	O (1)
ExpireSessionKeys	R	R
DecryptAndExecute	R	O (2)
Non-standard actions implemented by an UPnP vendor go here.	X	O

[1] R = Required, O = Optional, X = Non-standard, N = Not to be used.

Table 4: Actions invoked by a Security Console only

Name	Req. or Opt. [1]	Authoriz.
TakeOwnership	R	(1)
GetDefinedPermissions	R	O
GetDefinedProfiles	O	O
ReadACL	O[2]	O
WriteACL	O[2]	R
AddACLEntry	O[2]	R
DeleteACLEntry	O[2]	R
ReplaceACLEntry	O[2]	R
FactorySecurityReset	R	R
GrantOwnership	O	R
RevokeOwnership	O	R
ListOwners	O	O
Non-standard actions implemented by an UPnP vendor go here.	X	O

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

[2] The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3). If the device does have an ACL, then the ACL editing actions are required.

The Authoriz column indicates whether a given action may be, must be or must not be authorized.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Note (1): TakeOwnership and SetSessionKeys are special cases in that they must be signed but do not require authorization via ACL entry. TakeOwnership cannot be authorized because it works only when a device is unowned and an unowned device cannot have any ACL entries granting authorization. SetSessionKeys might be access controlled, at the vendor's discretion, if one is worried about the creation of too many session keys (perhaps as part of a denial-of-service attack), but if it is not itself authorized, no attacker can use that to get access to a secured action.

Note (2): DecryptAndExecute does not require its own authorization and must not be signed with the same session key used inside the encrypted SOAP body because that would introduce a conflict in freshness values. The encrypted SOAP body it carries should be signed and authorized, if the action in that body requires it. DecryptAndExecute used for tunneling might need to be signed, but in that case the signature keys of the outer and inner message would be different.

2.8. Cryptographic Notation for Selected Actions

To help explain the cryptographic aspects of some actions, a compact notation is used. There are typically three parties involved, a device, a control point, and a security console. The letters D, C and S are used to indicate these, respectively. The public key of a party is indicated by P, the private key by K. Thus P_D is the public key of the device, while K_D is its private key. Session encryption keys are represented by K subscripted with both parties; thus K_{DC} would be a session key used from the device to the control point, and K_{CD} the session key used from the control point to the device. Similarly, session signing keys are represented by S with the appropriate subscripts.

Encryption is represented by [], while signing is represented by {}. These operations are prefixed by the key used. So, for example, $P_D[K_C\{m\}]$ represents a message m, signed with the private key of control point C, and encrypted with the public key of device D. Concatenation is represented by |. In some cases, encryption requires an Initialization Vector (IV) in which case the key used will be subscripted with the IV as well, to indicate encryption under both the key and IV.

Finally, the transmission of a message m from A to B is written as A→B:m

2.9. Actions Invoked by Both CP and SC

2.9.1. GetPublicKeys

GetPublicKeys retrieves the set of public keys corresponding to private keys held by the device. This spec has defined a use only for a confidentiality key for the device, so one should expect to receive only one key in response.

2.9.1.1. Arguments

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

[R] = RetVal

The KeyArg argument is an escaped XML string (see section 3.2) containing key definition elements. Within this string, the Modulus and Exponent values are BASE64 encoded.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
<Keys>
  <Confidentiality>
    <RSAKeyValue>
      <Modulus>xA7SEU+e0y…</Modulus>
      <Exponent>AQAB</Exponent>
    </RSAKeyValue>
  </Confidentiality>
  <Signing>
    <RSAKeyValue>
      <Modulus>xA7SEU+e0y…</Modulus>
      <Exponent>AQAB</Exponent>
    </RSAKeyValue>
  </Signing>
</Keys>
```

In this version of the Security specification, devices have only confidentiality keys. The structure of a `<Keys>` block is designed to permit devices to have signing keys as well, should that need arise in future versions of this service. If there is no signing key, then the `<Signing>` element is omitted.

The device's confidentiality key is used for the TakeOwnership and SetSessionKeys actions.

Note: If a device is also a control point, that control point has a signing key, but the signing key is announced to a Security Console via a different mechanism (PresentKey) and would not be presented in this `<Keys>` element. (See SecurityConsole:1 for details.)

2.9.1.2. Effect on State

None.

2.9.1.3. Errors

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

2.9.2. GetAlgorithmsAndProtocols

GetAlgorithmsAndProtocols retrieves a description of the algorithms and protocols supported by the device.

2.9.2.1. Arguments

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

[R] = RetVal

The Supported parameter is an escaped XML string (see section 3.2) giving algorithms and protocols supported by this device, in the format below. The list of all currently defined protocol and algorithm IDs is given below.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
<Supported>
  <Protocols>
      <p>UPnP</p>
      <p>TLS</p>
      <p>IPSEC</p>
  </Protocols>
  <HashAlgorithms>
      <p>SHA1</p>
  </HashAlgorithms>
  <EncryptionAlgorithms>
      <p>NULL</p>
      <p>RSA</p>
      <p>AES-128-CBC[1]</p>
  </EncryptionAlgorithms>
  <SigningAlgorithms>
      <p>NULL</p>
      <p>RSA</p>
      <p>SHA1-HMAC</p>
  </SigningAlgorithms>
</Supported>
```

Note 1: The "RSA" ID refers to (RSA, PKCS#1V1.5) for encryption and (RSA, PKCS#1, SHA1) for signing. Should another algorithm tuple be defined for encryption or signing, that tuple would need a different algorithm ID.

Note 2: A NULL entry indicates that although the encryption or signing is *supported*, it is not *required*. If there is no NULL entry, then that feature is required. If there is no algorithm listed, then the feature is not supported.

2.9.2.2. Effect on State

None.

2.9.2.3. Errors

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

2.9.3. GetACLSizes

This action returns the current and total allocation sizes for ACLs and related storage.

A device can have three different blocks of memory holding ACL and related data:

1. the device ACL
2. the owner list (effectively an ACL for both the ACL and the owner list itself)
3. a certificate cache (for holding certificates that extend the base ACL via delegation)

This action returns the values of those memory allocations, both total and current, in units of number of entries. The FreeACLSize and FreeCertCacheSize values are estimates only, since ACL entries and certificates can be of variable length, although in many devices they will be of fixed length. An owner list entry is fixed length and should have definite list sizes.

[1] CBC: Cipher Block Chaining – see the Glossary.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.3.1. Arguments

Argument(s)	Direction	relatedStateVariable
ArgTotalACLSize	OUT	TotalACLSize
ArgFreeACLSize	OUT	FreeACLSize
ArgTotalOwnerListSize	OUT	TotalOwnerListSize
ArgFreeOwnerListSize	OUT	FreeOwnerListSize
ArgTotalCertCacheSize	OUT	TotalCertCacheSize
ArgFreeCertCacheSize	OUT	FreeCertCacheSize

A device developer is free to define memory structures within the device. There is no reason to reveal those definitions to a caller. Therefore, sizes returned in this call are in units of number of entries. Some methods of storing ACL entries and certs allow for arbitrary permission fields, with arbitrary parameter lists, perhaps stored as linked lists. Such a memory structure is not fixed length and any device choosing such a structure should estimate the average size of an ACL entry or cert (for that device) and report an estimate of the memory remaining for such structures. A CP or SC wishing to deliver an excessively large ACL entry or cert must therefore expect the possibility of an error for lack of memory even if the corresponding free size was non-0.

In the minimum persistent memory configuration, a device will remember one owner and that one owner will issue all permissions by certificate (since there is no ACL). Owners have `<all/>` permission by default and therefore do not need ACL entries for themselves. (see, section 4.5.1).

If a device has the memory for it, it may maintain a certificate cache. A CP or SC that wishes to can then submit one or more certificates to the device via the CacheCertificate action. Such certificates would remain on the device in the cache until that storage was required for other purposes or the certificates expire. An advanced device could pre-compute an effective ACL from its ACL and the cached certificates, and use that effective ACL in subsequent action authorization computations, but that is a performance choice and of no security import.

2.9.3.2. Effect on State

None.

2.9.3.3. Errors

errorCode	errorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
701	Not authorized	The caller was not authorized to perform this action.
711	Signature Failure	A signature failed to verify
712	Signature Missing	This command needs to be signed and authorized
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.9.4. CacheCertificate

An authorization certificate (see section 4.4.1) is a signed ACL entry, by which one SC delegates some or all of the rights it has been granted to another SC or CP. It can be generated by the device owner when the device has too little ACL space to hold all the entries that ACL would need. It can also be generated by a non-owner who has been given some rights and who wants to allow someone else to have some of those rights.

A name certificate is generated by a SC and is used to define a named group of control points (or SCs). Each name certificate declares one key (or other named group) to be a member of the group being defined.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

This action is optional. A device may, at the manufacturer's discretion, offer memory for a cache of certificates and report that decision via GetACLSizes. If the device does not offer memory for a certificate cache, this action should not be offered.

Support for certificates is optional. When they are supported, they can be communicated from the CP to the Device within the XML-Signature signature block, or they can be sent ahead, via CacheCertificate. If they are cached, then individual messages need not carry them and therefore can be smaller. Such certificates would remain on the device in the cache until that storage was required for other purposes or the certificates expire.

If CacheCertificate is offered, it is up to the manufacturer's discretion whether it should be access-controlled. One might want to employ access control in order to ward off Denial of Service attacks by using up a device's certificate cache memory. One might want not to employ access control in order to keep from complicating the design of the device's permissions.

2.9.4.1. Arguments

Argument(s)	Direction	relatedStateVariable
Certificates	IN	A_ARG_TYPE_string

The Certificates argument is an XML structure, communicated as a properly escaped string (see, section 3.2) in this parameter. The argument is a sequence of one or more certificates (e.g., a chain of certificates that grants a subject some permission(s)). [A "subject" is the grantee of some rights and can be an individual key or a named group of keys.] Those certificates are defined in section 4.4. The structure is of the form:

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

It is expected that a device will verify the signature on the offered certificates and cache only the certificate bodies of verified certificates, rejecting certificates that do not verify. It is up to the device whether to accept any valid certificates if they were in a `<Sequence>` with invalid certificates but it is recommended that valid certificates be accepted even if other certificates cached at the same time are invalid. If a certificate is invalid because its time has expired, it is up to the Control Point offering the certificate to get that certificate replaced or renewed. That is not the responsibility of the device.

2.9.4.2. Effect on State

If successful, the provided certificates are added to the device's certificate cache. Advanced devices can also pre-compute an effective ACL given the new certificate cache and the current ACL, and then use that new effective ACL for future authorization decisions when the caller provides no additional certificates with the call.

Caching of certificates via this action is independent of any decision on the part of the device to cache certificates that are presented within an XML-Signature element. Automatic caching of certificates offered in XML-Signature signature elements will not save on message size, because the CP will not know that the device has done this caching, but the device can cache certificate bodies whose signatures have been verified and thereby save signature verification time in future uses of those cached certificates.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.4.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not authorized	The caller was not authorized to perform this action.
711	Signature Failure	A signature failed to verify
712	Signature Missing	This command needs to be signed and authorized
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
731	Wrong device	The certificate offered is not for this device.
751	No memory	There is insufficient memory to accept the offered certificate.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.9.5. SetTimeHint

This optional action should be provided by those devices that have no local source of time but will need time for processing ACL entries or certificates that contain validity dates and times. A Security Console (SC) or other control point on discovering such a device can provide the date and time periodically, assuming it has been authorized to do so. This date and time is to be used in validity tests of certificates or ACL entries. The time interval between updates of the TimeHint can be a configuration parameter of a Security Console and not something for the device to be concerned about. The device merely accepts times as offered and assumes that to be the time until the next SetTimeHint. No attempt is expected to be made to actually count time, treating SetTimeHint as a clock setting function.

This action should be permitted only to those SCs and CPs the owner trusts, since otherwise an attacker might submit false times in order to recover expired access.

See section 4.2 for an explanation of time limits in ACL entries.

2.9.5.1. Arguments

Argument(s)	Direction	relatedStateVariable
ArgTimeHint	IN	TimeHint

The offered TimeHint replaces the stored TimeHint, assuming the caller was authorized to make this action call. Times are to be in the format given in section 4.2.1 e.g.: “2001-12-24T18:13:10Z”.

Note that a device containing a hardware calendar clock need not offer SetTimeHint and its associated state variable, TimeHint. If a device does include a calendar clock, how that clock is set is up to the device manufacturer who would probably not use SetTimeHint for that purpose.

2.9.5.2. Effect on State

The TimeHint variable is updated by the value provided. There is no requirement that changes to TimeHint be monotonic increasing.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.5.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not Authorized	Caller is not authorized to perform this action
711	Signature Failure	The signature failed to verify
712	Signature Missing	This command needs to be signed and authorized.
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.9.6. GetLifetimeSequenceBase

This action retrieves the current value of the LifetimeSequenceBase state variable.

2.9.6.1. Arguments

Argument(s)	Direction	relatedStateVariable
ArgLifetimeSequenceBase	OUT [R]	LifetimeSequenceBase

[R] = RetVal

The LifetimeSequenceBase variable holds a persistent, non-repeating string value that is changed (e.g., incremented) with each use. It is used in a public-key `<Freshness>` block and in computing the main parameter of TakeOwnership, in order to prevent replay.

This action must not be access controlled. It is used by a Security Console prior to a TakeOwnership action. Prior to TakeOwnership, a device can not have an ACL entry granting the permission to access this action.

The LifetimeSequenceBase must also not be reset by FactorySecurityReset. It is to be unique over the full lifetime of the device. However, if the device is capable of generating a new public-key pair, then the LifetimeSequenceBase can be reset on generation of a new key pair, since a device is known (under UPnP Security) only by its public key. With a new public-key, one has in effect a new device.

See sections 2.5.2 and 2.11 for further discussion of LifetimeSequenceBase.

2.9.6.2. Effect on State

There is no effect on state.

2.9.6.3. Errors

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.9.7. SetSessionKeys

SetSessionKeys allows a Control Point to establish symmetric keys for subsequent signed or encrypted operations. These will typically be:

1. signed SOAP action calls,
2. encrypted SOAP envelopes made via DecryptAndExecute,
3. and optionally other protocols (e.g., IPsec, TLS or whatever else the device manufacturer wants to provision)

A session will remain valid until either the CP or the Device discards it. The CP discards it via ExpireSessionKeys. The Device discards it silently, but the fact of that discard is apparent to the CP the next time it tries to use that session and gets an error return that there is no such session. Typically, a device will discard sessions only when they are inactive for a long time or when the Device is out of session description memory.

2.9.7.1. Arguments

Argument(s)	Direction	relatedStateVariable
EncipheredBulkKey	IN	A_ARG_TYPE_base64
BulkAlgorithm	IN	A_ARG_TYPE_string
Ciphertext	IN	A_ARG_TYPE_base64
CPKeyID	IN	A_ARG_TYPE_int
DeviceKeyID	OUT [R]	A_ARG_TYPE_int
SequenceBase	OUT	A_ARG_TYPE_string

[R] = RetVal

Using the cryptographic notation, one can write this as:

C→D: $K_C\{\text{SetSessionKeys}(P_D[K_{Bulk}, IV_{Bulk}], Algorithm_{Bulk}, K_{Bulk,IVbulk}[Keys], CPKeyID)\}$

D→C: $S_{DC}\{\text{SetSessionKeysResponse}(DeviceKeyID, SequenceBase)\}$

where *Keys* is the XML key structure described below.

The EncipheredBulkKey ($P_D[K_{Bulk}, IV_{Bulk}]$) is described in section 4.6, below. It carries a symmetric session key and IV for the Ciphertext.

The BulkAlgorithm ($Algorithm_{Bulk}$) argument is the name of the symmetric algorithm used to encipher the Ciphertext. The name specifies algorithm, key length and encryption mode. For example, AES-128-CBC is a recommended algorithm, key length and mode.

The Ciphertext ($K_{Bulk,IVbulk}[Keys]$) is bulk encrypted with the symmetric algorithm, symmetric key and IV given in the first two parameters. (See 4.8 and 4.6.1) It is encoded as a BASE64 string. Its plaintext (*SessionKeys*) is an XML structure as follows (for example):

```
<SessionKeys>
    <Confidentiality>
        <Algorithm>AES-128-CBC</Algorithm>
        <KeyToDevice>XXXXXXXX</KeyToDevice>
        <KeyFromDevice>YYYYYYYYY</KeyFromDevice>
    </Confidentiality>
    <Signing>
        <Algorithm>SHA1-HMAC</Algorithm>
        <KeyToDevice>XXXXXXXX</KeyToDevice>
        <KeyFromDevice>YYYYYYYYY</KeyFromDevice>
    </Signing>
</SessionKeys>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Key fields in the plaintext block above are in network standard byte order (most significant byte first) and BASE64 encoded.

CPKeyID is an integer type input parameter that contains the index into or search key of a key table on the control point. It is made available to the device for use in event or discovery messages, should those be encrypted by session key, so that the CP can be informed about the key that was used. For the reply to a message, the CP can assume that the same session was used in reply as in the call message.

DeviceKeyID is an integer type output parameter that contains a unique session key identifier for the newly opened session. The Control Point must refer to this key identifier when it terminates the session by making a call to ExpireSessionKeys and when it uses one of these session keys, in XML-Signature blocks or in DecryptAndExecute actions.

SequenceBase is a string type output parameter (possibly an integer encoded as a string) that contains a string unique to that device for this session. That string might be a large random byte string or might be monotonic index counting up from the birth of the device. Its only requirement is that it be unique (at least statistically) over the lifetime of the device and refer to this session. Its purpose is to help prevent replay attacks within sessions. See section 5.2 for a full description of the use of the SequenceBase.

The action call must be signed by the private signing key of the Control Point (i.e. verified by its public signing key). Most actions will be signed and verified by symmetric keys, established with a SetSessionKeys call, but this one must use public key verification.

The LifetimeSequenceBase is a non-repeating value, maintained by the device. It is used in this action within the `<Freshness>` block of the public key signature to prevent replay attacks.

A successful reply to this action, if signed, should be signed by the session key established in this call (<Signing><KeyFromDevice>). An error reply to this action can be sent unsigned.

It is up to the device manufacturer whether this action requires authorization. A session key is not granted any power *per se*. It is associated with the public key that verified the SetSessionKeys action call. If that public key is authorized, then the associated session key is authorized. However, the session keys (and SequenceBase) must be kept in memory and that memory might be a scarce resource. In that case, the device may choose to exercise access control over the SetSessionKeys action to prevent a resource exhaustion attack.

A manufacturer is free to restrict the allocation of sessions and to define error codes to reflect those restrictions. For example, it is possible that a manufacturer could permit only one active session per control point public key. In that case, it is recommended that an otherwise successful request for a new session override any previous session (in case the control point had reinitialized and forgotten both the existence and the values of the previous session). It is also possible for a device to discard a session at any time, forcing establishment of a new session, although excessively rapid dropping of sessions would create a great deal of needless overhead.

2.9.7.2. Effect on State

SetSessionKeys defines new session keys and key IDs maintained by the device until ExpireSessionKeys is called.

For this session, the SequenceBase (generated during the call) is remembered and the sequence numbers for messages in this session are set to 0. Subsequent messages using this session must have the same SequenceBase and monotonically increasing (but not necessarily contiguous) sequence numbers in each direction in order to be accepted as fresh. That is, the CP should keep two sequence number counters, one that it uses and one that the device uses. A device may keep one or two sequence counters, since each message has precisely one reply and the device could use the incoming sequence counter value as the reply counter value.

After a successful SetSessionKeys, the LifetimeSequenceBase is incremented (see 2.5.2) so that every new session is created by a fresh command.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.7.3. 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	Action Not Authorized	Caller is not authorized to perform this action. This probably means that the caller has no access rights at all on this device.
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed by an asymmetric key.
714	Invalid Sequence	LifetimeSequenceBase is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
721	Algorithm Not Supported	A requested algorithm is not supported.
722	No IPSEC	This device is not capable of IPSEC.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.9.8. ExpireSessionKeys

ExpireSessionKeys permits a control point to inform a device explicitly that it no longer intends to use a session. The device should then destroy the keying information and free any resources associated with it.

2.9.8.1. Arguments

Argument(s)	Direction	relatedStateVariable
DeviceKeyID	IN	A_ARG_TYPE_int

DeviceKeyID is the device's session key identifier returned by SetSessionKeys. After this call, the device will have forgotten all of the indicated session keys and those keys are therefore unusable.

This call can be honored only from the owner of that session, in order to prevent denial of service annoyances.

2.9.8.2. Effect on State

If the call was successfully authorized, ExpireSessionKeys removes the corresponding session from the table of session key information. Note: if the session being expired was used in this call, the reply keys for that session should be deleted only after encrypting and/or signing the reply.

With the removal of that session, if there are other things (encrypted events, IPSEC security associations, etc.) that rely on keys that had been defined in the session just removed, then those other things will probably need to be removed or stopped as well.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.9.8.3. 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	Action Not Authorized	Caller is not authorized to perform this action.
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.9.9. DecryptAndExecute

DecryptAndExecute provides secrecy of a SOAP message by tunneling it inside an action argument. Specifically, that message is held in the Request argument and any reply from that message is returned in the Reply argument. The Request and Reply arguments are encrypted. The payload message is probably signed, but the signing of that message is independent of the operation of DecryptAndExecute itself. The DecryptAndExecute message may also be signed, in special circumstances (e.g., when a message needs to be signed to be authorized to get through a SOAP intermediary or a gateway device). In this latter case, the key that signs the DecryptAndExecute message will doubtless be a different key from the one that signs the payload, since they are being processed by different devices.

There are three input arguments to DecryptAndExecute: the key ID of the key being used, an IV value for the symmetric encryption and a Ciphertext block containing a SOAP message to be executed. The output arguments are an output IV value and a Ciphertext block holding the reply from that SOAP message. It is assumed that the caller has done SetSessionKeys before doing DecryptAndExecute, in order to get the DeviceKeyID for the session keys being used.

2.9.9.1. Arguments

Argument(s)	Direction	relatedStateVariable
DeviceKeyID	IN	A_ARG_TYPE_int
Request	IN	A_ARG_TYPE_base64
InIV	IN	A_ARG_TYPE_base64
Reply	OUT [R]	A_ARG_TYPE_base64
OutIV	OUT	A_ARG_TYPE_base64

[R] = RetVal

Using the cryptographic notation, this action and response can be written:

C→D: DecryptAndExecute(*DeviceKeyID,* $K_{CD,InIV}[S_{CD}\{HTTP\text{-}SOAP\text{-}Request\}]$,*InIV*)

D→C: DecryptAndExecuteResponse($K_{DC,OutIV}[S_{DC}\{HTTP\text{-}SOAP\text{-}Response\}]$,*OutIV*)

The Request and Reply arguments contain Ciphertext. The former is encrypted using the indicated "to device" session key, and the latter using the corresponding "from device" key. Both ciphertexts are encoded as BASE64 strings. Plaintexts are to be padded as described in section 4.8.

The plaintext within Request or Reply is a full SOAP HTTP message. The plaintext is to be NULL-terminated, for the convenience of HTTP processors written in C. It is to be passed back into the HTTP processor, XML parser and SOAP engine, recursively.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Note: There is a conflict with sequence numbers if the DecryptAndExecute call itself is signed with the same DeviceKeyID as the internal (encrypted) message. The preferred solution is to not sign the DecryptAndExecute call itself, but to only sign the internal message. An alternate solution is to use different session keys for the two signatures.

2.9.9.2. Effect on State

The embedded action may affect the state or may not. DecryptAndExecute does not, itself, modify the device's state.

2.9.9.3. 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	The caller was not authorized to perform this action.
711	Signature Failure	A signature failed to verify
712	Signature Missing	This command needs to be signed and authorized
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
741	Invalid Key	The Action Ciphertext did not decipher to valid XML. The probable cause is that the incorrect key was used.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

Note: errors returned by DecryptAndExecute pertain to the execution of that action by itself and not the action contained in Request. One needs to decrypt and parse the Reply to discover whether the internal action produced an error and, if so, which one.

2.10. Actions Invoked by SC only

2.10.1. TakeOwnership

This action permits a Security Console to claim ownership of an unowned, security-aware device. It must be digitally signed by the Security Console private signing key. [An unowned device can not yet set up sessions, so the signature must be by public key algorithm.]

2.10.1.1. Arguments

Argument(s)	Direction	relatedStateVariable
HMACAlgorithm	IN	A_ARG_TYPE_string
EncryptedHMACValue	IN	A_ARG_TYPE_base64

Using the cryptographic notation, this action can be written as:

S→D: K_S\{TakeOwnership("SHA1-HMAC",P_D[SHA1-HMAC(*secret*, (P_S|P_D|*nonce*))])}

D→S: TakeOwnershipResponse

The HMACAlgorithm, currently only SHA1-HMAC, gives the hash algorithm used to construct the EncryptedHMACValue.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The EncryptedHMACValue is formed by encrypting a value formed according to RFC2104, using the hash algorithm named in HMACAlgorithm. The value is a BASE64 encoding of HMAC(Secret, (SCPublicKey | DevicePublicKey | LifetimeSequenceBase)), where the first argument to HMAC is the key and the second is the value being hashed. In this computation, both the Secret and the LifetimeSequenceBase are printable strings, while the PublicKey blocks are XML structures with no white space.

Specifically, let

H = SHA1-HMAC(Secret, (SCPublicKey | DevicePublicKey | LifetimeSequenceBase))

where SCPublicKey is the (XML formatted) Security Console signing public key and DevicePublicKey is the (XML formatted) Device confidentiality public key. The Secret is the device's ownership password (a UTF-8 string). The LifetimeSequenceBase is the string returned by the device from GetLifetimeSequenceBase, which needs to be called just before TakeOwnership. The operator "|" is normal string concatenation.

This value H is then encrypted in the public key of the device, using PKCS#1 V1.5 padding, as described in section 4.6 and 0.

The reply message to this action is not intended to be signed.

The LifetimeSequenceBase changes with each call to TakeOwnership, whether successful or not, so that the value, H, will not ever repeat..

Device manufacturers should note that the secret value used needs to be unique to the device and large enough to withstand repeated guessing attacks mounted by a computer when the device first comes online. The computation of H insures that no other CP-Device pair could produce that value and that even that pair would not produce the same value of H at a future time. Therefore, H can not be used in a replay attack. By encrypting H to form EncryptedHMACValue, no attacker could learn something from observing someone else's TakeOwnership message.

An attack on TakeOwnership then requires either possession of the secret (e.g., from seeing it written in device documentation or on a label on the physical device) or a network-mediated guessing attack on the secret. That attack will require 2 network round trips, one to get a new LifetimeSequenceBase and the second to try a new guessed Secret. If these two round trips take 1 millisecond each, for example, then the following table shows how long a secret of a given size is expected to withstand attack, assuming that the secret value is random and expressed as BASE32 characters:

# chars	Time
2	.5 sec
4	9 min
6	6 days
8	17 years

If the Device generates its own keys and Secret values (which it can, if it has a source of randomness and an output device capable of displaying or printing a key hash and a secret value), then the exposure time does not accumulate. The exposure time is only that time between when a device is plugged in and when TakeOwnership by the proper SecurityConsole succeeds. After that time, the device will not respond to TakeOwnership, even if the secret were correct, so an attacker would learn nothing from repeated attempts.

If the Device comes with a permanent built in secret, then exposure time accumulates over all times that the device is on the network and not yet owned, presumably times that occur only when the device changes hands from the store to the first owner and then to a second owner. These times should be years apart.

The use of a secret shared between the device and the Security Console provides a common mechanism for taking ownership. This mechanism must be available in all devices. However, this is not to preclude the possibility of a device and security console product, from the same vendor, constructed to permit the taking of ownership in some more efficient manner. For example, one might have a security console that is portable and that can be touched to an electrical contact on the device to achieve taking of ownership. Any alternative TakeOwnership mechanism must be analyzed for security to insure that it does not represent a security hole. For example, an exchange of keys over a dedicated point-to-point physical connection can be as secure as the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

TakeOwnership protocol defined here and can be more convenient for some users and some devices, at the cost of the additional hardware interface.

The security of the default TakeOwnership process requires that the user interface for TakeOwnership should require a user to verify the hash of the device key before supplying the device password.

2.10.1.2. Effect on State

If the device was already owned, the signature fails or the secret does not match, the action is rejected without processing. If the device was not owned, the secret matches and the signature verifies by the included Security Console public key, then the device enters the hash of the Security Console's signing public key as the first (and currently only) owner of the device and FreeOwnerListSize is updated appropriately.

The device's ACL is not affected by this action, but there is an implicit effect. Every owner is treated as if it had `<all/>` permissions, by virtue of being listed in the owner table, and therefore does not need an entry in the device's ACL.

After a TakeOwnership call, the LifetimeSequenceBase is updated, whether the call was successful or not. This is to prevent reuse of the EncryptedHMACValue by any eavesdropper.

2.10.1.3. Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed by an asymmetric signing key.
714	Invalid Sequence	LifetimeSequenceBase value is incorrect
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
721	Algorithm Not Supported	The HMAC algorithm chosen is not supported
761	Device Owned	Action not allowed on a device already owned.
762	HMAC failed	The HMAC value failed to verify
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.2.GetDefinedPermissions

This action may or may not be access controlled according to the device manufacturer's desire.

This action returns a list of permissions defined by the manufacturer for the device. The permission `<all/>`, which means "all permissions", is defined for all devices and is not included in the list returned.

2.10.2.1.Arguments

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

[R] = RetVal

The Permissions parameter is a string (escaped XML element – see section 3.2) of the form:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
<DefinedPermissions xmlns:mfgr="…">
      <Permission>
            <UIname>basic</UIname>
            <ACLEntry> <mfgr:p1/> </ACLEntry>
            <FullDescriptionURL>http://.....</FullDescriptionURL>
            <ShortDescription>
                  This permission allows the user to
                  control volume and channel.
            </ShortDescription>
      </Permission>
</DefinedPermissions>
```

This is the set of all manufacturer-defined device permissions. The <UIname> element is the name of the permission to be displayed by the Security Console during ACL editing. The <ACLEntry> name is the permission to be included in an ACL entry or certificate, to represent this permission. The short description is available e.g. for use as a tool tip. The full description, if present, is available for the user to read via a browser window.

There should be as many <Permission> elements inside the <DefinedPermissions> element as are defined on the device. Each <Permission> describes a single permission. A permission might be of the form, for example:

```
<mfgr:read/> <mfgr:write/> <mfgr:read-ACL/> <mfgr:power-on-off/>
<mfgr:operate/> <mfgr:reset/> or <APWG:APDeviceAll/>
```

and will be in the device manufacturer's namespace or in the namespace of the UPnP WC, whichever defined the permission.

Advanced devices may parameterize these permissions, but normal devices are assumed to name permissions as XML elements with no sub-structure. See section 4.5, below, for a discussion of permission parameters.

2.10.2.2.Effect on State

None.

2.10.2.3.Errors

errorCode	errorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
701	Not authorized	The caller was not authorized to perform this action.
711	Signature Failure	A signature failed to verify
712	Signature Missing	This command needs to be signed and authorized
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.3. GetDefinedProfiles

This action may or may not be access controlled at the manufacturer's discretion. If the manufacturer has defined profiles (named sets of permissions, probably relating to roles such as "reader", "reviewer", "guest", "child", "parent", "homeowner", etc.), then this action returns the set of defined profiles and their definitions.

Profiles are optional. They can be provided by a device manufacturer when the set of permissions is too large to permit simple administration. They can be used to indicate a manufacturer's idea of the permissions that users in various roles should receive, without binding the device owner to those selections.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2.10.3.1.Arguments

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

[R] = RetVal

Profiles is an XML structure (escaped for transmission as a string – see section 3.2) listing all defined named profiles, giving for each one the name of the profile, the set of permissions which is the definition of that profile and explanatory text defining the profile for the user, for example:

```
<Profiles xmlns:mfgr="…">
      <Profile>
            <UIname>parent</UIname>
            <Definition><mfgr:p1/><mfgr:p2/><mfgr:p3/></Definition>
            <FullDescriptionURL>http://.....</FullDescriptionURL>
            <ShortDescription>
                  This is the normal setting for parents.
            </ShortDescription>
      </Profile>
      <Profile>
            <UIname>child</UIname>
            <Definition><mfgr:p2/></Definition>
            <FullDescriptionURL>http://.....</FullDescriptionURL>
            <ShortDescription>
                  This is the normal setting for a young child.
            </ShortDescription>
      </Profile>
</Profiles>
```

The text in a <UIname> element is the role label that could be displayed by a Security Console that uses defined profiles for permission editing. The <Definition> is the set of permissions that constitute the profile and this set will appear in an ACL entry or certificate. The <ShortDescription> might be used as a tool tip. The <FullDescriptionURL> can give the user a full description to read, via a browser window.

2.10.3.2.Effect on State

None.

2.10.3.3.Errors

errorCode	ErrorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not authorized	The caller was not authorized to perform this action.
711	Signature Failure	A signature failed to verify
712	Signature Missing	This command needs to be signed and authorized
714	Invalid Sequence	The sequence base or number is incorrect
715	Invalid Control URL	The control URL within the signature block does not match the one used to deliver the action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.4.ReadACL

This action returns a device's ACL and the current version of that ACL. The version returned here must be supplied as an input argument to WriteACL, DeleteACLEntry or ReplaceACLEntry since those actions assume current knowledge of the ACL state. With any edit of the ACL, the version is changed, and that change should not be predictable. For example, one might use the BASE64 encoding of the hash of the entire ACL as its Version. AddACLEntry does not need to know the current state of the ACL and therefore does not need to be preceded by a ReadACL call.

ReadACL may or may not be access controlled, at the device manufacturer's discretion.

The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3),

2.10.4.1.Arguments

Argument(s)	Direction	relatedStateVariable
Version	OUT [R]	A_ARG_TYPE_string
ACL	OUT	A_ARG_TYPE_string

[R] = RetVal

The Version is a string. See section 2.11.3 for details about ACLVersion.

The ACL is an XML structure (escaped to form a string – see section 3.2) of the form:

```
<acl> {<entry> … </entry>}* </acl>
```

where the ACL Entry is defined in section 4.2, below.

2.10.4.2.Effect on State

None.

2.10.4.3.Errors

errorCode	errorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Action Not Authorized	Caller is not authorized to perform this action.
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.5.WriteACL

This action replaces the existing ACL of the device with a new ACL assuming the caller knows the correct current version of the ACL being replaced. This action must be available only to authorized entities (owners and possibly delegates).

The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3),

2.10.5.1.Arguments

Argument(s)	Direction	relatedStateVariable
Version	IN	A_ARG_TYPE_string
ACL	IN	A_ARG_TYPE_string
NewVersion	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

For details about Version and NewVersion, see section 2.11.3.

The ACL is an XML structure (escaped to form a string – see section 3.2) of the form:

```
<acl> {<entry> … </entry>}* </acl>
```

where the ACL Entry is defined in section 4.2, below.

2.10.5.2.Effect on State

If the call is successful, the ACL given in the arguments replaces the existing ACL.

2.10.5.3.Errors

errorCode	errorDescription	Description
402	Invalid Argument	One of the input arguments is invalid.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Action Not Authorized	Caller is not authorized to perform this action.
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
774	Incorrect ACLVersion	The TargetACLVersion given did not match ACLVersion at the time of this call.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.6.AddACLEntry

This action must be invoked by authorized entities (owners and possibly delegates). The provided ACL entry is added to the device's ACL, assuming it is not already present, is correctly formatted and has memory in which to be held.

The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3),

2.10.6.1.Arguments

Argument(s)	Direction	relatedStateVariable
Entry	IN	A_ARG_TYPE_string

The ACL entry is an XML element, as described in section 4.2, encoded for transmission as a string – see section 3.2.

This ACL modification action is the exception to the rule that a correct ACLVersion value must be provided as an input. The entries in an ACL are treated, logically, as an unordered set. A subject receives the union of the permissions assigned to it by whatever entries grant it permission. Therefore, adding an entry to the ACL does not require any position or index into the ACL and does not require knowledge of the previous state of the ACL. As a result, the caller is not required to submit an ACLVersion value

AddACLEntry does not refer to any existing entry and therefore the caller does not need to demonstrate knowledge of the state of the ACL (which is what ACLVersion is used to do). See section 2.11.3 for details about ACLVersion.

2.10.6.2.Effect on State

If successful, the device ACL is modified to include the new entry and ACLVersion is changed to reflect the change in state of the ACL. The new ACLVersion is not returned to the caller since the caller may not have known the state of the ACL prior to performing this action and may not know what the device does when adding an entry.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.6.3.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not Authorized	The action in question was not issued by an authorized party (in this case, an owner of the device)
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
751	Insufficient memory	There is insufficient memory to add a new ACL entry.
771	Entry already present	The offered ACL entry is already present in the ACL
773	Malformed entry	The offered ACL entry has a format error
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.7.DeleteACLEntry

This action must be invoked by authorized entities (owners and possibly delegates). The TargetACLVersion provided must match the current ACL Version, to guarantee that the caller based its DeleteACLEntry decision on the correct current contents of the ACL.

The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3),

2.10.7.1.Arguments

Argument(s)	Direction	relatedStateVariable
TargetACLVersion	IN	A_ARG_TYPE_string
Index	IN	A_ARG_TYPE_int
NewACLVersion	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

For details of TargetACLVersion and NewACLVersion, see section 2.11.3.

The Index into the ACL starts at 0 and counts entries. The first entry is 0, the second is 1, etc.

2.10.7.2.Effect on State

If the action invocation is successful, the indicated ACL entry is deleted and the ACLVersion is changed. The nature of the change is up to the device manufacturer and is not to be assumed by any control point. The BASE64 encoding of the hash of the ACL is recommended as a good ACLVersion.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.7.3.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not Authorized	The action in question was not issued by an authorized party (in this case, an owner of the device)
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed by an asymmetric key.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
772	Entry does not exist	The indicated ACL entry does not exist.
774	Incorrect ACLVersion	The TargetACLVersion given did not match ACLVersion at the time of this call.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.8.ReplaceACLEntry

This action must be invoked by authorized entities (owners and possibly delegates). If successful, it replaces one ACL entry with a new value. That modification of the ACL causes the ACL Version to be changed. If the new ACL entry is in the same index location as the one being replaced, then NewACLVersion should be the new ACLVersion. See section 2.11.3 for details about ACLVersion.

The ACL editing Actions may be omitted from a DeviceSecurity instance if that device declares that it has no ACL (see section 2.9.3),

2.10.8.1.Arguments

Argument(s)	Direction	relatedStateVariable
TargetACLVersion	IN	A_ARG_TYPE_string
Index	IN	A_ARG_TYPE_int
Entry	IN	A_ARG_TYPE_string
NewACLVersion	OUT [R]	A_ARG_TYPE_string

[R] = RetVal

For details of TargetACLVersion and NewACLVersion, see section 2.11.3.

The ACL entry is an XML element, as described in section 4.2, encoded for transmission as a string – see section 3.2.

The Index into the ACL starts at 0 and counts entries. The first entry is 0, the second is 1, etc.

2.10.8.2.Effect on State

If the action invocation is successful, then the indicated entry in the device ACL is replaced by the entry given in the call. On that change, the ACL Version is changed. The nature of the change up to the device manufacturer and is not to be assumed by any control point. For example, one can use the BASE64 encoding of the hash of the ACL as the ACL Version.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.8.3.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control
501	Action Failed	See UPnP Device Architecture section on Control
602	Not Implemented	Optional action, not implemented
701	Not Authorized	The action in question was not issued by an authorized party (in this case, an owner of the device)
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
772	Entry does not exist	The indicated ACL entry does not exist.
773	Malformed entry	The offered ACL entry has a format error
774	Incorrect ACLVersion	The TargetACLVersion given did not match ACLVersion at the time of this call.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.9.FactorySecurityReset

This action must be executed by an owner of the device.

FactorySecurityReset is intended to return the device to initial conditions, once the device is disconnected from the network (or, perhaps, on next power-up). This is designed to support sale of a device to another party. The reason for waiting until network disconnect or power-up reset to take effect is to prevent some attacker who has learned the device's TakeOwnership secret from taking ownership of the device in the time interval between FactorySecurityReset and disconnection.

2.10.9.1.Arguments

There are no arguments.

2.10.9.2.Effect on State

If signed by an owner of the device, this action causes the device's ACL to be cleared and all but the caller to be erased from the ownership list. The timing of the removal of the last owner (thus enabling the TakeOwnership action) is up to the device manufacturer, but that action should not occur while the device is still connected to the network, if the manufacturer is concerned about leaving a window of attack open. For example, the last owner could be removed immediately, if the device first cuts itself off from the network and remains off the network until the next power-up. Note that if the device's TakeOwnership secret is sufficiently strong (for example, 8 or more random characters that are unique for each individual device), then it may not be necessary for a device to automatically go offline after FactorySecurityReset. It is good security practice to do so, but this behavior is not mandatory.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.9.3.Errors

errorCode	errorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
701	Not Authorized	The action in question was not issued by an authorized party (in this case, an owner of the device)
711	Signature Failure	The signature failed to verify
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.10.GrantOwnership

Some devices will allow multiple owners. Those that do must support this action. If this action is supported, then it must be invoked by an owner of the device.

The effect of this action is to add the indicated key to the list of owners, assuming it is not already listed and there is memory for it. If there is no more room in the ownership table, an error (751) is returned and the ownership table is not changed.

2.10.10.1.Arguments

Argument(s)	Direction	relatedStateVariable
HashAlgorithm	IN	A_ARG_TYPE_string
KeyHash	IN	A_ARG_TYPE_base64

The HashAlgorithm identifies the hash algorithm used for the key hash. Currently, the only algorithm supported is SHA1.

The KeyHash is a BASE64 encoded binary string, which is the hash of the respective control point or security console's signing key [cf., section 4.7], for example:

```
dRDPBgZzTFq7Jl2Q2N/YNghcfj8=
```

2.10.10.2.Effect on State

If the action is properly authorized and if the indicated key hash is not already a listed owner, then it is added to the list of owners. Otherwise, there is no change and the appropriate error return is given.

The device's ACL itself is not affected by this action, but there is an implicit effect. Every owner is treated as if it had `<all/>` permissions, by virtue of being listed in the owner table, and therefore does not need an entry in the device's ACL.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.10.3.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not authorized	The action was not properly authorized
711	Signature Failure	The signature failed to verify.
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
751	Out Of Memory	Insufficient memory for an additional owner
765	Already present	The indicated key is already an owner
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.11.RevokeOwnership

Devices that allow multiple owners must offer this action. This action must be invoked only by a present owner of the device. The owner must not be the same as the one being revoked. If the last owner wants to revoke his or her own ownership, then the action FactorySecurityReset must be used.

2.10.11.1.Arguments

Argument(s)	Direction	relatedStateVariable
HashAlgorithm	IN	A_ARG_TYPE_string
KeyHash	IN	A_ARG_TYPE_base64

The HashAlgorithm identifies the hash algorithm used for the key hash. Currently, the only algorithm supported is SHA1.

The KeyHash is a BASE64 encoded binary string, which is the hash of the respective control point or security console's signing key [cf., section 4.7], for example:

```
dRDPBgZzTFq7Jl2Q2N/YNghcfj8=
```

2.10.11.2.Effect on State

If the action is properly authorized and the indicated key is listed as an owner of the device, then that key hash is removed from the ownership table.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.11.3.Errors

errorCode	errorDescription	Description
402	Invalid Args	See UPnP Device Architecture section on Control.
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not authorized	The action was not properly authorized
711	Signature Failure	The signature failed to verify.
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
763	May not delete self	The key specified is the same key used to authorize this action. The caller may not delete itself via this action. Use FactorySecurityReset instead.
764	No such entry	The indicated key is not listed as an owner.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

2.10.12.ListOwners

This action may be access controlled through the normal device ACL, at the discretion of the device manufacturer.

The function of the action is to return a list of the key hashes of all owners of the device.

2.10.12.1.Arguments

Argument(s)	Direction	relatedStateVariable
ArgNumberOfOwners	OUT [R]	NumberOfOwners
Owners	OUT	A_ARG_TYPE_string

[R] = RetVal

If properly authorized, the action returns the number of owners of the device and the hashes of their keys. The list of owners is in the form of an XML element, properly escaped to be returned as a string (see section 3.2):

```
<Owners>
<hash><algorithm>SHA1</algorithm><value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value
></hash>
. . .
</Owners>
```

2.10.12.2.Effect on State

None

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.10.12.3.Errors

errorCode	errorDescription	Description
501	Action Failed	See UPnP Device Architecture section on Control.
602	Not Implemented	Optional action, not implemented
701	Not authorized	The caller was not authorized to perform this action.
711	Signature Failure	The signature failed to verify.
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence or SequenceNumber is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.11. Relationships among Actions

A security-aware device will be discovered using SSDP, by various Control Points and Security Consoles. It may or may not disclose all of its characteristics at that time, but it will disclose at least its own public keys and expose at a minimum the GetPublicKeys and TakeOwnership actions.

We assume here a device that has no generally available actions. In that case, the device must first be dealt with by a Security Console and only later by Control Points. If a device has generally available actions, then those are made available at all times and are not affected by the operation of DeviceSecurity.

2.11.1. Relationships among Actions invoked by Security Console

A Security Console (SC) discovers a device using SSDP and calls **GetPublicKeys** to get the device's confidentiality public key. With that key and under direction from a human user (who must be in possession of a secret (password) known by that device), the Security Console calls **GetLifetimeSequenceBase** followed by **TakeOwnership**, to acquire the ability to edit the device's Access Control List (ACL). It might also call **GetAlgorithmsAndProtocols**, to verify that it has algorithms in common with the device.

ACL editing usually involves initial calls to **GetACLSizes**, **GetDefinedPermissions**, **ReadACL** and maybe **GetDefinedProfiles**, followed by one or more of **AddACLEntry**, **DeleteACLEntry** or **ReplaceACLEntry**. Through that ACL editing, control points will have been granted access to the device.

If the Security Console operator wants to share ownership of the device (i.e., the permission to edit the ACL and the ownership list), the SC can call **GrantOwnership**. If the SC wants to revoke a prior grant of co-ownership, then it can call **RevokeOwnership**. At any time, it can call **ListOwners**, to get a list of current device owners.

If the owning SC has created ACL entries with expiration dates and the device offers the **SetTimeHint** action (implying that it has no internal clock), the SC might call **SetTimeHint** periodically. The net result is a very low resolution clock. The actual period between calls to **SetTimeHint** is up to the SC (probably the SC operator) and depends on how much accuracy of expiration times that operator desires. Frequencies of update on the order of 1 day should not be unusual.

A Security Console can grant rights to another SC, without granting it co-ownership. That second SC can then grant rights on that device to various Control Points it knows about. It grants those rights by certificate, since it is not allowed to edit the ACL. If the device offers space for a certificate cache, as determined by the return values of **GetACLSizes**, the second SC might plant those certificates in the device's certificate cache, using **CacheCertificate**.

2.11.2. Relationships among Actions invoked by normal Control Point

A control point (CP) that desires to operate a secured device must first have been given permission to do so by some Security Console. Assume here that that has happened, as described in section 2.11.1.

A control point should first call **GetPublicKeys** and **GetAlgorithmsAndProtocols** for the target device. If **GetAlgorithmsAndProtocols** permits a "NULL" encryption algorithm, then the CP does not need to encrypt all actions, otherwise it does need to.

Assuming no need to encrypt an action, the CP must still digitally sign it in order to prove authorization. SOAP messages could be digitally signed by public key algorithm, using the CP's private signing key. However, that is inefficient, especially for real-time operation of devices with limited processing power. Therefore, the CP should first set up session keys.

The CP generates keys at random and calls **GetLifetimeSequenceBase** followed by **SetSessionKeys** to communicate those to the device, along with a session ID that is meaningful to the CP. The CP receives, in return, a session ID meaningful to the device. That session ID is then used in the <ds:KeyInfo> field of the XML-Signature signature block to indicate which symmetric signing key was used in that signature. A well-behaved CP that knows it has stopped using a device, might call **ExpireSessionKeys**, in order to free memory on that device.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If the CP is empowered for this device by certificate instead of ACL entry (which it would know by having received certificates from a Security Console (see SecurityConsole:1)), then it will include those certificates in any XML-Signature <ds:KeyInfo> block, unless it can first cache them at the device via a call to **CacheCertificate**.

If the CP needs to encrypt an action, either because the device demands that for all actions or because the CP knows that the action invocation or one of its arguments is confidential, then it can use **DecryptAndExecute**, with existing session keys. In that case, the CP builds a signed SOAP message as if it were to be sent in the clear and then encrypts that XML document.

2.11.3. ACLVersion

The ACLVersion value is used in ACL reading or editing (sections 2.10.4, 2.10.5, 2.10.7, 2.10.8) and is affected by AddACLEntry (section 2.10.6). It is a value that is associated with a particular ACL content. [For example, it could be the BASE64 encoding of the hash of the ACL, in whatever form the device stores it.] The ACLVersion value serves the same purpose as a locking mechanism to resolve conflicts in case there are multiple simultaneous editors of the ACL, but without forcing us to create lock and unlock actions, timeouts to resolve cases when someone forgets to do an unlock action, etc.

The value of the ACLVersion is irrelevant. No SC should read that value and attempt to learn anything from it. Rather, it should be treated as an opaque string that must be presented in some ACL editing commands. If it matches the current ACLVersion in the device, then that action is allowed to proceed. If it does not match, the action fails and the SC needs to do another ReadACL to learn the new state of the ACL.

In normal operation, this need to do another ReadACL should almost never occur. Race conditions rarely happen. However, that rarity also leads to the most serious bugs unless the race condition is correctly handled by the code.

The suggestion that the ACLVersion be the hash of the ACL as stored in the device satisfies the full purpose of the ACLVersion – namely that when the SC submits a correct ACLVersion, the device can know that the SC's own copy of the ACL matches the device's copy. This lets the device know that the SC made its editing decision based on correct data (the ACL state). If the two copies differ, then the two ACLVersion values must differ. How this is achieved is up to the device manufacturer. The hash of the ACL contents is offered as one way to achieve this.

The ACLVersion returned to the SC after an edit assumes that the caller's version of the ACL will be the same as the device's. That is, after WriteACL, that the device keeps the ACL as the caller presented it; after ReplaceACLEntry, that the new entry is in the same slot as the old entry; after DeleteACLEntry, that the ACL entries after the deleted one were moved up in the array to close the gap. If the actual behavior of the ACL edit is not this, then the SC will not have the same ACL as the device and it must NOT be given a correct ACLVersion. In this case, the ACLVersion returned should be the empty string.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2.12. 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. These are the most commonly used errors, although there are other error codes defined and used in some of the actions.

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	The caller was not authorized to perform this action.
711	Signature Failure	The signature failed to verify.
712	Signature Missing	This action must be signed.
714	Invalid Sequence	SessionSequence is incorrect, probable replay attack.
715	Invalid Control URL	The control URL in the signature block did not match the Control URL used to deliver this action.
781	No Such Session	The signature key ID does not correspond to a valid session.
800-899	*TBD*	*(Specified by UPnP vendor.)*

When an action defined by some other device type is secured, it can return the following errors, in response to errors in the security sections of those action messages.

errorCode	errorDescription	Description
606	Action not authorized	The action requested requires authorization and the sender was not authorized.
607	Signature failure	The sender's signature failed to verify.
608	Signature missing	The action requested requires a digital signature and there was none provided.
609	Not encrypted	This action requires confidentiality but the action was not delivered encrypted.
610	Invalid sequence	SessionSequence is incorrect, possible replay attack.
611	Invalid control URL	The controlURL within the <freshness> element does not match the controlURL of the action actually invoked (or the controlURL in the HTTP header).
612	No such session	The session key reference is to a non-existent session. This could be because the device has expired a session, in which case the control point needs to open a new one.
800-899	*TBD*	*(Specified by UPnP vendor.)*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3. Supporting Information

3.1. Glossary

ACL: Access Control List – a list of ACL Entries

ACL Entry: a data record listing one principal, a set of permissions and, optionally, validity dates

Authentication: The second element of IAA (Identify, Authenticate, Authorize). In this document, the word "authentication" refers to verification that a message or message portion was signed by a particular key (either asymmetric or symmetric).

CBC: Cipher Block Chaining – a mode of encryption that hides the existence of repetitions of some plaintext block. CBC is standard for encryption of blocks of data longer than one plaintext block (where a block is typically 64 or 128 bits in length).

Component: generic term for something that is a control point, a device or both

Control point: a component that discovers devices, initiates SOAP actions, subscribes to events, etc.

Delegate: a Control Point or Security Console to which some privilege has been delegated.

Device: a component that advertises itself via SSDP, receives and acts on SOAP actions, publishes events to subscribed control points, etc.

DSig: Digital signature element in an XML document – see references in 3.2.

IV: an initialization vector, for a CBC mode encipherment

Owner: a principal permitted to edit the device's ACL

Principal: a public signing key, represented by its cryptographic hash except in a digital signature when it appears in full. The signing key "speaks for" the device or person that controls it.

Security ID: the SHA-1 hash of a public key. When printed in full, this Security ID should be encoded as 8 groups of 4 BASE32 digits each. Since BASE32 is not a standard representation, it is defined in section 3.3 of this spec.

Security Console: an application that allows the user to manage the security aspects of Devices and Control Points in a UPnP network.

3.2. XML Strings as UPnP Arguments

The UPnP V.1 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. i.e. 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 '&'

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3.3. BASE32 Encoding

There are large binary byte strings, such as random strings, public key values, symmetric key values and hash values that are used throughout UPnP Security. These values are communicated, within XML, as BASE64 throughout. There is one case when such a value is exposed to a user, and that is the Security ID (the hash of the key of a control point or device). For display to a user, BASE32 encoding is specified. This appears more familiar to a user, since the alphabetic characters are all upper case (as in product registration codes) and digits that are easily confused with letters are omitted. Strictly speaking, UPnP specs do not deal with user interfaces. However, correct functioning of UPnP Security depends on the user's ability to compare two copies of a Security ID, produced from the same underlying byte string by code from two different vendors. Therefore, that encoding to be used for printing or display must be standardized.

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 are 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 ambiguities.

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

```
193d 9354 ca84 f119 d9ee c17b c307 8c71 8a7b a70c
```

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.

3.4. Namespaces

In XML, an element name is not just the local part but that part combined with a namespace ID (explicit or default) to form a qualified name. XML processing requires that one deals with qualified names rather than merely local parts. See the UPnP architecture document for more details.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4. Data Structures

XML data structures are defined below for an Access Control List (ACL) (section 4.2) and Authorization Certificate (cert) (section 4.4.1. These data structures are used frequently in the actions described in this document.

4.1. Namespaces

In what follows, a namespace for the manufacturer is indicated, since a manufacturer defines permissions in his or her namespace or in the namespace of the working committee defining the DCP for that device.

4.2. Access Control List (ACL) Structure

An ACL is a collection of entries. Each ACL entry is communicated as an XML element of the form (with white space added for readability here, but not needed and therefore not desired in the computer-to-computer text). [The ACL will presumably be held in device memory in some parsed form, different from the wire format, and the ACL is not a signed construct like a certificate, so the presence or absence of white space is strictly a matter of transmission efficiency.]

```
<entry>
    <subject> {<hash>, <name> or <any/>} </subject>
    <may-not-delegate/>
    <access> {mfgr def'd permission elements or <all/>} </access>
    <valid> {optional <not-before> and/or <not-after> elements} </valid>
</entry>
```

where permissions are expressed as XML elements. In the simplest case, these will be without any internal structure, although device manufacturers are free to use parameters inside these elements if parameterization is necessary. Dates/times in validity limits are expressed in the following form, so that devices comparing dates/times need only do string comparisons of single tokens (contiguous non-white-space character strings). [See section 4.2.1 for details.] For example:

```
<not-before>2001-10-23T05:17:32Z</not-before>
<not-after>2003-12-31T23:59:59Z</not-after>
```

while a `<hash>` element includes the name of the hash algorithm and the hash value (base64 encoded), as for example:

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

The `<subject>` element specifies the principal being granted authority on this device. If the subject is a hash, then the principal is a specific public key and the hash value is the hash of that public key. If the subject is `<any/>`, then the permissions granted in this entry apply to all control points, whether they sign their command messages or not.

The syntax of a `<name>` element is, for example:

```
<name>
  <hash>
    <algorithm>SHA1</algorithm>
    <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value>
  </hash>
  <text>name text</text>
</name>
```

where "text" is the case-sensitive text of the name. If a device manufacturer's intention is to have a name that is case insensitive, then that manufacturer are responsible for transforming that name into all one case. The hash is of the key of the person (or device or application) that defined the name. That hash identifies the namespace in which the name is defined. Names are translated to hashes or other names by name definition certificates (section 4.4.2) and in that manner a named group is populated. A device processing a use of a name does not

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

need to be aware of the entire group, only of the membership status of the particular key that is claiming some relevant permission. Therefore, no device is likely to see an entire group's definitions.

It is possible for a device to define other <subject> types. The three given here are required, but one might define a different <subject> type if that device or device class has some special form of authentication. For example, one device might allow authentication by physical touch or by having a communication come in over a dedicated USB connector and might define a <subject> type to represent that authentication.

If present, the optional <may-not-delegate/> is a flag that implies that the subject may not delegate this right further. Although this is provided in the definition of the ACL and certificate, the Working Committee has reviewed its uses and decided that use of this flag in an ACL entry or certificate is probably unwise. That is because an individual who wants to delegate rights on to someone else will find some way to do that, e.g., by sharing his or her private key with someone else. As long as delegation is permitted by default, if someone gets access to a resource it is likely to be by his or her own private key with a full certificate chain showing the path of that delegation. All of that information can be available in an audit log, but if delegation is prohibited and is therefore achieved by sharing of private key, this ability to audit behavior is thwarted.

The <access> element is required and gives the permission(s) being granted to the subject. The element <all/> may be used to represent all defined permissions.

The <valid> element is optional and, if present, gives time limits on the grant of authorization. If present, it can include one or both of the <not-after> and <not-before> elements. Since an ACL entry can be deleted at will, a validity interval is not often needed. However, if some grant of access is to be for a specific time interval and the human defining the ACL does not desire to be forced to delete that entry at the expiration time, the time limit can be included at entry definition time. The use of time limits implies a source of time for the device. [See SetTimeHint in section 2.9.5.]

If an ACL entry does have an expiration time, <not-after>, and the entry expires, it is up to the device manufacturer whether to remove that entry from the ACL (and modify ACLVersion appropriately), or leave it in place.

4.2.1. Note on date and time format: ISO 8601

The date and time format used here is conformant with ISO 8601, but simplified. Dates and times must be expressed in the form: yyyy-mm-ddThh:mm:ssZ (indicating UTC).

ISO 8601 allows a number of options for specification of the time, including different levels of precision and different time zones. In UPnP Security, one must use only the one format, precision and time zone given, so that date and time values can be compared as text strings without the need to parse the date and time.

4.3. Owner List

Each device has a list of Owners. The entries in the owner list are hashes of signing keys. This list is effectively an ACL on the ACL. It is also an extension of the ACL, giving each owner <all/> permissions. It is reported via ListOwners as the following XML element giving full <hash> details as shown in the example below, but can be stored as just the list of binary hash values, especially if the device supports only one hash algorithm (SHA-1, to start with).

```
<Owners>
<hash><algorithm>SHA1</algorithm><value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value
></hash>
<hash><algorithm>SHA1</algorithm><value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value
></hash>
. . .
</Owners>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.4. Certificates

There are two forms of certificates defined for UPnP Security: an authorization certificate and a name definition certificate. In the text below, these are shown with white space to make them easier to read. However, when they are transmitted from computer to computer they shall be in canonicalized form (cf., section 5.2.2), with no white space.

Certificates are signed using <ds:Signature> detached signature form, as defined by XML-Signature, and should be generated and transmitted in exc-C14N form. Since a certificate is meant only for computer consumption, it is recommended that all white space be removed although the examples here show white space for human readability.

Note that the us:Id must be unique over the document in which it appears and that document may contain an arbitrary number of other certificates. Therefore, it is good practice to use a globally unique us:Id for that ID value.

This specification defines <ds:Signature> forms for public-key signatures and for session-key (symmetric-key) signatures [section 5.2]. Either can be used to sign a certificate, although the public-key form is more general. A certificate does not include a <freshness> element, since it is not a message. It has a <valid> element instead. If one wants to think of a certificate as a message, it is one addressed "to whom it may concern" and meant to be delivered multiple times and at an arbitrary time in the future (subject to any constraint imposed by the <valid> element).

- **Public-key:** A public-key signed certificate is generally verifiable, since the key used to verify it (the public key of a key pair) can be shared an arbitrary number of times without damaging its security. Therefore, this is the most general form of certificate signature. However, public-key verification operations take much more time to execute than symmetric-key operations, so there is a performance disadvantage with public-key signatures.

- **Symmetric-key:** A symmetric-key signed certificate can be verified only by a device holding a copy of the key used. Sharing the same symmetric key among more than the two endpoints of a session is bad key hygiene, so this security specification does not provide for that mode of operation. As a result, a certificate that is intended to be used by multiple devices (e.g., a group name certificate or an authorization certificate specifying multiple <device> targets), should be signed by public-key. Others can be signed by symmetric key, gaining a significant performance advantage, assuming those session keys will live as long as the certificate is supposed to be active. Note: a session key is indicated by a key ID private to the device doing the verification. Should some other device attempt to verify such a signature and happen to have a session key of that ID, the verification will fail because those keys will be different.

4.4.1. Authorization Certificate

An authorization certificate for UPnP is the equivalent of a signed ACL entry. It differs from an ACL entry in that the issuer needs to be specified as does the target device. In addition, a certificate is likely to have a validity element, while an ACL entry is unlikely to. E.g.,

```
<entry>
      <subject>
        <hash><algorithm>SHA1</algorithm>
          <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value></hash>
      </subject>
      <access xmlns:mfgr="…"> <mfgr:display/> <mfgr:play/> </access>
</entry>
```

might become

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
<cert us:Id="…" xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1">
      <issuer>
        <hash><algorithm>SHA1</algorithm>
          <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value></hash>
      </issuer>
      <subject>
        <hash><algorithm>SHA1</algorithm>
          <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value></hash>
      </subject>
      <tag>
       <device>
        <hash><algorithm>SHA1</algorithm>
          <value>2jmj7l5rSw0yVb/vlWAYkK/YBwk=</value></hash>
       </device>
       <access xmlns:mfgr="…"> <mfgr:display/> <mfgr:play/> </access>
      </tag>
      <valid>
        <not-after>2002-10-01T17:00:00Z</not-after>
      </valid>
</cert>
<ds:Signature> … </ds:Signature>
```

The public signature verification key will be carried in the XML-Signature <ds:KeyInfo> element of the <ds:Signature>. Other references to it, e.g., in `<issuer>`, will be as the hash of the key in order to save space. (See 4.7).

4.4.2. Name Definition Certificate

Each name definition certificate adds one key (or one whole group) to a named group. A key is added by referring to its hash. A named group is added by referring to its name.

The format for a name definition certificate is, for example:

```
<cert us:Id="…" xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1">
      <define>
        <name>
          <hash><algorithm>SHA1</algorithm>
            <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value></hash>
          <text>Name Text</text>
        </name>
      </define>
      <subject>
        <hash><algorithm>SHA1</algorithm>
          <value>dRDPBgZzTFq7Jl2Q2N/YNghcfj8=</value></hash>
      </subject>
      <valid>
        <not-after>2002-10-01T17:00:00Z</not-after>
      </valid>
</cert>
<ds:Signature> … </ds:Signature>
```

The <subject> can be either a <hash> (specifying one key) or a <name>. Note that there is no <tag> field. This certificate does not grant any authorization. It merely adds a member to a group. That group must be granted authorization, by name, in either an ACL entry or an authorization certificate.

The certificate must be signed by the key that hashes to the hash value given in the <define><name> as the namespace owner. No other key is allowed to add names to some key's namespace.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

4.5. Permission Language

The permission field of a certificate, <tag>, and of an ACL entry, <access>, consists of a sequence of XML elements. Five special elements of a permission field are available to device manufacturers to use. One of these is mandatory. The others are optional.

4.5.1. <all>

This element stands for the set of all possible XML elements. In other languages it might be expressed as "*". <all> is mandatory to implement.

4.5.2. <set>

This optional element lists a set of possible XML elements. The <access> element is implicitly a <set>, but <set> can be used anywhere within a <tag>. For example,

```
<tag>
      <device>
            <hash><algorithm>SHA1</algorithm>
                  </value>2jmj7l5rSw0yVb/vlWAYkK/YBwk=</value>
            </hash>
      </device>
      <access xmlns:mfgr="…">
            <mfgr:fileAccess>
                  <file>C:\My Documents\temp.doc</file>
                  <set>
                        <mfgr:read/>
                        <mfgr:delete/>
                        <mfgr:create/>
                  </set>
            </mfgr:fileAccess>
      </access>
</tag>
```

4.5.3. <elt>

This element is an escape for a text set element. That is, the set

```
<set> <elt> 10 </elt> <elt> 24 </elt> </set>
```

is the set consisting of two strings, "10" and "24". If this had been written

```
<set> 10 24 </set>
```

then by the rules of XML there would be only one set element, the string "10 24".

4.5.4. <prefix>

This optional element has one text datum. It stands for the set of all possible text data items that start with the given text. For example,

```
<prefix> c:\from-net\documents\upnp\ </prefix>
```

might refer to all pathnames and therefore files below the pathname given (in this case, presumably UPnP documents fetched from the network).

4.5.5. <range>

This optional element refers to a set of possible text items, in a specific format, and within a specific range. This is the least likely permissions element to be used, but is included in order to cover any desires to specify access control policies based on times of day or some numeric control value (e.g., a maximum volume some control point is allowed to set on a TV).

```
<range> {format} {limit}* </range>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
{format}:: <numeric> | <date> | <time> ;
{limit}:: <low>"minimum exclusive value"</low> |
            <low-equal>"minimum inclusive value"</low-equal> |
            <high>"maximum exclusive value"</high> |
            <high-equal>"maximum inclusive value"</high-equal> ;
```

For example,

```
<access xmlns:mfgr="…"> <mfgr:volume>
      <range>
            <numeric>
              <low-equal>0</low-equal>
              <high>9</high>
            </numeric>
      </range>
      </mfgr:volume>
</access>
```

might allow the subject to set the volume to any value between 0 and 8, inclusive.

Times must be in device local time in the format HH:MM:SS, for example:

```
23:59:59
```

Dates must be in device local time in the format yyyy-mm-ddThh:mm:ssZ, for example:

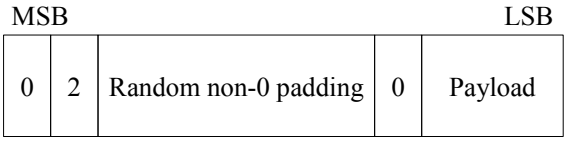
```
2001-12-29T23:59:59Z
```

Notice that dates and times in those formats can be compared by normal string comparison, for the purpose of the `<range>` element.

4.6. RSA Encryption Padding

The TakeOwnership and SetSessionKeys actions use public key cryptography for confidentiality, rather than symmetric session keys.

The padding used is also known as PKCS#1 (V1.5). The process of doing that padding is shown in Figure 1, below.



- **0:** a single byte of all 0 (0x00)
- **2:** a single byte with the value 2 (0x02)
- **Random non-0 padding:** random bytes to fill out the block, with no bytes of 0. (See the text for details.)
- **Payload:** the value being encrypted. This varies depending on use. (See the text for details.)

Figure 1: RSA encryption padding layout

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

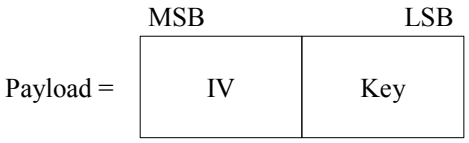
The purpose of this padding is to encode the operation type (2 = encrypt) and the length of the payload. One discovers the length of the payload, after decryption, by scanning the raw decrypted value, which is exactly the size of the RSA modulus, from left to right, starting with the 3rd byte (the first byte of random non-0 padding), looking for the first byte of 0. What is to the right of that 0 byte is the payload.

There are two ways to detect errors in the decryption operation just from this padding. If the most significant two bytes are not 0 and 2 respectively, then there was an error. Also if the payload has an improper length (a length of 0 or a length greater than the buffer provided to the RSA decrypt operation for holding the decrypted payload), then there was an error. A standard cryptographic API will probably have error codes for these two conditions. These errors, if reported back to the caller (the attacker), have been used in an attack on RSA encryption using PKCS#1 and those attacks are addressed in section 4.6.3.

The non-0 random padding has bits which are 0, of course, but has no bytes that are 0. This padding can be generated by taking bytes from a random (or good pseudo-random) source and using them in the padding if they are non-0 and discarding any that are 0.

4.6.1. SetSessionKeys

SetSessionKeys must create a temporary (single use) session key and communicate that to the receiver. This is accomplished by embedding the session key (bulk key) along with an IV (initialization vector, for CBC mode) in a block that is encrypted with the public key. That IV and session key become the payload for the padded RSA block as shown in Figure 1. This payload will be significantly less than the size of the RSA modulus. See Figure 2, below, for details.



• **IV:** an initial value for CBC mode, the size of a cipher block in the symmetric cipher (e.g., 128 bits for AES)

• **Key:** the key for the symmetric cipher (e.g., 256 bits for full strength AES)

• All bytes of these two values are in network standard byte order (also known as “big endian”). That is, the most significant byte is to the left in each value.

Figure 2: EncipheredBulkKey Payload

4.6.2. TakeOwnership

TakeOwnership communicates a value small enough that it is encrypted directly by the public key, using padding as shown in Figure 1. The HMAC value, H, being communicated by TakeOwnership is the Payload in Figure 1. See section 2.10.1 for details of the computation of that HMAC value, H.

4.6.3. Counteracting attacks on PKCS#1 V 1.5 padding

As noted above, there are attacks based on PKCS#1 V1.5 because it creates error conditions that might be reported. If those errors are reported to the attacker, this allows information about the private key to be revealed. This class of attack based on PKCS padding has the attacker generating false ciphertexts (or modifying legitimate

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

ciphertexts) and submitting those to the device. There are three possible outcomes from such a modification of the ciphertext:

1. The decrypted block will not start with bytes 0x00 0x02. This should happen in all but 1 in 65536 times.

2. The decrypted block will start with 0x00 0x02, but the length of payload will be wrong (because the first 0x00 after the header will be in the wrong place).

3. The header will be correct and the payload will be the correct length, but the bytes of the payload will be wrong. In the case of UPnP Security, when this happens, TakeOwnership will fail as if the caller had the wrong secret password and SetSessionKeys will most likely give an error about incorrect XML. The actual choice of error code will depend on the implementation.

If the device returns errors that allow the attacker to tell which of these three cases has occurred, then the attacker can, with enough probes, learn bits of the device's private key.

The solution to this is to keep the attacker from learning which of these cases has occurred.

One simple way to achieve that result is to force all three errors into error #3. That is, if one has a cryptographic library that returns error codes corresponding to cases #1 and #2 above, then instead of giving that error back to the caller in any form, the code should generate a completely random payload and then use that payload as if there had been no error. This will produce an error, as in case #3, but the attacker will not have learned anything from this use of PKCS#1.

4.6.4. Historical note about padding and padding attacks

This use of PKCS#1 was not in early versions of this spec. The padding used there was non-standard but provably avoided all forms of this kind of attack. PKCS#1 is now being used because it is so completely pervasive in the industry that some implementations of cryptographic libraries offer nothing else and, in particular, offer no opportunity for implementing the padding mechanism that was specified in the original versions of this spec.

Because PKCS#1 opens us to a class of attacks, a procedure is specified in section 4.6.3 to counteract that class of attacks.

4.7. Public Keys and their hashes

A public key is expressed as an XML structure of the form, for example:

```
<RSAKeyValue>
  <Modulus>xA7SEU+e0y…</Modulus>
  <Exponent>AQAB</Exponent>
</RSAKeyValue>
```

That is, the modulus and exponent are each expressed as the BASE64 encoding of the long-integer value. That value, after BASE64 decoding, is a byte string in network-byte-order (most significant byte first). These values are unsigned large integers. However, some cryptographic routines use multi-precision routines that assume all multi-precision integers are signed.

Implementations of DeviceSecurity use the hash of this public key, which means they need to take the hash of some canonical encoding. It is to be expected that many implementations will keep public keys in a binary form for normal use, and not to keep the XML by which they were communicated. However, for later use (e.g., in TakeOwnership) the implementation needs to generate the XML from that binary form. That generation needs to be the same for every implementation. Therefore, the following process for generating the XML from a public key is to be followed:

1. Each multi-precision integer is in network byte order (most significant byte first) and if the high order bit of the unsigned byte string is a 1, that string is prefixed with one byte of 0. In other words, the multi-precision number is the minimal expression of the value, assuming that it will be interpreted as a signed integer.

2. Multi-precision values are encoded as BASE64, by the rules given in section 4.9, below.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3. The encoding is XML with no white space.

4. The hash includes the <RSAKeyValue> open and close tags.

5. There are no namespace declarations or prefixes used in this structure. This <RSAKeyValue> is conformant with <ds:RSAKeyValue> except for namespace indication.

In the examples given in this document, white space has been added (carriage return, line feed and spaces) to increase the XML's readability in this document. When actually transmitted as an argument or as a component of some other XML structure, there is to be no white space in the <RSAKeyValue> element.

4.8. Symmetric cipher mode and padding

This version of UPnP Security has have chosen AES with 128-bit keys and 128-bit block size for the mandatory symmetric cipher and CBC (Cipher Block Chaining) as the cipher mode.

When a block cipher is applied to an arbitrary byte string, one must handle the case that the byte string is not an integral number of blocks long. Therefore, one uses a padding mechanism to pad out plaintext to an integral number of blocks. This padding must be removed after decryption of the ciphertext byte string. Therefore, it is common practice to put the count of padding bytes into the padding itself, in a known place – specifically into the last byte. The rest of the padding is arbitrary. Note that since there must be at least one byte of padding (in order to carry the count of number of padding bytes), a plaintext block that is already an integral number of cipher blocks will be increased in size by a whole cipher block.

The following conventional padding algorithm is used:

A plaintext string of bytes that needs to be encrypted has padding appended to it. There is a minimum of 1 byte of padding and a maximum of the same number of bytes as the encryption algorithm block. [That maximum padding size is used when the plaintext is already an integral number of encryption algorithm blocks in length.] The last byte of the padding is the number of bytes of padding. The other bytes of the padding are arbitrary. If the sender has access to a good source of random bytes, it can use random bytes. Otherwise, it can use any bytes it chooses, including constant values.

This padding is removed after decryption by going to the last byte of the decrypted byte string and reading from that byte the number of padding bytes that were added. That many bytes is then removed from the end of the decrypted byte string. The bytes that remain are the original plaintext.

4.9. Canonical BASE64 Encoding

BASE64 encoding is defined in RFC1521. The concern of that RFC is the encoding of binary values for successful transport through e-mail handlers. Since some of those e-mail handlers would break lines that were longer than 80 characters, RFC1521 specifies that all BASE64 encodings should be limited to 76 character maximum line lengths.

UPnP does not use e-mail transport but rather HTTP, an 8-bit channel with no line-length restrictions. The breaking of lines is difficult to canonicalize, especially since a BASE64 value will be used within XML elements that may add to the line length.

For the purposes of UPnP Security, canonical BASE64 will include no white space (no spaces, tabs or end-of-line characters – in short, no ASCII character codes less than or equal to 32 decimal).

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

In addition, when one encodes the last bytes of binary into BASE64, there are three possibilities. BASE64 encodes blocks of three binary bytes into four ASCII characters. If there are fewer than three bytes left to be encoded, BASE64 encodes whatever bytes there are and uses the character "=" to indicate how many bytes were actually encoded, as by the following table:

# binary bytes	# "=" characters	Example
3	0	`Y21l`
2	1	`eHk=`
1	2	`WA==`

When building that last group of 4 encoded characters in all but the 3-byte case, the last encoded character contains bits from the last binary byte and what should be the next byte. For canonical form BASE64, that "next byte" must be a byte of 0. That "next byte" could be random and the BASE64 string would convert back to the same binary string, but for the purpose of building canonical BASE64 those choices must be limited.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

5. Theory of Operation

UPnP consists of a number of different protocols, for different functions. There is SSDP for discovery, SOAP for control, GENA for event subscription and reporting and HTTP for presentation pages and bulk data transfer. The UPnP Security Working Committee has addressed all of these protocols, but DeviceSecurity speaks only to securing SOAP control actions.

The SOAP interface to a security-aware device is protected in the following ways:

1. **Authorization:** any control point must be authorized to perform a secured action before that action will be accepted from that control point. Some actions may be permitted to <a:any/>, that is to all control points whether they sign control messages or not, but that is up to the device owner through actions at the Security Console (assuming the manufacturer declares the action to be access controlled).

2. **Integrity protection:** any signed message is integrity protected as a side-effect of its signature.

3. **Replay prevention:** once a secure session is established, via SetSessionKeys, there is a sequence number available for replay prevention. The sequence number is in two parts: an arbitrary string (possibly a long random number) generated by the device as part of creating the session, and an unsigned 32-bit integer that counts from 0. For replay prevention of the non-session actions, TakeOwnership and SetSessionKeys, the LifetimeSequenceBase state variable provides a non-repeating value over the lifetime of the device.

4. **Confidentiality:** When confidentiality is required, DeviceSecurity provides an action, DecryptAndExecute, that carries a Ciphertext as one argument, and inside that Ciphertext there is a complete UPnP message of some other action. The other action is executed, if it is authorized, and any reply from that action is encrypted with keys from the same session to return to the caller as the reply from DecryptAndExecute. This mechanism was necessary in order to keep from doing damage to the UPnP Device Architecture V. 1.0. However, it is useful in certain edge cases and is likely to remain even if the V1 Device Architecture were to change to allow other forms of encryption.

In order to enforce authorization, two things are required.

1. Each authorized control point must digitally sign SOAP messages.

2. Some application with a user interface must allow the device/network owner to configure security policy on devices on that owned network. This application is called a Security Console. It operates both as a control point (calling some of the actions described in this document) and as a device (see SecurityConsole:1).

For performance reasons, most operations will use session keys: symmetric keys (e.g., AES or SHA1-HMAC) to be used instead of the device's or control point's public keys.

5.1. Access Control Lists and Certificates

Each device is responsible for its own access control decisions. Those decisions are made according to a security policy, established in turn by the device's owner, through the actions of a Security Console. The security policy for a device is encoded in the form of three data structures, of decreasing power (of granted access):

1. the list of device owners

2. an Access Control List (ACL)

3. authorization certificates

The list of device owners is the list of (the hashes of) those signature keys that are permitted to edit the ACL of the device. By default, each of these keys is given total permission to operate the device as well.

The ACL lists those signature keys (of a CP, usually, but sometimes of a SC) that are being granted less than full ownership privilege. Each ACL entry lists a signature key (indicated by its hash) and one or more permissions being granted that key. These permissions are defined by the device manufacturer and correspond to some set of actions that the signature key is allowed to perform. The mapping from defined permission to set of permitted

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

actions is left up to the manufacturer to specify. An ACL entry might also contain limiting specifications: validity date/time limits and, possibly, the prohibition against further delegation. These limitations are not expected in normal ACL entries, but they are available for advanced SC-Device interaction.

There are times when permission cannot be granted by editing an ACL entry. For example, some SC might have been granted only limited permission at a device and might want to grant that permission to some CP or some SC might be an owner of a device that has no memory for ACL storage. The SC would then issue a certificate to grant that permission to the CP. It is possible for the user interface for this certificate generation to be indistinguishable from the user interface for ACL editing, so that the user would not need to know details of the mechanisms involved.

Certificates must be communicated to the device that is doing the access control test. They can be communicated directly, via a CacheCertificate call to the device itself, but they will probably be communicated to the control point being empowered, for that control point to pass along to the device. The control point can pass those certificates to the device via CacheCertificate or within an XML-Signature `<ds:KeyInfo>` element. The former has a performance advantage, in the case of repeated authorized actions, but may not be allowed due to device memory constraints. The communication of certificates from the Security Console to the Control Point is provided via the SecurityConsole:1 service.

5.1.1. ACL and Certificate Processing Model

Historically, some ACLs were processed in some order and can include statements such as “stop processing here”. The ACLs defined here follow the capability model, as do the certificates defined here. That is, each ACL entry (section 4.2) or certificate (section 4.4) grants some permission. No ACL entry or certificate can override some other entry’s permission. That is, this is a strictly monotonic permission granting model. Therefore, for example, when one adds a new ACL entry via AddACLEntry, that entry is always added at the end of the ACL.

In this model, a given person’s rights are the union of the rights of all of his ACL entries or certificates. This processing model for ACLs is necessary to make ACL entries have the same semantics as certificates.

Revocation of a grant of rights is accomplished in different ways with ACL entries and certificates. When a permission is granted by ACL entry, one can revoke that permission by editing or deleting that entry. When a permission is granted by certificate, the permission can be revoked only by having the certificate expire and not be renewed.

Since certificate revocation is by date and time and since some UPnP devices will not have calendar clocks, revocation of permissions granted by certificate is not to be considered a precise operation.

5.2. Signature block format

There must be no more than one <us:SecurityInfo> header element per SOAP actor. If there is more than one such element targeted at the device, the device should fail the message. This document does not specify the use of <us:SecurityInfo> for intermediary actors but does not preclude such a specification at a later date.

Actions that are access controlled must be digitally signed by the issuing control point, using XML-Signature format. That digital signature applies to two fields, a <Freshness> block, shown below, and the SOAP <body>. The <SecurityInfo> element, including the signature and the Freshness element, is carried in the SOAP header. The Freshness element exists to defeat replay attacks. A unique value is used for each message. That value has three parts:

1. a SequenceNumber, which is monotonically increasing through a session in each direction. It is a 32-bit value and that implies a maximum of 4 billion messages in a session before it expires and a new session must be created. Both CPs and devices keep their own sequence numbers, which are not necessarily related to each other. [Under this protocol, in which each transaction is initiated by a CP, a device is free to keep only one sequence number counter (the CP’s) and use that for its own sequence numbering while a CP should keep two (since a reply to a message might get lost in transit).] Message verification checks that the new sequence number is higher than the previous sequence number in this session from the other entity.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2. a SequenceBase, which is a non-repeating value generated by the device at the time the session is created. This value, perhaps a large random number, prevents replay between sessions.

3. a controlURL, the URL to which the call is being addressed. The called device should confirm that the URL given in this element matches the URL actually being called, thus preventing redirection attacks.

Note that the digest and signature values in the examples below were not calculated from this example, so they will fail to verify.

There are two forms of signature defined. The most common form should use a symmetric session key for signatures and would look like:

```
<SecurityInfo>
<Freshness xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1"
xmlns:us="urn:schemas-upnp-org:service:DeviceSecurity:1"
us:Id="Freshness"><SequenceBase>XXX. .
.XXX</SequenceBase><SequenceNumber>YYY. .
.YYY</SequenceNumber><controlURL>http://ZZZ. .
.ZZZ</controlURL></Freshness>
 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></CanonicalizationMethod>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1"></SignatureMethod>
        <Reference URI="#Body">
  <Transforms>
    <Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#></Transform>
  </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
            <DigestValue>SiGg1/kFmfx7aQ4XWq56rdUQfyo=</DigestValue>
        </Reference>
        <Reference URI="#Freshness">
  <Transforms>
    <Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#></Transform>
  </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
            <DigestValue>fYLMutGy/KVdDzsEPTqcYKxOVzM=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>tTFeDxEb9LIXoI8DFgu8bkzed6Q=</SignatureValue>
    <ds:KeyInfo>
        <KeyName>17</KeyName>
    </ds:KeyInfo>
 </Signature>
</SecurityInfo>
```

The KeyName above is a session key ID. For messages going to the device, it is the DeviceKeyID. For messages coming from the device to the CP, it is the CPKeyID. Note: according to the XML-Signature schema, <KeyName> is a string. The decimal value 17 is also a string, besides being a number.

In some cases, it is necessary to sign a message using a public key algorithm (e.g., in SetSessionKeys or TakeOwnership). In those cases, there is no sequence base or number, so freshness is established via the LifetimeSequenceBase. Replies to these messages are not signed by public key algorithms, so there is no device-to-CP freshness counter for public-key signatures. A public-key signature block would look like:

```
<SecurityInfo>
<Freshness xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1"
xmlns:us="urn:schemas-upnp-org:service:DeviceSecurity:1"
us:Id="Freshness"><LifetimeSequenceBase>XXX. .
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
.XXX</LifetimeSequenceBase><controlURL>http://ZZZ. .
.ZZZ</controlURL></Freshness>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo xmlns=”http://www.w3.org/2000/09/xmldsig#">
        <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#"></CanonicalizationMethod>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"></SignatureMethod>
        <Reference URI="#Body">
  <Transforms>
    <Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#></Transform>
  </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
            <DigestValue>SiGg1/kFmfx7aQ4XWq56rdUQfyo=</DigestValue>
        </Reference>
        <Reference URI="#Freshness">
  <Transforms>
    <Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#></Transform>
  </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
            <DigestValue>fYLMutGy/KVdDzsEPTqcYKxOVzM=</DigestValue>
        </Reference>
    </SignedInfo>

<SignatureValue>Tx3dGYKl8UWjx00Q+fE0aYKlMcr2UTO96shC/duR9xYkFY2za5UEVrf8o22
mBEq7LQg3LQF9L5EpLpChtXZEgQ==</SignatureValue>
    <KeyInfo>
        <KeyValue>
            <RSAKeyValue>

<Modulus>tPK7xYLJqm77saltSus77darlxIHHWNajVEdxlwV7YmlnUyp/plhKltFr1jXzozXfP
Wc3ZwN6JfpdbyDwlJ74Q==</Modulus>
                <Exponent>AQAB</Exponent>
            </RSAKeyValue>
        </KeyValue>
    </KeyInfo>
</Signature>
</SecurityInfo>
```

In both cases, the <SignatureValue> is computed over the <SignedInfo> block. As shown here, that block is formatted for easy reading in this document. As actually transmitted, no white space should be used. However, if there is any white space, the hash must include that white space, under the rules of Exclusive Canonicalization. Note: by the rules of section 5.2.2, the <us:Freshhess>, <ds:SignedInfo> and <soap:Body> must be in Exc-C14N form on output from any UPnP control point or intermediary that is conformant with this specification.

5.2.1. Sequence Numbering

Every message under a given key needs a unique sequence number. Otherwise, it is possible for there to be replay attacks.

UPnP Security defines a sequence number as two components: the SequenceBase and the SequenceNumber.

A message is considered fresh if the sender uses the correct SequenceBase for the session key indicated and if its SequenceNumber is greater than the last one used accepted from the sender. That is, a message with a very high sequence number but an invalid signature would not bump the sequence number memory.

The SequenceBase is a string, presumably random, generated by the device when a new set of session keys is established. The CP is generating those session keys and the device generates the SequenceBase so that only if both of them reuse matching old values will uniqueness of sequence numbering be violated.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Within one session, messages are differentiated from one another by SequenceNumber, which is a 32-bit value. This value starts at 0, when a session is created, and should increase with each message sent under that session.

If that SequenceNumber should reach 2^{32}, then the session should expire automatically (without requiring an ExpireSessionKeys action).

When a reply is signed, that reply needs a <Freshness> block. The SequenceNumber in that reply message should be greater than the SequenceNumber used in the last reply.

Within a given session, there is no set relationship between a SequenceNumber sent by a CP and a SequenceNumber sent by a device. SequenceNumber values from both entities should be strictly increasing, but need not increment by 1 or any other fixed delta. Note that if a device manufacturer chooses to keep only one sequence number in memory for both messages and replies and then increment that value, that procedure conforms to this requirement – as would an implementation that keeps two counters, one for messages and one for replies. A CP, however, must not assume that there is a single counter, in order to be fully general.

If an action is signed using a public key algorithm, the LifetimeSequenceBase is used as both the sequence base and number for freshness of that action's signature. This implies that one must fetch a new LifetimeSequenceBase before each public key operation.

5.2.2. Hashing and Canonicalization

In order to form a signature over an XML document, that document must be canonicalized. The appropriate canonicalization method for XML-Signature is Exclusive XML Canonicalization Version 1.0; W3C Recommendation 18 July 2002. This canonicalization does not apply to an entire document, but rather to signed portions of a document.

This method of canonicalization can be very expensive, when performed at the receiver, so some small devices may have difficulty doing that work. To accommodate those devices, the sender (or re-sender) of a UPnP secured message must transmit messages whose signed elements are canonical according to Exclusive XML Canonicalization Version 1.0 so that a receiver can use the bytes as they arrived in the incoming message without doing any canonicalization. At the same time, the signature will verify while some other receiver whose stack automatically does Exclusive C14N will also verify signatures.

Part of canonicalization is the processing of line endings. It is strongly recommended that XML transmissions in security-aware UPnP not use any line endings or other formatting white space, especially in any signed elements. These are messages between two computers, not for human consumption. If one does use line endings, care must be taken to ensure that they conform to the Exclusive XML Canonicalization form.

The use of SOAP intermediaries is outside the scope of this document. An intermediary that does not output messages canonicalized as specified here will cause signatures not to verify. One might want to use DecryptAndExecute to tunnel signed messages through such an intermediary.

5.2.3. UPnP Certificate Transport

It is possible to communicate certificates from a Control Point to a Device only via calls to the action, CacheCertificate. However, that assumes that the Device has room for a cache of certificates. Caching certificates would be wise if they will be used multiple times to authorize the same control point.

If the Device has inadequate room for a certificate cache or if the control point needs to use certificates only once, then it makes sense to transmit the certificates within the XML-Signature `<ds:Signature>` element. XML-Signature provides a mechanism for that transport, within a `<ds:KeyInfo>` element.

UPnP certificates are strongly similar to SPKI certificates, but are not the same. They have slightly different syntax and are encoded in XML rather than canonical S-expressions. Therefore, a new element type is defined for carrying UPnP certificates (both authorization and group membership (name) certificates).

`<UPnPData> . . . </UPnPData>`

This element carries a certificate sequence. The resulting `<KeyInfo>` block (defined within xmlns=http://www.w3.org/2000/09/xmldsig#) is then, for example:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
<KeyInfo>
    <KeyName>17</KeyName>
    <UPnPData xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1">
        <Sequence>
          <cert> . . </cert><ds:Signature> . . </ds:Signature>
          <cert> . . </cert><ds:Signature> . . </ds:Signature>
          . .
        </Sequence>
    </UPnPData>
</KeyInfo>
```

Note that the XML-Signature specification allows `<KeyInfo>` to contain multiple elements, but each one needs to refer to the same signing key. These certificates are assumed to be offered in support of the signing key, completing a proof of its authorization to perform the action it is trying to do.

5.2.4. IDs for XML-Signature

XML-Signature is used for UPnP message authentication (see section 5.2) and for certificate signatures (see section 5.2.3).

An implementation must not trust information without verifying that it was signed by an appropriately authorized key.

The global us:Id attribute is defined to be of type ID and is therefore required to be unique within the document that contains it. Use of the us:Id attribute for XML-Signature references is the only method supported in DeviceSecurity.

5.2.5. Signature Processing Model

XML-Signature is used for UPnP message authentication (see section 5.2) and for certificate signatures (see section 5.2.3).

If a message is signed and the signature fails, then if the action requires authorization, that action must respond with a SOAP fault response.

If a certificate signature fails, then that certificate is not added to the device's cache of certificate bodies. This does not invalidate other certificates transmitted at the same time or the message in which the certificate was being carried. If the certificate is rejected, then it is likely to expect an action needing that certificate not to be successfully authorized and to fail on that account, but the failure would be failure of authorization rather than failure of certificate signature.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6. XML Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion> <!-- UPnP version 1.0 -->
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>GetPublicKeys</name>
      <argumentList>
        <argument>
          <name>KeyArg</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetAlgorithmsAndProtocols</name>
      <argumentList>
        <argument>
          <name>Supported</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetACLSizes</name>
      <argumentList>
        <argument>
          <name>ArgTotalACLSize</name>
          <relatedStateVariable>TotalACLSize</relatedStateVariable>
          <direction>out</direction>
        </argument>
        <argument>
          <name>ArgFreeACLSize</name>
          <relatedStateVariable>FreeACLSize</relatedStateVariable>
          <direction>out</direction>
        </argument>
        <argument>
          <name>ArgTotalOwnerListSize</name>
          <relatedStateVariable>TotalOwnerListSize</relatedStateVariable>
          <direction>out</direction>
        </argument>
        <argument>
          <name>ArgFreeOwnerListSize</name>
          <relatedStateVariable>FreeOwnerListSize</relatedStateVariable>
          <direction>out</direction>
        </argument>
        <argument>
          <name>ArgTotalCertCacheSize</name>
          <relatedStateVariable>TotalCertCacheSize</relatedStateVariable>
          <direction>out</direction>
        </argument>
        <argument>
          <name>ArgFreeCertCacheSize</name>
          <relatedStateVariable>FreeCertCacheSize</relatedStateVariable>
          <direction>out</direction>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
        </argument>
      </argumentList>
    </action>
    <action>
      <name>CacheCertificate</name>
      <argumentList>
        <argument>
          <name>Certificates</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetTimeHint</name>
      <argumentList>
        <argument>
          <name>ArgTimeHint</name>
          <relatedStateVariable>TimeHint</relatedStateVariable>
          <direction>in</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetLifetimeSequenceBase</name>
      <argumentList>
        <argument>
          <name>ArgLifetimeSequenceBase</name>
          <relatedStateVariable>LifetimeSequenceBase</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>SetSessionKeys</name>
      <argumentList>
        <argument>
          <name>EncipheredBulkKey</name>
          <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>BulkAlgorithm</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>Ciphertext</name>
          <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>CPKeyID</name>
          <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>DeviceKeyID</name>
          <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
        <argument>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
        <name>SequenceBase</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>out</direction>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>ExpireSessionKeys</name>
    <argumentList>
      <argument>
        <name>DeviceKeyID</name>
        <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
        <direction>in</direction>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>DecryptAndExecute</name>
    <argumentList>
      <argument>
        <name>DeviceKeyID</name>
        <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>Request</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>InIV</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>Reply</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
        <direction>out</direction>
        <retval/>
      </argument>
      <argument>
        <name>OutIV</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
        <direction>out</direction>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>TakeOwnership</name>
    <argumentList>
      <argument>
        <name>HMACAlgorithm</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>EncryptedHMACValue</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
        <direction>in</direction>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>GetDefinedPermissions</name>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      <argumentList>
        <argument>
          <name>Permissions</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>GetDefinedProfiles</name>
      <argumentList>
        <argument>
          <name>Profiles</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>ReadACL</name>
      <argumentList>
        <argument>
          <name>Version</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
        <argument>
          <name>ACL</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>WriteACL</name>
      <argumentList>
        <argument>
          <name>Version</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>ACL</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
        </argument>
        <argument>
          <name>NewVersion</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>out</direction>
          <retval/>
        </argument>
      </argumentList>
    </action>
    <action>
      <name>AddACLEntry</name>
      <argumentList>
        <argument>
          <name>Entry</name>
          <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
          <direction>in</direction>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      </argument>
    </argumentList>
  </action>
  <action>
    <name>DeleteACLEntry</name>
    <argumentList>
      <argument>
        <name>TargetACLVersion</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>Index</name>
        <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>NewACLVersion</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>out</direction>
        <retval/>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>ReplaceACLEntry</name>
    <argumentList>
      <argument>
        <name>TargetACLVersion</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>Index</name>
        <relatedStateVariable>A_ARG_TYPE_int</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>Entry</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>NewACLVersion</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>out</direction>
        <retval/>
      </argument>
    </argumentList>
  </action>
  <action>
    <name>FactorySecurityReset</name>
  </action>
  <action>
    <name>GrantOwnership</name>
    <argumentList>
      <argument>
        <name>HashAlgorithm</name>
        <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
        <direction>in</direction>
      </argument>
      <argument>
        <name>KeyHash</name>
        <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
            <direction>in</direction>
          </argument>
        </argumentList>
      </action>
      <action>
        <name>RevokeOwnership</name>
        <argumentList>
          <argument>
            <name>HashAlgorithm</name>
            <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
            <direction>in</direction>
          </argument>
          <argument>
            <name>KeyHash</name>
            <relatedStateVariable>A_ARG_TYPE_base64</relatedStateVariable>
            <direction>in</direction>
          </argument>
        </argumentList>
      </action>
      <action>
        <name>ListOwners</name>
        <argumentList>
          <argument>
            <name>ArgNumberOfOwners</name>
            <relatedStateVariable>NumberOfOwners</relatedStateVariable>
            <direction>out</direction>
            <retval/>
          </argument>
          <argument>
            <name>Owners</name>
            <relatedStateVariable>A_ARG_TYPE_string</relatedStateVariable>
            <direction>out</direction>
          </argument>
        </argumentList>
      </action>
  </actionList>
   <serviceStateTable>
      <stateVariable sendEvents="yes">
        <name>NumberOfOwners</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
        <name>LifetimeSequenceBase</name>
        <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
        <name>TimeHint</name>
        <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
        <name>TotalACLSize</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
        <name>FreeACLSize</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
        <name>TotalOwnerListSize</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
        <name>FreeOwnerListSize</name>
        <dataType>i4</dataType>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      </stateVariable>
      <stateVariable sendEvents="no">
        <name>TotalCertCacheSize</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
        <name>FreeCertCacheSize</name>
        <dataType>i4</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>
      <stateVariable sendEvents="no">
        <name>A_ARG_TYPE_int</name>
        <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
        <name>A_ARG_TYPE_boolean</name>
        <dataType>boolean</dataType>
      </stateVariable>
    </serviceStateTable>
</scpd>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

Annex A
(normative)

Device Security Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:schemas-upnp-org:service:DeviceSecurity:1"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="urn:schemas-upnp-
org:service:DeviceSecurity:1" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="CPList">
      <xs:annotation>
         <xs:documentation>See Section 2.3.1 in the SecurityConsole spec.
</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="hash" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="icon">
      <xs:annotation>
         <xs:documentation>See Section 2.5.1 in the SecurityConsole spec.
</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="mimetype" type="xs:string"/>
            <xs:element name="width" type="xs:nonNegativeInteger"/>
            <xs:element name="height" type="xs:nonNegativeInteger"/>
            <xs:element name="depth" type="xs:positiveInteger"/>
            <xs:element name="url" type="xs:anyURI"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="SignedNameList">
      <xs:annotation>
         <xs:documentation>See Section 2.5.2 in the SecurityConsole spec.
</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Names">
               <xs:complexType>
                  <xs:sequence minOccurs="0" maxOccurs="unbounded">
                     <xs:element name="Device" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="name" type="xs:string"/>
                              <xs:element ref="hash"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="CP" minOccurs="0"
maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="name" type="xs:string"/>
                              <xs:element ref="hash"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
                  </xs:sequence>
                  <xs:attribute name="Id" type="xs:ID" use="required"/>
               </xs:complexType>
            </xs:element>
            <xs:element name="Signature"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="hash">
      <xs:annotation>
         <xs:documentation>See Section 2.10.12 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="algorithm" type='OpenHashAlgorithm' >
            </xs:element>
            <xs:element name="value" type="xs:base64Binary"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="cert">
      <xs:annotation>
         <xs:documentation>See Section 4.4.1 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:choice>
            <xs:sequence>
               <xs:annotation>
                  <xs:documentation>Authorization
Certificate</xs:documentation>
               </xs:annotation>
               <xs:element name="issuer">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element ref="hash"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element ref="subject"/>
               <xs:element name="may-not-delegate" minOccurs="0"/>
               <xs:element name="tag">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="device">
                           <xs:complexType>
                              <xs:sequence>
                                 <xs:element ref="hash"/>
                              </xs:sequence>
                           </xs:complexType>
                        </xs:element>
                        <xs:element name="access">
                           <xs:complexType>
                              <xs:choice>
                                 <xs:element name="all" minOccurs="0"/>
                                 <xs:any namespace="##targetNamespace"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                              </xs:choice>
                           </xs:complexType>
                        </xs:element>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element ref="valid" minOccurs="0"/>
            </xs:sequence>
            <xs:sequence>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
               <xs:annotation>
                  <xs:documentation>Name Certificate</xs:documentation>
               </xs:annotation>
               <xs:element name="define" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="name">
                           <xs:complexType>
                              <xs:sequence>
                                 <xs:element ref="hash"/>
                                 <xs:element name="text" type="xs:string"/>
                              </xs:sequence>
                           </xs:complexType>
                        </xs:element>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
               <xs:element ref="subject"/>
               <xs:element ref="valid" minOccurs="0"/>
            </xs:sequence>
         </xs:choice>
         <xs:attribute name="Id" type="xs:ID" use="optional"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Sequence">
      <xs:annotation>
         <xs:documentation>See Section 5.2.3 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="cert" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Signature" minOccurs="0"
maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Owners">
      <xs:annotation>
         <xs:documentation>See Section 2.10.12 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="hash" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Keys">
      <xs:annotation>
         <xs:documentation>See Section 2.9.1 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Confidentiality">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="RSAKeyValue">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="Modulus"
type="xs:base64Binary"/>
                              <xs:element name="Exponent"
type="xs:base64Binary"/>
                           </xs:sequence>
                        </xs:complexType>
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="Signing" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="RSAKeyValue">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="Modulus"
type="xs:base64Binary"/>
                           <xs:element name="Exponent"
type="xs:base64Binary"/>
                        </xs:sequence>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="Supported">
   <xs:annotation>
      <xs:documentation>See Section 2.9.2 in the DeviceSecurity
spec.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Protocols" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="p" type="OpenProtocols"
minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="HashAlgorithms" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="p" type="OpenHashAlgorithms"
minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="EncryptionAlgorithms" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="p" type="OpenEncryptionAlgorithms"
minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="SigningAlgorithms" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="p" type="OpenSigningAlgorithms"
minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
   <xs:element name="SessionKeys">
      <xs:annotation>
         <xs:documentation>See Section 2.9.7 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Confidentiality">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Algorithm"
type="OpenEncryptionAlgorithms"/>
                     <xs:element name="KeyToDevice"
type="xs:base64Binary"/>
                     <xs:element name="KeyFromDevice"
type="xs:base64Binary"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="Signing">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Algorithm"
type="OpenSigningAlgorithms"/>
                     <xs:element name="KeyToDevice"
type="xs:base64Binary"/>
                     <xs:element name="KeyFromDevice"
type="xs:base64Binary"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="DefinedPermissions">
      <xs:annotation>
         <xs:documentation>See Section 2.10.2 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Permission" minOccurs="0"
maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="UIName" type="xs:string"/>
                     <xs:element name="ACLEntry">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:any namespace="##targetNamespace"
processContents="lax" minOccurs="0"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="FullDescriptionURL"
type="xs:anyURI"/>
                     <xs:element name="ShortDescription" type="xs:string"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Profiles">
      <xs:annotation>
         <xs:documentation>See Section 2.10.3 in the DeviceSecurity
spec.</xs:documentation>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Profile" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="UIname" type="xs:string"/>
                     <xs:element name="Definition">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:any namespace="##targetNamespace"
processContents="lax"/>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="FullDescriptionURL"
type="xs:anyURI"/>
                     <xs:element name="ShortDescription" type="xs:string"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="acl">
      <xs:annotation>
         <xs:documentation>See Section 2.10.4 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="entry" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="subject"/>
                     <xs:element name="may-not-delegate" minOccurs="0"/>
                     <xs:element name="access">
                        <xs:complexType>
                           <xs:choice>
                              <xs:element name="all" minOccurs="0"/>
                              <xs:any namespace="##targetNamespace"
processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                           </xs:choice>
                        </xs:complexType>
                     </xs:element>
                     <xs:element ref="valid" minOccurs="0"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Freshness">
      <xs:annotation>
         <xs:documentation>See Section 5.2 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="SequenceBase" type="xs:string"/>
            <xs:element name="SequenceNumber" type="xs:string"/>
            <xs:element name="controlURL" type="xs:anyURI"/>
         </xs:sequence>
         <xs:attribute name="Id" type="xs:ID" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="UPnPData">
```

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
      <xs:annotation>
         <xs:documentation>See Section 5.2.3 in the DeviceSecurity
spec.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Sequence"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="valid">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="not-before" type="xs:dateTime"
minOccurs="0"/>
            <xs:element name="not-after" type="xs:dateTime" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="subject">
      <xs:complexType>
         <xs:choice>
            <xs:element ref="hash" minOccurs="0"/>
            <xs:element name="name" minOccurs="0">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="hash"/>
                     <xs:element name="text" type="xs:string"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="any" minOccurs="0"/>
            <xs:any namespace="##targetNamespace" processContents="lax"
minOccurs="0"/>
         </xs:choice>
      </xs:complexType>
   </xs:element>

   <xs:simpleType name='OpenProtocols' >
      <xs:union memberTypes='Protocols xs:string' />
   </xs:simpleType>
   <xs:simpleType name='Protocols' >
   <xs:restriction base='xs:string' >
      <xs:enumeration value='UPnP' />
      <xs:enumeration value='TLS' />
      <xs:enumeration value='IPSEC' />
   </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name='OpenHashAlgorithms' >
      <xs:union memberTypes='HashAlgorithms xs:string' />
   </xs:simpleType>
   <xs:simpleType name='HashAlgorithms' >
   <xs:restriction base='xs:string' >
      <xs:enumeration value='SHA1' />
   </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name='OpenEncryptionAlgorithms' >
      <xs:union memberTypes='EncryptionAlgorithms xs:string' />
   </xs:simpleType>
   <xs:simpleType name='EncryptionAlgorithms' >
   <xs:restriction base='xs:string' >
      <xs:enumeration value='NULL' />
      <xs:enumeration value='RSA' />
      <xs:enumeration value='AES-128-CBC' />
   </xs:restriction>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    </xs:simpleType>

    <xs:simpleType name='OpenSigningAlgorithms' >
       <xs:union memberTypes='SigningAlgorithms xs:string' />
    </xs:simpleType>
    <xs:simpleType name='SigningAlgorithms' >
    <xs:restriction base='xs:string' >
       <xs:enumeration value='NULL' />
       <xs:enumeration value='RSA' />
       <xs:enumeration value='SHA1-HMAC' />
    </xs:restriction>
    </xs:simpleType>

</xs:schema>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex B
(informative)

Security Ceremonies

B.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.

B.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 B.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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

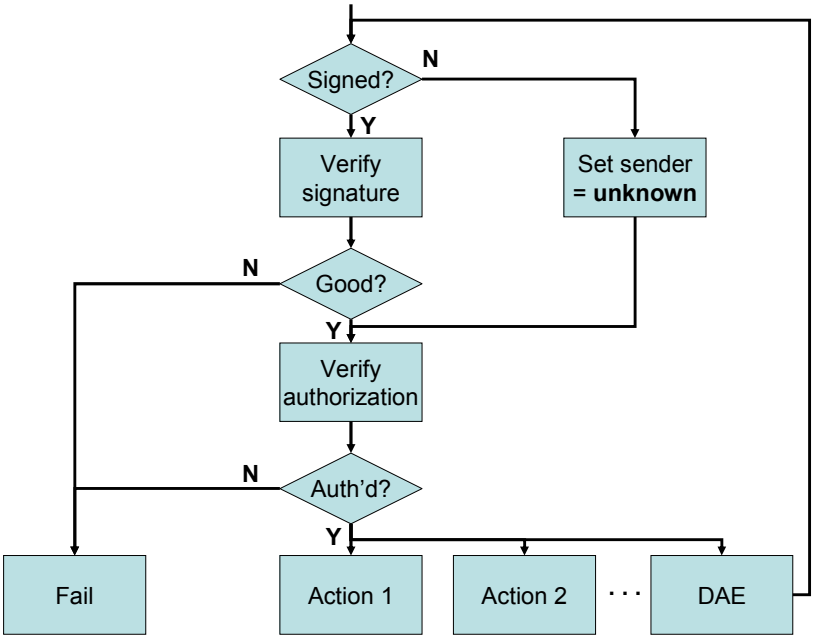


Figure B.1: Message Security Flowchart

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 B.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 B.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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.2.1 Security Policy Data

The message security processing described in Figure B.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 B.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.

B.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.

[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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.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 B.2, below.

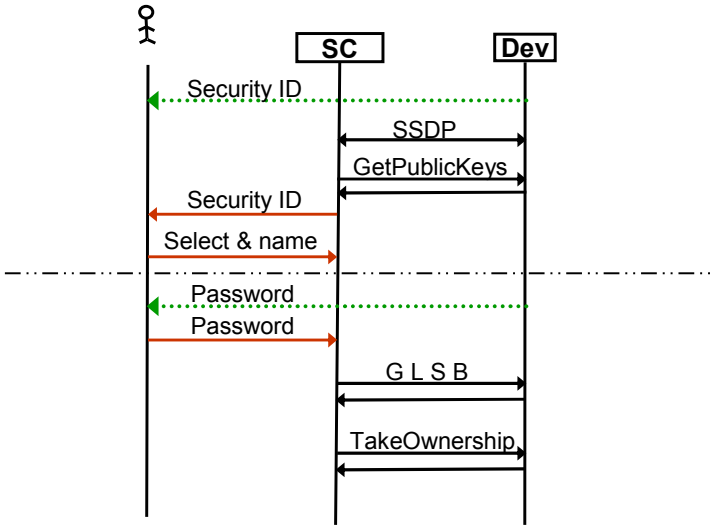


Figure B.2: Device discovery ceremony and TakeOwnership

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 B.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.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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 B.4.1 for more details about TakeOwnership and its options.]

B.3.1.1 How to read the ceremony diagram

Figure B.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)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.3.2 Discovery of a Secured CP or SC

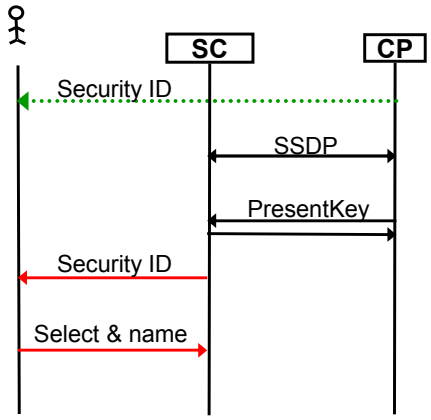


Figure B.3: Discovery of CP and SC nodes

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.

B.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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

B.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 B.4.3).

The default TakeOwnership process is described in section B.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 B.4, below. The security of this method depends on the cable's connecting exactly two entities – the device and the SC.

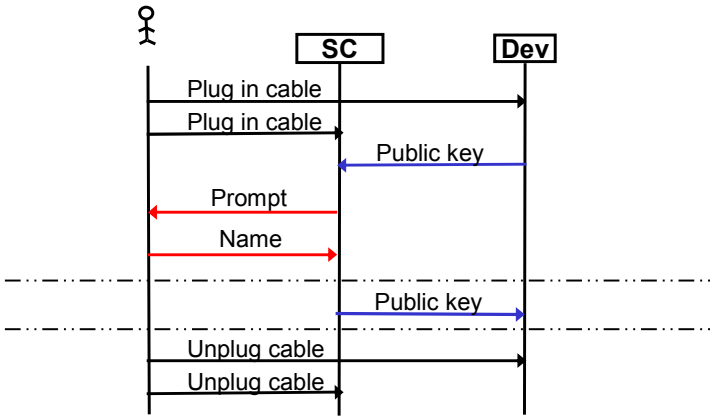


Figure B.4: Taking ownership via private cable

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.

B.4.1.1 Note on Figure B.4

Figure B.4 uses a color code, as the one exception to the general expectation set in section B.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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.4.2 ListOwners

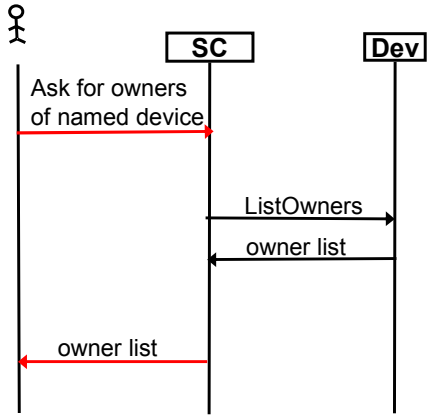


Figure B.5: ListOwners

The ListOwners action is invoked by the SC as the user's client and agent. The flow is as shown in Figure B.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 B.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 B.3). If there are owners listed in the owner list that have not been named, the SC can display those as raw SecurityIDs.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.4.3 GrantOwnership

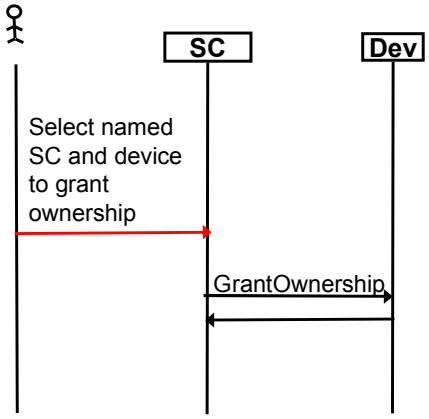


Figure B.6: GrantOwnership

Having named some other SC (see Figure B.3) and some target device (see Figure B.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.

B.4.4 RevokeOwnership

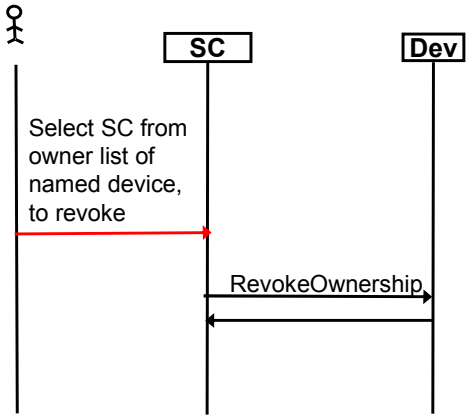


Figure B.7: RevokeOwnership

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

B.4.5 FactorySecurityReset

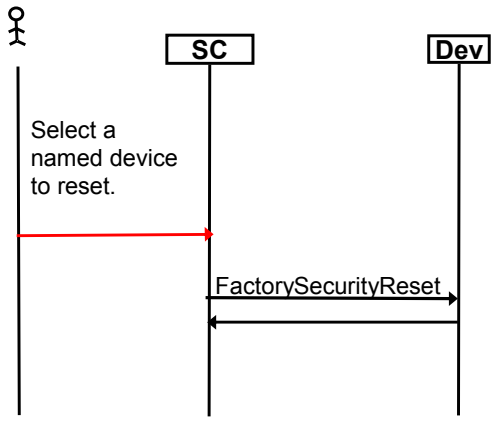


Figure B.8: FactorySecurityReset

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.

B.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 B.9, below.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

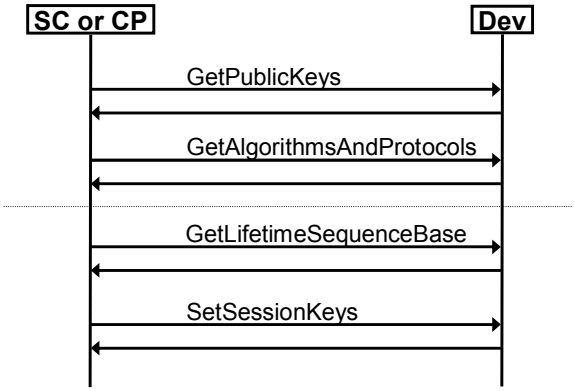


Figure B.9: Setting Session Keys

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.

B.6 ACL Editing

The main purpose of security ownership (see section B.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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

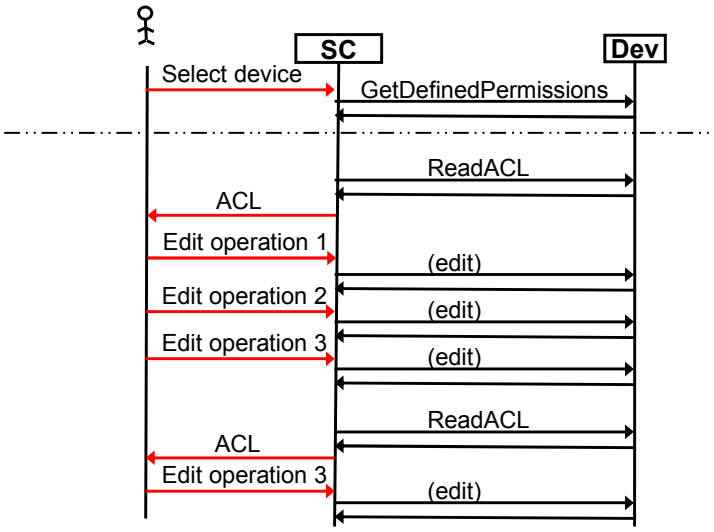


Figure B.10: ACL Editing

ACL editing is a user-driven operation, so we can not prescribe a full ceremony for it. Figure B.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.

B.7 Certificate caching

There are two purposes for a certificate in UPnP Security:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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 B.11, below.

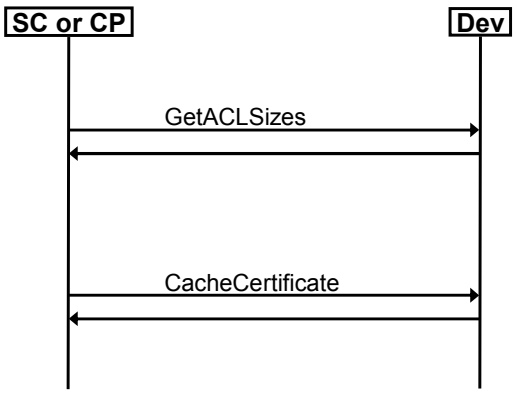


Figure B.11: Certificate caching on the device

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 B.12, below.

--`,,```,,,,````-`-,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

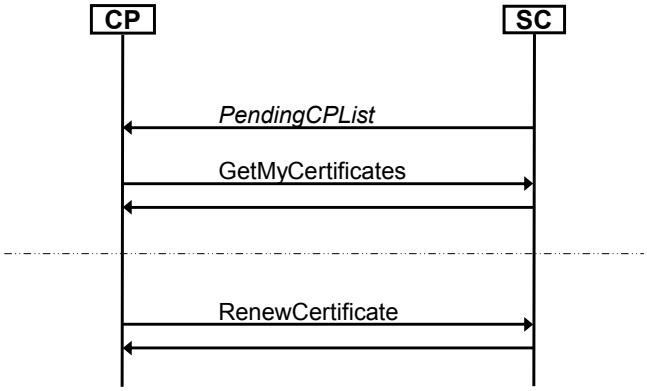


Figure B.12: Delivering certificates to the CP

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 B.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.

B.8 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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale