Video Services Forum (VSF)
Technical Recommendation TR-10-17

Internet Protocol Media Experience (IPMX): Control Plane Security

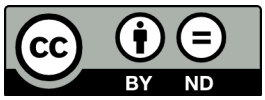
November 16, 2026 (v1.0)

DRAFT

© 2026 Video Services Forum

This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-nd/4.0/

or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.



http://www.videoservicesforum.org

INTELLECTUAL PROPERTY RIGHTS

RECIPIENTS OF THIS DOCUMENT ARE REQUESTED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT CLAIMS OR OTHER INTELLECTUAL PROPERTY RIGHTS OF WHICH THEY MAY BE AWARE THAT MIGHT BE INFRINGED BY ANY IMPLEMENTATION OF THE RECOMMENDATION SET FORTH IN THIS DOCUMENT, AND TO PROVIDE SUPPORTING DOCUMENTATION.

THIS RECOMMENDATION IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NONINFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS RECOMMENDATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS RECOMMENDATION.

LIMITATION OF LIABILITY

VSF SHALL NOT BE LIABLE FOR ANY AND ALL DAMAGES, DIRECT OR INDIRECT, ARISING FROM OR RELATING TO ANY USE OF THE CONTENTS CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION ANY AND ALL INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, LOSS OF PROFITS, LITIGATION, OR THE LIKE), WHETHER BASED UPON BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING NEGATION OF DAMAGES IS A FUNDAMENTAL ELEMENT OF THE USE OF THE CONTENTS HEREOF, AND THESE CONTENTS WOULD NOT BE PUBLISHED BY VSF WITHOUT SUCH LIMITATIONS.

DRAFT

BY-ND

Executive Summary

Internet Protocol Media Experience (IPMX) was created to foster the adoption of open standards-based protocols for interoperability over IP in the media and entertainment and professional audio/video industries. IPMX is based on the SMPTE ST 2110 standard and as such the VSF TR-10 suite of Technical Recommendations is a set of differences between SMPTE ST 2110 and IPMX.

This Technical Recommendation documents the requirements for an IPMX device with respect to the Control Plane Security.

Table of Contents

1 Introduction (Informative) 6
2 Contributors 6
3 About the Video Services Forum 6
4 Conformance Notation 7
5 Normative References 8
6 Definitions 8
7 Scope 9
7.1 NMOS Node 10
7.2 NMOS Registry 11
7.3 Peer to Peer Mode 11
8 TLS Communications and Cipher Suites 11
9 Registry Access Policy (RAP) 12
9.1 Unrestricted Registration 13
9.2 Restricted Registration 13
10 Node Access Policy (NAP) 13
10.1 Unrestricted Read Write 13
10.2 Unrestricted Read Only 14
10.3 Restricted Read Write 14
11 Restricted Access Authorization Mode (RAAM) 14
11.1 Mutual TLS authentication 15
11.2 Server TLS authentication with OAuth 2.0 authorizations 15
11.3 Mutual TLS authentication with OAuth 2.0 authorizations 15

DRAFT

CC BY-ND

12 Device Configuration Options 16
12.1 Registry Access Policy (RAP) 16
12.2 Node Access Policy (NAP) 16
12.3 Restricted Access Authorization Mode (RAAM) 16
12.4 OAuth 2.0 Audience Identification Mode (OAIM) 17
12.5 TLS Certificate Type (TCT) 17
12.6 TLS Server Certificate(s) (TSC) 17
12.7 Client Trusted CA(s) (CTCA) 17
12.8 Client Trusted Certificates Revocation List(s) (CTCRL) 17
12.9 TLS Client Certificate(s) (TCC) 18
12.10 Node endpoints Server Trusted CA(s) (NESTCA) 18
12.11 Node endpoints Server Trusted Certificates Revocation List (NESTCRL) 18
12.12 Control endpoints Server Trusted CA(s) (CESTCA) 18
12.13 Control endpoints Server Trusted Certificates Revocation List(s) (CESTCRL) 19
12.14 Additional Provisions: 19
12.15 NMOS Tags 20
13 IS-10 Authorizations General Provisions 20
13.1 IS-10 and BCP-003-02 20
14 IS-10 Authorizations specific requirements 21
14.1 Scope 21
14.2 Paths 21
14.3 Behavior 21
14.3.1 Time Synchronization 21
14.3.2 Public Keys 21
14.3.2.1 Authorization Server Metadata Endpoint 22
14.3.3 Access Token 23
14.3.3.1 Lifetime 23
14.3.3.2 Type and Algorithms 23
14.3.3.3 Grants 24
14.3.3.4 Validation 24
14.3.3.5 Security Failure Handling 28

DRAFT

CC BY-ND

14.3.3.6 Mutual TLS Client Certificate Binding 28
14.3.3.7 WebSocket 29
14.3.4 HTTP Status Codes 29
14.3.5 Examples (Informative) 30
14.3.5.1 Pseudocode (Informative) 31

DRAFT

CC BY-ND

1 Introduction (Informative)

IPMX, which stands for Internet Protocol Media Experience, is based on two families of specifications. The SMPTE ST 2110 Professional Media Over Managed IP Networks suite of standards for the transport of video, audio, and ancillary/control signals over IP networks, and the NMOS REST APIs from AMWA, which provide discovery, connection management, and control.

IPMX is an accessible, open standard that meets the needs of professional and consumer video and audio users in a wide variety of contexts while giving manufacturers and developers what they need to build low-latency, interoperable, IP based audiovisual products or applications.

This document covers the Control Plane Security requirements for IPMX. Other aspects of the IPMX system and their individual requirements are documented in other parts of this Technical Recommendation.

2 Contributors

The following individuals participated in the Video Services Forum IPMX working group that developed this technical recommendation.

Alain Bouchard (Matrox)	Jack Douglass (PacketStorm)	Phil Bernal (PB Technologies Group)
Adam Salkin (Diversified)	Jean Lapierre (Matrox)	Phil Nguyen (Nextera)
Amanda Douglass (Macnica)	Jean-Jacques Osytiguy (Matrox)	Raymond Hermans (Adeas)
Andre Testa (Matrox)	Jed Deame (Nextera)	Ryan Wallenberg (Cobalt)
Andreas Hildebrand (RAVENNA)	Jeroen Steendam (DekTec)	Steve Kolta (Meinberg)
Andrew Starks (Macnica)	Kenneth Jonassen (Bridge Technologies)	Thomas True (Nvidia)
Danny Pierini (Matrox)	Kyle Wilken (Cobalt Digital)	Tim Bruylants (intoPIX)
David Konz (plexusAV)	Nick Nicas (AT&T)	
Hung Lam (EvertzAV)	Paulo Francisco (EvertzAV)	

3 About the Video Services Forum

The Video Services Forum, Inc. (www.videoservicesforum.org) is an international association dedicated to video transport technologies, interoperability, quality metrics and education. The VSF is composed of service providers, users and manufacturers. The organization's activities include:

- providing forums to identify issues involving the development, engineering, installation, testing and maintenance of audio and video services;
- exchanging non-proprietary information to promote the development of video transport service technology and to foster resolution of issues common to the video services industry;
- identification of video services applications and educational services utilizing video transport services;
- promoting interoperability and encouraging technical standards for national and international standards bodies.

DRAFT

BY-ND

The VSF is an association incorporated under the Not For Profit Corporation Law of the State of New York. Membership is open to businesses, public sector organizations and individuals worldwide. For more information on the Video Services Forum or this document, please call +1 929-279-1995 or e-mail opsmgr@videoservicesforum.org.

4 Conformance Notation

Normative text describes elements of the design that are indispensable or contain the conformance language keywords: "shall," "should," or "may."

Informative text is potentially helpful to the user but not indispensable and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except the Introduction and any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed to conform to the document and from which no deviation is permitted.

The keywords "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; followed by formal languages; then figures; and then any other language forms.

DRAFT

BY-ND

5 Normative References

- VSF TR-10-1 Internet Protocol Media Experience (IPMX): System Timing and Definitions
- AMWA IS-10 version 1.0.1 - AMWA IS-10 NMOS Authorization Specification
- AMWA BCP-002-02 version 1.0.0 - AMWA BCP-002-02 NMOS Asset Distinguishing Information
- AMWA BCP-003-01 version 1.0.1 - AMWA BCP-003-01 Secure Communication in NMOS Systems
- AMWA BCP-003-02 version 1.0.0 - AMWA BCP-003-02 Authorization in NMOS Systems
- AMWA IS-04 version 1.3.3 - AMWA IS-04 NMOS Discovery and Registration Specification
- AMWA IS-05 version 1.1.1 - AMWA IS-05 NMOS Device Connection Management Specification
- AMWA IS-08 version 1.0.1 - AMWA IS-08 NMOS Audio Channel Mapping Specification
- AMWA IS-11 version 1.0.0 - AMWA IS-11 NMOS Stream Compatibility Management
- AMWA IS-12 version 1.0.1 - AMWA IS-12 NMOS Control Protocol
- AMWA IS-14 version 1.0.0 - AMWA IS-14 NMOS Device Configuration Specification
- AMWA MS-05-02 version 1.0.0 - AMWA MS-05-02 NMOS Control Framework
- Internet Engineering Task Force (IETF) RFC 4592: The Role of Wildcards in the Domain Name System available at https://www.ietf.org/rfc/rfc4592.txt
- Internet Engineering Task Force (IETF) RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage available at https://www.ietf.org/rfc/rfc6750.txt
- Internet Engineering Task Force (IETF) RFC 8414: OAuth 2.0 Authorization Server Metadata at https://tools.ietf.org/html/rfc8414 "

6 Definitions

The NMOS terms User, Controller, Registry, Node, Device, Source, Flow, Sender, Receiver are used as defined in the NMOS Glossary. For the purposes of this document, the terms, and definitions of VSF TR-10-1 and the following apply.

NMOS	AMWA Networked Media Open Specifications
AMWA	Advanced Media Workflow Association
BCP	Best Current Practice
CA	Certificate Authority
CESTCA	Control Endpoints Server Trusted CA
CESTCRL	Control Endpoints Server Trusted Certificate Revocation List
CN	Common Name
CRL	Certificate Revocation List
CTCA	Client Trusted CA
CTCRL	Client Trusted Certificate Revocation List
DNS	Domain Name System

DRAFT

BY-ND

DNS-SD	DNS Service Discovery
ECDSA	Elliptic Curve Digital Signature Algorithm
GTCRL	Global Trusted Certificate Revocation List
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IPMX	Internet Protocol Media Experience
IS	Interface Specification
JWT	JSON Web Token
mDNS	Multicast DNS
mTLS	Mutual Transport Layer Security
MS	Method Specification
NAP	Node Access Policy
NESTCA	Node Endpoints Server Trusted CA
NESTCRL	Node Endpoints Server Trusted Certificate Revocation List
NMOS	Networked Media Open Specifications
NTP	Network Time Protocol
OAIM	OAuth 2.0 Audience Identification Mode
PEM	Privacy Enhanced Mail
PFS	Perfect Forward Secrecy
PKCS	Public Key Cryptography Standards
PTP	Precision Time Protocol
RAAM	Restricted Access Authorization Mode
RAP	Registry Access Policy
RFC	Request for Comments
RSA	Rivest-Shamir-Adleman
SAN	Subject Alternative Name
SMPTE	Society of Motion Picture and Television Engineers
TCC	TLS Client Certificate
TCT	TLS Certificate Type
TLS	Transport Layer Security
TSC	TLS Server Certificate
VSF	Video Services Forum

DRAFT

7 Scope

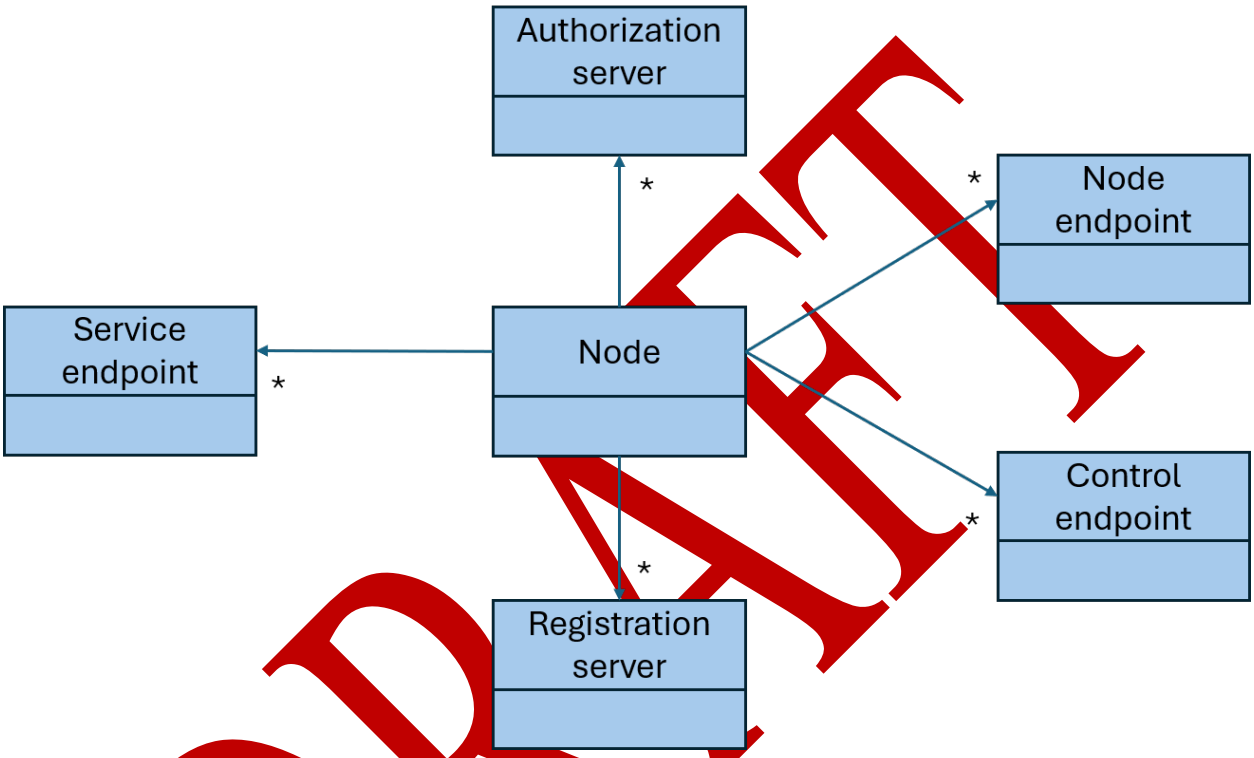
This Technical Recommendation does not redefine NMOS APIs or resource models; it specifies additional and/or overriding control-plane security requirements for the NMOS interfaces a Node exposes and consumes, as defined by the applicable AMWA NMOS specifications.

This Technical Recommendation is based on AMWA IS-10 and AMWA BCP-003-01. Where any requirement in this Technical Recommendation conflicts with a requirement in AMWA IS-10 or AMWA BCP-003-01, the requirement in this Technical Recommendation shall take precedence for any IPMX implementation claiming compliance with this Technical Recommendation.

BY-ND

A newly manufactured device shall implement the default values specified in this Technical Recommendation for all security-relevant configuration options for which defaults are defined. The scope of any vendor-specific mechanisms for a device to restore partially or fully the manufactured state is outside the scope of this Technical Recommendation.

7.1 NMOS Node



DRAFT

An NMOS Node comprises multiple Node API endpoints (IS-04) declared in the *api.endpoints* attribute of the Node's "node" resource. It also comprises several control API endpoints (IS-05, IS-08, IS-11, IS-12, IS-14, etc.), as declared in the *controls* attribute of the Node's "device" resources. Optionally, an NMOS Node exposes some service API endpoints and declares them in the *services* attribute of the Node's "node" resource. All those endpoints are in the scope of this Technical Recommendation.

Additionally, an NMOS Node listens to and serves other clients accesses from the network that are outside the scope of this Technical Recommendation.

An NMOS Node performs client accesses to several NMOS Registry's Registration API endpoints and IS-10 OAuth 2.0 endpoints. Accesses to those endpoints are in the scope of this Technical Recommendation.

Additionally, an NMOS Node performs other client accesses to network services that are outside the scope of this Technical Recommendation.

BY-ND

Note: Accesses from/to IS-05 transport streams (RTP, MQTT, WebSocket, etc.) are outside the scope of this Technical Recommendation which focuses on the standard NMOS TCP/IP based control plane, excluding transport streams that are covered in IS-05 or BCP-007 transport specific documents.

Note: Accesses to / from DHCP, PTP, NTP, DNS, mDNS services are not covered by this document. Similarly, 802.1x protection of Ethernet access is outside the scope of this Technical Recommendation.

7.2 NMOS Registry

The IS-04 Registration API shall not require the NMOS Nodes to use OAuth 2.0 authorizations. The IS-04 Registration API shall be secured using TLS with server authentication or mutual client-server authentication.

- *api_auth* of the Registry DNS-SD record shall be false.

7.3 Peer to Peer Mode

A Node operating in peer-to-peer mode (without an NMOS Registry) broadcasts its Node API information through mDNS which is not encrypted, providing an "Unrestricted Read Only" access to the broadcast information. mDNS payload content is out of scope of the Technical Recommendation. The Node's Node API follows the regular Node Access Policy (NAP).

A device shall provide a configuration option to turn off peer-to-peer mode to prevent any use of mDNS for strict customer security policies.

8 TLS Communications and Cipher Suites

An IPMX device shall comply with this Technical Recommendation. This Technical Recommendation is based on AMWA BCP-003-01 for secure communications; where requirements differ, this Technical Recommendation is authoritative. The following additional requirements apply:

Implementations should support TLS 1.3 (RFC 8446) and shall support TLS 1.2 (RFC 5246). While TLS v1.2 is supported for backward compatibility, implementations should use TLS v1.3 for enhanced security.

All TLS connections shall use cipher suites that provide Perfect Forward Secrecy (PFS), such as those based on ECDHE or DHE. An IPMX device shall support the ephemeral key exchange groups 25519 and secp256r1 and should support secp521r1 and 448.

Note: The cipher suite naming in this section follows the universal IANA/RFC nomenclature (e.g. TLS_AES_128_GCM_SHA256), which corresponds to the suites defined in AMWA BCP-

DRAFT

BY-ND

003-01. The curve names (e.g. 25519, 448) follow the nomenclature used in VSF TR-10-13 (PEP) and correspond to the X25519 and X448 groups in universal TLS nomenclature.

For TLS v1.2, an IPMX device shall support the cipher suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 and should support TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384. An IPMX device may support the TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 and TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 cipher suites.

An IPMX device may support the TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 cipher suites when required due to constrained-device performance or capability limitations. A CBC-mode cipher suite should not be used unless the `encrypt_then_mac` extension is successfully negotiated.

For TLS v1.3, an IPMX device shall support the cipher suite TLS_AES_128_GCM_SHA256 and should support TLS_AES_256_GCM_SHA384 and TLS_AES_128_CCM_SHA256. An IPMX device may support the TLS_CHACHA20_POLY1305_SHA256 cipher suite.

For state-of-the-art security, the use of GCM-based suites or TLS v1.3 is strongly recommended.

Only the cipher suites and key exchange groups listed as "shall", "should", or "may" in this section shall be used. All other cipher suites are prohibited.

9 Registry Access Policy (RAP)

An NMOS Registry operates in one of the following policies regarding access to the Registration of resources. All the NMOS registries accessible by an NMOS Node shall use the same access policy.

The AMWA IS-04 specification indicates in the “Registration API – DNS-SD Advertisement” section: “Multiple DNS-SD advertisements for the same API are permitted where the API is exposed via multiple ports and/or protocols.”. If multiple DNS-SD records are available for a Registry, some having the *api_proto* parameter set to “http” and some other having *api_proto* set to “https”, a device compliant with this Technical Recommendation shall select the DNS-SD record having the *api_proto* parameter set to “https” and *api_auth* set to false.

DRAFT

BY-ND

9.1 Unrestricted Registration

An NMOS Registry configured with that policy grants registration access to anyone.

If the *api_proto* parameter of the Registry DNS-SD record is “http”, the IS-04 Registration API is not protected by TLS.

If the *api_proto* parameter of the Registry DNS-SD record is “https”, the IS-04 Registration API is protected by TLS and an IPMX device shall authenticate the Registry using TLS server authentication. The *api_auth* parameter of the Registry DNS-SD record shall be false.

Unrestricted Registration shall be supported by all compliant IPMX devices.

9.2 Restricted Registration

An NMOS Registry configured with that policy grants registration access to those presenting a client certificate authorized for registration. An IPMX device and a Registry shall authenticate each other using TLS mutual authentication.

- *api_proto* of the Registry DNS-SD record shall be “https”
- *api_auth* of the Registry DNS-SD record shall be false.

Restricted Registration shall be supported by all compliant IPMX devices.

10 Node Access Policy (NAP)

An NMOS Node operates in one of the following policies regarding access to the Node’s resources and states. An IPMX device shall allow an administrator to configure the device to use one of the following policies, which default to “Unrestricted Read Write”. The policy applies to all incoming accesses to the Node’s endpoints and to all outbound client accesses performed by the Node.

10.1 Unrestricted Read Write

An IPMX device configured with that policy explicitly indicates that it is not compliant with this Technical Recommendation. In that specific case, the protocol is HTTP without TLS, and the device shall not claim compliance with this Technical Recommendation while so configured.

- Unrestricted read access shall be permitted to all clients.
- Unrestricted write access shall be permitted to all clients.

The Node's endpoints *protocol* and services *href* attribute indicates HTTP and the *authorization* attribute is false. The Device's control endpoints *href* attribute indicates HTTP and the *authorization* attribute is false.

DRAFT

BY-ND

10.2 Unrestricted Read Only

An IPMX device configured with that policy grants read only access to anyone but requires authorization for write access (accesses with side effects on the device state). This mode of operation is not allowed when OAuth 2.0 authorizations are used, in which case even read access shall be explicitly provided by the OAuth 2.0 authorizations.

- Unrestricted read access shall be permitted to all clients.
- Restricted write access shall be enforced by the NMOS Node in accordance with the configured RAAM policy.

The Node's endpoints *protocol* and services *href* attribute indicates HTTPS and the *authorization* attribute is set according to RAAM. The Device's control endpoints *href* attribute indicate HTTPS and the *authorization* attribute is set according to RAAM.

10.3 Restricted Read Write

An IPMX device configured with that policy grants read and write access to only those having the proper authorization. This mode of operation shall be supported by all IPMX devices. conforming to this Technical Recommendation.

- Restricted read access shall be enforced by the NMOS Node in accordance with the configured RAAM policy.
- Restricted write access shall be enforced by the NMOS Node in accordance with the configured RAAM policy.

The Node's endpoints *protocol* and services *href* attribute indicates HTTPS and the *authorization* attribute is set according to RAAM. The Device's control endpoints *href* attribute indicate HTTPS and the *authorization* attribute is set according to RAAM.

11 Restricted Access Authorization Mode (RAAM)

An NMOS Node applies access restrictions based on one of the following modes of operation: Mutual TLS authentication, Server TLS authentication with OAuth 2.0 authorizations, or Mutual TLS authentication with OAuth 2.0 authorizations.

Note: In the following sections, the expression “not supporting IS-10 or when an IS-10 OAuth 2.0 server is not available” refers to a configuration option of the device activating/de-activating the use of an OAuth2.0 server on the network. A device supporting IS-10 could have this option enabled or disabled.

DRAFT

BY-ND

11.1 Mutual TLS authentication

- **Shall be supported by all compliant IPMX devices not supporting IS-10 or when an IS-10 OAuth 2.0 server is not available.**
- Clients are authorized if properly authenticated.
- RSA, ECDSA, or both (ECDSA attempted first by client)
    - An IPMX device shall allow an administrator to configure the type of certificates to use as RSA, ECDSA, or both. The certificate type applies to both endpoint and client accesses, and to server and client certificates.
    - **RSA and ECDSA shall independently be supported by all compliant IPMX devices. Supporting both simultaneously is optional.**

The Node's endpoints *protocol* and services *href* attribute indicates HTTPS and the *authorization* attribute is set to false. The Device's control endpoints *href* attribute indicates HTTPS and the *authorization* attribute is set to false.

11.2 Server TLS authentication with OAuth 2.0 authorizations

- **Shall be supported by all compliant IPMX devices supporting IS-10 when an IS-10 OAuth 2.0 server is available.**
- Clients are authorized only by explicit OAuth 2.0 authorizations
- RSA, ECDSA, or both (ECDSA attempted first by client)
    - An IPMX device shall allow an administrator to configure the type of certificates to use as RSA, ECDSA, or both. The certificate type applies to both endpoint and client accesses, and to server and client certificates.
    - **RSA and ECDSA shall independently be supported by all compliant IPMX devices. Supporting both simultaneously is optional.**

The Node's endpoints *protocol* and services *href* attribute indicates HTTPS and the *authorization* attribute is set to true. The Device's control endpoints *href* attribute indicates HTTPS and the *authorization* attribute is set to true.

11.3 Mutual TLS authentication with OAuth 2.0 authorizations

- Clients are authorized if properly authenticated and have explicit OAuth 2.0 authorizations.
- RSA, ECDSA, or both (ECDSA attempted first by client)
    - An IPMX device shall allow an administrator to configure the type of certificates to use as RSA, ECDSA, or both. The certificate type applies to both endpoint and client accesses, and to server and client certificates.
    - **RSA and ECDSA shall independently be supported by all compliant IPMX devices. Supporting both simultaneously is optional.**

DRAFT

CC BY-ND

The Node's endpoints *protocol* and services *href* attribute indicates HTTPS and the *authorization* attribute is set to true. The Device's control endpoints *href* attribute indicates HTTPS and the *authorization* attribute is set to true.

12 Device Configuration Options

12.1 Registry Access Policy (RAP)

- Unrestricted Registration (default = HTTP) (value 0)
- Unrestricted Registration (HTTPS server authentication) (value 1)
- Restricted Registration (HTTPS mutual authentication) (value 2)
- **An IPMX device compliant with this Technical Recommendation shall support the Unrestricted and Restricted Registration modes.**

12.2 Node Access Policy (NAP)

- Unrestricted Read Write (default = HTTP) (value 0)
- Unrestricted Read Only (value 1)
- Restricted Read Write (value 2)
- **An IPMX device compliant with this Technical Recommendation shall support the Restricted Read Write mode.**

12.3 Restricted Access Authorization Mode (RAAM)

- Mutual TLS Authentication (default) (value 0)
- OAuth 2.0 Authorizations (value 1)
- Mutual TLS Authentication with OAuth 2.0 Authorizations (value 2)
- **An IPMX device compliant with this Technical Recommendation, not supporting IS-10 or when an IS-10 OAuth 2.0 server is not available, shall support Mutual TLS Authentication.**
- **An IPMX device compliant with this Technical Recommendation, supporting IS-10 and when an IS-10 OAuth 2.0 server is available, shall support OAuth 2.0 Authorizations.**

Note: The expression "not supporting IS-10 or when an IS-10 OAuth 2.0 server is not available" refers to a configuration option of the device activating/de-activating the use of an OAuth2.0 server on the network. A device supporting IS-10 could have this option enabled or disabled.

DRAFT

BY-ND

12.4 OAuth 2.0 Audience Identification Mode (OAIM)

- Serial Number (default) (value 0)
- TLS Server Certificate (TSC) Common Name (CN) or alternative DNS names. (value 1)
- Either (value 2)
- **An IPMX device compliant with this Technical Recommendation shall support all modes.**

12.5 TLS Certificate Type (TCT)

- RSA (default) (value 0)
    - RSA certificates shall use a minimum key size of 2048 bits.
- ECDSA (value 1)
    - ECDSA certificates shall use a minimum curve size equivalent to secp256r1 or 25519.
- Both (value 2)
- Shall be common to all certificates and Trusted CAs of the device.
- **RSA and ECDSA shall independently be supported by all compliant IPMX devices. Supporting both simultaneously is optional.**

12.6 TLS Server Certificate(s) (TSC)

- According to TLS Certificate Type (TCT) configuration
- Default: Vendor certificates or None (shall be configured to enable related configurations)
- May be common for the Node and Control endpoints or specific to each one.

12.7 Client Trusted CA(s) (CTCA)

- According to TLS Certificate Type (TCT) configuration
- Root or Intermediate CA(s)
- Default: Vendor Root CA(s)
- Used to validate the TLS Server Certificate of the NMOS Registry and the OAuth 2.0 Authorization Server.
- May be common for the Registration and Authorization accesses or specific to each one.
- Shall support at least two CTCA for each type of access to allow certificate re-provisioning.
- Also used to verify Certificate Revocation List CRL as per X.509

12.8 Client Trusted Certificates Revocation List(s) (CTCRL)

- According to TLS Certificate Type (TCT) configuration

DRAFT

BY-ND

- Signed by a Client Trusted CA(s)
    - If the server certificate is signed by intermediate CA, not the root directly, the CRL must be signed by the intermediate CA.
- Used to validate a TLS Server Certificate of the NMOS Registry and the OAuth 2.0 Authorization Server.
- Shall support at least two CTCRL to allow certificate re-provisioning.

12.9 TLS Client Certificate(s) (TCC)

- According to TLS Certificate Type (TCT) configuration
- Default: Vendor certificates or None (shall be configured to enable related configurations)
- May be common for the Registration and Authorization accesses or specific to each one.
- May be hidden from an Admin when the Restricted Access Authorization Mode does not include mutual authentication.

12.10 Node endpoints Server Trusted CA(s) (NESTCA)

- According to TLS Certificate Type (TCT) configuration
- Root or intermediate CA(s)
- Default: Vendor Root CA(s) or None (shall be configured to enable related configurations)
- Used to validate a TLS Client Certificate on the Node endpoints and services.
- May be hidden from an Admin when the Restricted Access Authorization Mode does not include mutual authentication.
- Shall support at least two NESTCA to allow certificate re-provisioning.

12.11 Node endpoints Server Trusted Certificates Revocation List (NESTCRL)

- According to TLS Certificate Type (TCT) configuration
- Signed by a Node endpoints Server Trusted CA(s)
    - If the client certificate is signed by intermediate CA, not the root directly, the CRL must be signed by the intermediate CA.
- Used to validate a TLS Client Certificate(s) on the Node endpoints and services.
- Shall support at least two NESTCRL to allow certificate re-provisioning.

12.12 Control endpoints Server Trusted CA(s) (CESTCA)

- According to TLS Certificate Type (TCT) configuration
- Root or intermediate CA(s)

DRAFT

BY-ND

- Default: Vendor Root CA(s) or None (shall be configured to enable related configurations)
- Used to validate a TLS Client Certificate on the Control endpoints.
- May be hidden from an Admin when the Restricted Access Authorization Mode does not include mutual authentication.
- Shall support at least two CESTCA to allow certificate re-provisioning.

12.13 Control endpoints Server Trusted Certificates Revocation List(s) (CESTCRL)

- According to TLS Certificate Type (TCT) configuration
- Signed by a Control endpoints Server Trusted CA(s)
    - If the client certificate is signed by intermediate CA, not the root directly, the CRL must be signed by the intermediate CA.
- Used to validate a TLS Client Certificate(s) on the Control endpoints.
- Shall support at least two CESTCRL to allow certificate re-provisioning.

12.14 Additional Provisions:

- The device shall provide a means to update trust material such that a valid existing trust configuration remains usable until a new configuration has been successfully validated.

- When an administrator changes the security policy configuration (e.g., NAP, RAP, RAAM, OAIM, TCT) or updates trust material (CAs, CRLs, certificates), new connections shall follow the new configuration immediately. Existing sessions shall continue with the configuration that was in effect when the session was established until they can safely be terminated.

- The device shall provide a means for an administrator or management system to retrieve the current effective values of the security policy configuration options described in this section (e.g., NAP, RAP, RAAM, OAIM, TCT) and to identify the trust material in use (e.g., certificates, CAs, CRLs) through non-sensitive metadata (such as subject/issuer names, validity dates, serial numbers, fingerprints, or equivalent identifiers). Private keys are out of scope for retrievable configuration and shall be write-only and shall not be retrievable. User interfaces may omit options that are not applicable under the current configuration.

- An implementation may combine the Client Trusted Certificates Revocation List (CTCRL), Node endpoints Server Trusted Certificates Revocation List (NESTCRL) and Control endpoints Server Trusted Certificates Revocation List (CESTCRL) into a single

DRAFT

BY-ND

Global Trusted Certificates Revocation List (GTCRL) by concatenating the various CRL, each signed by its respective CA. If a single CA is used across all CTCA, NESTCA and CESTCA, then a single CRL may be used.

- There shall be as many different CRL as there are when described as independent resources.
- All the CA(s) CTCA, NESTCA and CESTCA shall be considered during the validation of the CRL signature.

- Certificates and Private Keys are either transferred in PEM format or PKCS#12 encrypted with a strong password .p12 format.
  - When using the PEM format the transfer channel shall be secure.
  - When using the PKCS#12 format the transfer of the password shall be secure.

12.15 NMOS Tags

The following tags should be published in the Node's 'tags' attribute to indicate the configuration of the Node's security parameters: NAP, RAP, RAAM, OAIM, TCT.

urn:x-vsf:tag:tr-10-sec:nap-config/v1.0

urn:x-vsf:tag:tr-10-sec:rap-config/v1.0

urn:x-vsf:tag:tr-10-sec:raam-config/v1.0

urn:x-vsf:tag:tr-10-sec:oaim-config/v1.0

urn:x-vsf:tag:tr-10-sec:tct-config/v1.0

The value of those tags is an array of strings where the first entry (index 0) shall be the single decimal digit in string form of the value associated with the configuration option. The second entry (if any, index 1) may be a string of a maximum of 128 characters describing the configuration value.

13 IS-10 Authorizations General Provisions

13.1 IS-10 and BCP-003-02

An implementation supporting the OAuth 2.0 authorization scheme shall comply with AMWA/NMOS IS-10 and AMWA BCP-003-02 except where this Technical Recommendation specifies otherwise. In case of conflict, this Technical Recommendation takes precedence.

Note: This Technical Recommendation overrides some of the IS-10 and BCP-003-02 requirements and provides additional IPMX specific requirements to favor smaller, simpler and more deterministic implementations in IPMX devices.

DRAFT

BY-ND

14 IS-10 Authorizations specific requirements

14.1 Scope

The scope of an OAuth 2.0 authorization is usually the name of the NMOS API used in the path to access the API. For example, accessing the IS-05 ConnectionAPI at *http://api.example.com/x-nmos/connection/{version}* implies the "connection" scope. For IS-04 NodeAPI, QueryAPI, RegistrationAPI, IS-05 ConnectionAPI, IS-08 ChannelMappingAPI, IS-11 StreamCompatibilityManagementAPI and the IS-14 ConfigurationAPI the scope shall be "node", "query", "registration", "connection", "channelmapping", "streamcompatibility" and "configuration" respectively. For IS-12 the scope for accessing the MS-05-02 API shall either be "nc" or "control". For accessing an "x-manufacturer" based API the scope shall be "manufacturer".

14.2 Paths

Access to "/" and "/x-nmos" shall use the "node" API scope. Access to "/x-manufacturer" and "/x-manufacturer/*" shall use the "manufacturer" scope.

Access to an endpoint of the "urn:x-nmos:control:ncp" control type shall use either the "nc" or "control" API scope.

14.3 Behavior

14.3.1 Time Synchronization

The requirements on the clock used by NMOS Nodes are relaxed. An NMOS Node may not be capable of synchronizing its clock to an external source of time. The Node estimation of the true time shall be within 30 minutes of the true NTP / PTP time used by the OAuth 2.0 Authorization Server. The estimated time used by the Node to validate the claims of a token shall comply with this requirement. The estimated time used by the Node to schedule the fetch / update of the OAuth 2.0 Authorization Server public keys shall comply with this requirement.

14.3.2 Public Keys

An NMOS Node shall cache the OAuth 2.0 Authorization Server Public Keys. An NMOS Node shall fetch an initial set of Public Keys after it boots/resets/restarts or after an explicit administrative request, and it shall update them every 23 hours plus X seconds, where X is a random number in the range 0 to 3600. If an NMOS Node cannot obtain an initial set of the Public Keys, it shall refuse access to the NMOS APIs until it retries and obtains an initial set of Public Keys. An NMOS Node shall invalidate the Public Keys from a previous fetch / update operation 36 hours after obtaining them. It shall then refuse access to NMOS APIs until it is able

DRAFT

BY-ND

to obtain a new set of Public Keys. An NMOS Node should use an exponential backoff, from 1 to 64 seconds, when retrying a fetch / update operation.

An NMOS Node shall log an event if it invalidates the Public Keys and it should log an event when it gets a set of Public Keys. An NMOS Node may log an event when it starts refusing access because of a lack of Public Keys. An NMOS Node may log an event when it stops refusing access because of a lack of Public Keys.

An NMOS Node shall discover through DNS-SD the OAuth 2.0 Authorization Servers URL from the standard IS-10 *_nmos-auth._tcp* service or it may be configured with a list of URLs. An NMOS Node shall not use the *iss* claim of a Bearer token to get Public Keys. All the accessible OAuth 2.0 Authorization Servers shall publish the same set of Public Keys such that any OAuth 2.0 Authorization Server may be used by an NMOS Node to obtain the Public Keys and validate access tokens.

Note: The api_selector value is obtained from the DNS-SD TXT records.

An NMOS Node shall use TLS v1.2 or v1.3 when fetching / updating Public Keys from an OAuth 2.0 Authorization Server. It shall validate that the Authorization Server certificate has been signed by a trusted Certificate Authority. An NMOS Node shall be configured with a set of trusted Certificate Authorities for validating access to OAuth 2.0 Authorization Servers.

14.3.2.1 Authorization Server Metadata Endpoint

The location of the Public Keys (the jwks_uri) is not advertised directly. It shall be obtained from the Authorization Server's metadata document, defined in RFC-8414 and referenced from IS-10. The DNS-SD *_nmos-auth._tcp* advertisement carries only the Authorization Server's hostname and port, plus an optional *api_selector* TXT record corresponding to the path component of the issuer identifier (per RFC-8414 §3.1, with leading and trailing / omitted).

IS-10 / RFC-8414 specifies that the metadata URL is constructed by inserting */.well-known/oauth-authorization-server* between the host and the *api_selector*:

<scheme>://<hostname>:<port>/.well-known/oauth-authorization-server[/<api_selector>]

In practice, some widely deployed Authorization Servers instead serve the metadata at the alternative form with the *api_selector* placed before the well-known suffix:

<scheme>://<hostname>:<port>[/<api_selector>]/.well-known/oauth-authorization-server

In addition, every OpenID-Connect-compliant Authorization Server publishes an equivalent metadata document at the OpenID Connect Discovery 1.0 location, which uses the same "well-known appended to the issuer" placement form:

<scheme>://<hostname>:<port>[/<api_selector>]/.well-known/openid-configuration

DRAFT

BY-ND

The OpenID Connect Discovery document carries a *jwks_uri* field with the same semantics as the RFC-8414 metadata document and should be considered an acceptable substitute for it.

To remain interoperable with Authorization Servers, an NMOS Node shall attempt the following URLs in order, stopping at the first one that returns HTTP 200 with a parseable JSON metadata document:

<scheme>://<hostname>:<port>/.well-known/oauth-authorization-server[/<api_selector>]

<scheme>://<hostname>:<port>[/<api_selector>]/.well-known/oauth-authorization-server

<scheme>://<hostname>:<port>[/<api_selector>]/.well-known/openid-configuration

The three forms collapse into two (or one) distinct URLs when no *api_selector* is present, so the additional probes are no-ops for Authorization Servers whose issuer has no path component.

Once a metadata document has been retrieved, the Node shall read the *jwks_uri* field from that document and fetch the Public Keys from that URI. The Node shall not hardcode the JWKS path (e.g. /.well-known/jwks.json or /jwks); the JWKS location is identified normatively only via the metadata document's *jwks_uri*.

14.3.3 Access Token

14.3.3.1 Lifetime

Authorizations are not meant to be provided for short periods of time. An authorization is expected to be delivered for an immediate need for a complete workday. An OAuth 2.0 Bearer token shall have a minimum expiration time (*exp* claim) of 1 hour and a maximum of 24 hours from its creation time (*iat* claim).

If the Node estimated time is greater than the *exp* claim value, accounting for the clock synchronization tolerance defined in this Technical Recommendation, the request shall be rejected with HTTP 401 (Unauthorized).

14.3.3.2 Type and Algorithms

The JOSE header *typ* parameter shall be present and shall have one of the following values: *JWT, at+jwt*, or *application/at+jwt*.

The algorithm *alg* used for signing the Bearer token shall be one of *RS256*, *RS512*, *ES256,* or *ES512*. When *ES256* is used, the elliptic curve shall be *P256*. When *ES512* is used, the elliptic curve shall be *P521*.

Note: This Technical Recommendation extends the interoperability expectations of IS-10.

DRAFT

BY-ND

The Authorization Server may use any of the permitted *alg* values listed above; therefore, an IPMX Node claiming compliance with this Technical Recommendation shall support validation of Bearer tokens using all permitted *alg* values.

14.3.3.3 Grants

NMOS Controllers and similar NMOS sub-systems shall obtain Bearer tokens to access the APIs of NMOS Nodes.

NMOS Controllers and similar NMOS sub-systems should obtain Bearer tokens with *client_credentials* grants to access the APIs of NMOS Nodes.

NMOS Controllers, similar NMOS sub-systems, users and tools may obtain Bearer tokens with *authorization_code* grants to access the APIs of NMOS Nodes.

The *sub* and *client_id* claims of a Bearer token shall be equal for the *client_credentials* grant and shall not be equal for the *authorization_code* and other grants.

An NMOS Node may be configured by an administrator to a) only accept Access Tokens with *client_credentials* grants, b) only accept Access Tokens with *authorization_code* grants, or c) accept both *client_credentials* and *authorization_code* grants. By default, both *client_credentials* and *authorization_code* grants shall be supported.

The claims *iss, aud, sub, exp, scope, client_id* shall be present in the Bearer token.

The *nbf* claim should not be present in the Bearer token. If it is present, it may be ignored.

The *iat* claim may be present in the Bearer token. If it is present, it shall be ignored.

The private claims *x-nmos-** should be placed in an *ext* claim to separate them from standard claims. An NMOS Node shall support having the private claims *x-nmos-** either in the *ext* claim or along with the standard claims. An Access Token should either have the private claims *x-nmos-** in the *ext* claim or along with the standard claims. If the private *x-nmos-** claims are duplicated, they shall be identical.

14.3.3.4 Validation

An NMOS Node shall require TLS v1.2 or v1.3 when serving HTTP requests. An NMOS Node shall only accept Access Tokens from the *Authorization* HTTP header of a request.

An NMOS Node shall validate the cryptographic signature of the Access Token before processing any claims. If signature validation fails, the request shall be rejected with HTTP 401 (Unauthorized) and include a WWW-Authenticate response header as per RFC 6750.

The following requirements define an ordered sequence of validation steps that shall be performed in the specified order. Each step may block access if validation fails.

DRAFT

BY-ND

ReadOnly access to a Node's API shall be blocked if one of the following claims rejects Read accesses. ReadWrite access to a Node's API shall be blocked if one of the following claims rejects Read or Write accesses.

Note: The following requirements are summarized in the pseudo-code section at the end of this Technical Recommendation.

If the OAuth 2.0 Audience Identification Mode (OAIM) configuration is “Serial Number”, the *aud* claim shall not allow access to the current API if it is not ["*"] and no entry corresponds to "*", or contains a DNS name that includes, possibly as a sub-string, the BCP-002-02 Instance Identifier of the NMOS Node. There may be additional characters before and after the Instance Identifier in the DNS name. Authorizations should be delivered to OAuth 2.0 Clients for specific NMOS Nodes based on their serial number, as defined in the BCP-002-02 Instance Identifier. The DNS name of the *aud* clause matching the Instance Identifier of the Node shall additionally be either the Common Name (CN) or one of the alternative DNS names of the TLS server certificate associated with the NMOS endpoint.

If the OAuth 2.0 Audience Identification Mode (OAIM) configuration is “TLS Server Certificate Common Name (CN) or alternative DNS names”, the *aud* claim shall not allow access to the current API if it is not ["*"] and no entry corresponds to "*", the Common Name (CN), or one of the alternative DNS names of the TLS server certificate associated with the NMOS endpoint. The *aud* entries may contain wild-card characters to target a subset of devices on a network. Such wild-carding of domain names is documented in RFC 4592. Implementations shall support RFC 4592 DNS wildcard matching when evaluating audience entries. Authorizations may be delivered to OAuth 2.0 Clients for specific NMOS Nodes based on their DNS name.

If the OAuth 2.0 Audience Identification Mode (OAIM) configuration allows either of the two previous modes, both evaluations are performed and access is allowed if either allows access, and it is denied if both deny access.

If the *aud* claim is an empty array, access shall be denied.

The ordering of the *aud* array is significant for the purpose of interpreting the private *x-nmos-** claims that reference *aud* entries by index (see *read* and *write* attributes processing below). For audience validation, the *aud* claim is processed as usual, with no specific ordering requirement.

An implementation shall maintain the *aud* ordering consistently within the processing of a given access token.

The indexing of the *aud* array is zero-based, so the first entry of the *aud* array has index 0.

The *scope* claim shall not allow access to the current API if the API name is not an element of the space separated list of APIs of the claim. If the API name is present, the *scope* claim shall provide a default Read access for that API. The scope claim shall not grant Write access. An

DRAFT

BY-ND

NMOS Node shall provide such Read access independently of the path being accessed. The presence of an *x-nmos-** claim shall remove the default Read access from the *scope* claim for the associated API, and authorization for that NMOS API shall be determined exclusively by the explicit permissions in the corresponding *x-nmos-** access permissions object.

If the *scope* claim is an empty string, access shall be denied.

Note: Unlike the AMWA NMOS IS-10 Technical Recommendation, the presence of an *x-nmos-** claim matching an NMOS API does not grant implicit Read access. This Technical Recommendation overrides IS-10 such that the presence of an *x-nmos-** claim matching an NMOS API removes the implicit Read access that would otherwise apply due to the *scope* claim for that NMOS API.

This design resolves an internal logical contradiction in the IS-10 Technical Recommendation, which states both that the presence of a private claim grants implicit read access and that permission keys must be omitted if the permission is not granted. By requiring explicit read permission whenever a private claim is present, this Technical Recommendation provides a consistent and deterministic authorization model where the absence of a permission key reliably indicates that the permission has not been granted.

The *read* attribute of an *x-nmos-** claim, if present, shall provide Read access if the array of paths is ["*"] and shall deny Read access if the array of paths is [""]. The absence of a *read* attribute prevents Read access. An NMOS Node shall provide such Read access independently of the path being accessed. Values other than ["*"], [""], or an array of signed integers shall not be used. Implementations shall support all three forms of the *read* attribute: ["*"] for allow, [""] for deny, and arrays of signed integers for indexed allow/deny. The array of signed integers shall not be empty and shall be sorted to have positive integers first followed by negative integers. The integer value 0 shall be considered as a positive integer. If the absolute value of any array entry is outside the bounds of the *aud* claim array, the access token is invalid and Read access shall be denied.

The array of signed integers is split into two sub-arrays, one for non-negative integers and one for negative integers, keeping the same ordering as in the original array. The non-negative integer array is an allow-list while the negative integer array is a deny-list.

If the allow-list is not empty, Read access shall be denied unless, for at least one allow-list entry *i*, the associated *aud* claim array entry *aud[i]* considered alone allows access. If the allow-list processing allows Read access, such access shall be denied if, for any deny-list entry *i*, the associated *aud* claim array entry *aud[abs(i)]* considered alone allows access, otherwise Read access shall be allowed.

If the allow-list is empty, the deny-list is a deny-only list: Read access shall be denied if, for any deny-list entry *i*, the associated *aud* claim array entry *aud[abs(i)]* considered alone allows access, otherwise Read access shall be allowed.

DRAFT

CC BY-ND

The *write* attribute of an *x-nmos-** claim, if present, shall provide Write access if the array of paths is ["*"] and shall deny write access if the array of paths is [""]. The absence of a write attribute prevents Write access. Both Read and Write access shall be allowed to get Write access. An NMOS Node shall provide such Write access independently of the path being accessed. Values other than ["*"], [""], or an array of signed integers shall not be used. Implementations shall support all three forms of the *write* attribute: ["*"] for allow, [""] for deny, and arrays of signed integers for indexed allow/deny. The array of signed integers shall not be empty and shall be sorted to have positive integers first followed by negative integers. The integer value 0 shall be considered as a positive integer. If the absolute value of any array entry is outside the bounds of the *aud* claim array, the access token is invalid and Write access shall be denied.

The array of signed integers is split into two sub-arrays, one for non-negative integers and one for negative integers, keeping the same ordering as in the original array. The non-negative integer array is an allow-list while the negative integer array is a deny-list.

If the allow-list is not empty, Write access shall be denied unless, for at least one allow-list entry *i*, the associated *aud* claim array entry *aud[i]* considered alone allows access. If the allow-list processing allows Write access, such access shall be denied if, for any deny-list entry *i*, the associated *aud* claim array entry *aud[abs(i)]* considered alone allows access, otherwise Write access shall be allowed.

If the allow-list is empty, the deny-list is a deny-only list. Write access shall be denied if, for any deny-list entry *i*, the associated *aud* claim array entry *aud[abs(i)]* considered alone allows access, otherwise Write access shall be allowed.

If the current API access is having side-effects on the state of the NMOS Node, Read and Write access shall be allowed by the access token. Otherwise, the API request shall fail with HTTP 403 (*Forbidden*) if the token is valid, but permissions are insufficient, or HTTP 401 (*Unauthorized*) with a *WWW-Authenticate* response header if the token is invalid or missing.

If the current API access is not having side-effects on the state of the NMOS Node, Read access shall be allowed by the access token. Otherwise, the API request shall fail with HTTP 403 (*Forbidden*) if the token is valid, but permissions are insufficient, or HTTP 401 (*Unauthorized*) with a *WWW-Authenticate* response header if the token is invalid or missing.

An NMOS Node should increment a status counter

a) when a ReadOnly access is denied:
    a.1) based on the *sub* claim,
    a.2) based on the *aud* claim,
    a.3) based on the *scope* claim,
    a.4) based on the *x-nmos-** claim,
b) when a ReadWrite access is denied:
    b.1) based on the *sub* claim,
    b.2) based on the *aud* claim,

DRAFT

BY-ND

b.3) based on the scope claim,
b.4) based on the *x-nmos-** claim,
c) when an access without an Access Token is performed,
d) when an access with an invalid or corrupted token is performed,
e) when an access with an expired or not-yet-valid token is performed,
f) when a TLS client certificate validation fails,
g) when a TLS server certificate validation fails during a client access,
h) when a fetch/update of the OAuth 2.0 Authorization Server public keys fails,
i) when an access is denied because no valid Public Keys are available.

The status counters shall be 64-bit unsigned integers and shall be monotonic (non-decreasing) since boot/reset/restart. An IPMX device may persist the counters across boot/reset/restart and if so, shall provide an administrator a means to reset them through an explicit administrative action.

Note: These status counters represent the minimum required audit capability. This requirement does not prevent implementations from providing more elaborate logging (e.g., source IP, client_id, or timestamps) for security events if device resources permit.

14.3.3.5 Security Failure Handling

An NMOS Node shall adopt a "fail-closed" security posture. Access shall be denied if any required security validation step cannot be successfully completed. This includes, among other things, the following cases:

If a Certificate Revocation List (CRL) is required but cannot be retrieved, has an invalid signature, or is expired, the Node shall treat all certificates that would have been validated against that CRL as invalid and shall deny access.

If the OAuth 2.0 Authorization Server is unreachable and the Node has no valid cached Public Keys, the Node shall deny access to all requests requiring OAuth 2.0 authorization.

If a certificate validation fails for any reason (including expiry, untrusted CA, or revocation), the connection shall be terminated if an HTTP response is not possible. If an HTTP response is possible, access shall be denied and the request shall return the status HTTP 401 (Unauthorized).

14.3.3.6 Mutual TLS Client Certificate Binding

When a Node API endpoint is accessed using mutual TLS (mTLS) and an OAuth 2.0 access token is provided in the *Authorization* HTTP header of a request, the Node shall enforce an additional *client_id* entity verification as specified below:

- The Node shall authorize the request only if the *client_id* claim value matches the Common Name (CN) or one of the alternative DNS names of the TLS client certificate accessing the NMOS endpoint.

DRAFT

BY-ND

- The comparison shall be case-insensitive.

- Wildcards shall not be considered a match. If a wildcard is present in any SAN entry the Node shall treat it as non-matching for the purpose of this check. If no match is found, the Node shall reject the request.

This check binds the access token to the identity of the OAuth 2.0 client (the Controller or tool), as identified by the *client_id* claim. The *sub* claim identifies the end user (where applicable) and shall not be used for this binding.

When mTLS is used, OAuth 2.0 clients (Controllers or tools) shall be provisioned such that their *client_id* value matches (case-insensitively) the Common Name (CN) or one of the alternative DNS names of their TLS client certificate. Wildcards shall not be used for this purpose. An OAuth 2.0 Authorization Server should enforce this requirement at client registration time and/or when issuing access tokens.

14.3.3.7 WebSocket

An NMOS Node should provide endpoints for getting a WebSocket upgrade that are specific for ReadOnly access and ReadWrite access. The ReadOnly endpoint should have the "Guest" suffix. If a ReadOnly access endpoint is not provided, the endpoint is qualified as having ReadWrite access and causing side-effects on the state of the NMOS Node. So, although a *GET* verb is used to get an upgrade to a WebSocket, the request cannot be qualified as ReadOnly unless explicitly qualified.

Subscribing to notification messages shall be considered a read-only operation. Registering a websocket for receiving notification messages from objects may cause side-effects on the state of the websocket connection. This shall not be considered as causing side-effects on the state of the NMOS Node. The Read-Only versus Read-Write qualifiers relate to objects in the MS-05-02 framework, not to the IS-12 mechanisms for accessing, controlling and monitoring those objects.

14.3.4 HTTP Status Codes

Implementations shall use appropriate HTTP status codes for access failures:

- **401 Unauthorized**: Returned when the access token is missing, expired, malformed, has an invalid signature, or fails the Mutual TLS Client Certificate Binding check. The response includes a *WWW-Authenticate* response header as per RFC 6750.

- **403 Forbidden**: Returned when the token is valid, but the requested operation is denied due to insufficient permissions (e.g., scope mismatch, audience restrictions, or explicit deny via *x-nmos-** claims).

DRAFT

BY-ND

14.3.5 Examples (Informative)

The following examples illustrate access tokens for a Controller operating across multiple NMOS Nodes.

Example 1: complete token using ["*"] for multiple APIs and multiple Nodes

This example follows a simple pattern where the Authorization Server grants full read/write access for the listed APIs on all targeted Nodes. This example uses authorization_code grants.

```
{
  "iss": "https://oauth2.matrox.com/v1.0",
  "scope": "offline node connection streamcompatibility",
  "sub": "user@matrox.com",
  "aud": ["MTXCIP-CC91629", "MTXCIP-CC91699"],
  "client_id": "nmosController-54321",
  "exp": 1.720538859e+09,
  "x-nmos-node": { "read": ["*"], "write": [""] },
  "x-nmos-connection": { "read": ["*"], "write": ["*"] },
  "x-nmos-streamcompatibility": { "read": ["*"], "write": ["*"] }
}
```

Example 2: compact token using aud indices

This example illustrates how the same token can remain compact by referencing *aud* entries by index. In this example, the Controller is authorized for write accesses only on node MTXCIP-CC91699. The *x-nmos-node* claim is not present because the *scope* claim already provides read access.

```
{
  "iss": "https://oauth2.matrox.com/v1.0",
  "scope": "offline node connection streamcompatibility",
  "sub": "user@matrox.com",
  "aud": ["MTXCIP-CC91629", "MTXCIP-CC91699"],
  "client_id": "nmosController-54321",
  "exp": 1.720538859e+09,
  "x-nmos-connection": { "read": [0, 1], "write": [1] },
  "x-nmos-streamcompatibility": { "read": ["*"], "write": [1] }
}
```

Example 3: compact token using aud indices with universal read-only

This example illustrates how the same token can remain compact by referencing *aud* entries by index. In this example, the Controller is authorized for write accesses only on nodes MTXCIP-

BY-ND

CC91629 and MTXCIP-CC91699 while ReadOnly access is authorized for any node. The *x-nmos-node* claim is not present because the *scope* claim already provides read access.

```
{
  "iss": "https://oauth2.matrox.com/v1.0",
  "scope": "offline node connection streamcompatibility",
  "sub": "user@matrox.com",
  "aud": ["*", "MTXCIP-CC91629", "MTXCIP-CC91699"],
  "client_id": "nmosController-54321",
  "exp": 1.720538859e+09,
  "x-nmos-connection": { "read": ["*"], "write": [1, 2] },
  "x-nmos-streamcompatibility": { "read": ["*"], "write": [1, 2] }
}
```

Example 4: complete token using ["*"] for multiple APIs and multiple Nodes

This example follows a simple pattern where the Authorization Server grants full read/write access for the listed APIs on all targeted Nodes. This example uses client_credentials grants.

```
{
  "iss": "https://oauth2.matrox.com/v1.0",
  "scope": "node connection streamcompatibility",
  "sub": "nmosController-54321",
  "aud": ["*"],
  "client_id": "nmosController-54321",
  "exp": 1.720538859e+09,
  "x-nmos-node": { "read": ["*"], "write": [""] },
  "x-nmos-connection": { "read": ["*"], "write": ["*"] },
  "x-nmos-streamcompatibility": { "read": ["*"], "write": ["*"] }
}
```

14.3.5.1 Pseudocode (Informative)

The pseudocode below illustrates the normative requirements of the Validation section for evaluating read/write access for a single HTTP request.

```
#
# Inputs:
# - claims: decoded JWT access token claims (signature already validated)
# - read_write: True if the request has side-effects on the state of the NMOS Node
# - api_name: current NMOS API name (e.g. "node", "connection", "query", ...)
# - node_instance_id: BCP-002-02 Instance Identifier of the NMOS Node (if used in aud)
# - tls_server_cert_names: set of DNS names from TLS server certificate (CN + SANs)
# - use_client_credentials_grant_only: bool (policy switch)
# - use_serial_number_in_aud: bool
#     - True: aud entries contain the Instance Identifier (serial number) and MUST also match TLS server
certificate identity
#     - False: aud entries are (wildcard-capable) TLS server certificate names
#
# Output:
# - allow: boolean
```


BY-ND

```
# - otherwise the request fails with HTTP 401 and includes a WWW-Authenticate response header

def validate_access(
    claims,
    read_write,
    api_name,
    node_instance_id,
    tls_server_cert_names,
    use_client_credentials_grant_only,
    use_serial_number_in_aud,
):
    # ---- required claims ----
    required = ["iss", "sub", "aud", "client_id", "exp", "scope"]
    for k in required:
        if k not in claims:
            return DENY  # invalid token


    # ---- exp check ----
    exp = claims["exp"]
    if not is_number(exp):
        return DENY  # invalid token
    if now() > unix_time(exp):
        return DENY


    # ---- grant policy (optional) ----
    sub = claims["sub"]
    client_id = claims["client_id"]
    if not is_string(sub) or not is_string(client_id):
        return DENY  # invalid token


    if use_client_credentials_grant_only:
        if sub != client_id:
            return DENY  # invalid token


    # ---- scope gating ----
    scope = claims["scope"]
    if not is_string(scope):
        return DENY  # invalid token
    if api_name not in split_by_spaces(scope):
        return DENY


    # ---- aud gating ----
    aud = claims["aud"]
    if not is_array(aud) or any(not is_string(a) for a in aud):
        return DENY  # invalid token


    if not any(aud_entry_allows_current_node(a, node_instance_id, tls_server_cert_names,
use_serial_number_in_aud) for a in aud):
        return DENY


    # ---- locate private claim x-nmos-<api> (it may be in ext or at top-level) ----
    priv = None
    access_key = "x-nmos-" + api_name


    if "ext" in claims:
        if not is_object(claims["ext"]):
```


BY-ND

```
            return DENY  # invalid token
        if access_key in claims["ext"]:
            priv = claims["ext"][access_key]
    else:
        if access_key in claims:
            priv = claims[access_key]

    # ---- no private claim: RO allowed, RW denied ----
    if priv is None:
        return ALLOW if not read_write else DENY

    if not is_object(priv):
        return DENY  # invalid token

    # Presence of x-nmos-<api> removes default implicit read access from scope.
    # Read must be granted explicitly by the private claim.
    read_allowed = eval_indexed_attr(priv.get("read", MISSING), aud, node_instance_id,
tls_server_cert_names, use_serial_number_in_aud)
    if read_allowed is INVALID:
        return DENY  # invalid token
    if read_allowed is not ALLOW:
        return DENY

    write_allowed = eval_indexed_attr(priv.get("write", MISSING), aud, node_instance_id,
tls_server_cert_names, use_serial_number_in_aud)
    if write_allowed is INVALID:
        return DENY  # invalid token

    # Consistency rule: it is invalid for a token to grant write access without also granting read access.
    if (write_allowed is ALLOW) and (read_allowed is not ALLOW):
        return DENY  # invalid token

    if read_write:
        return ALLOW if ((read_allowed is ALLOW) and (write_allowed is ALLOW)) else DENY
    else:
        return ALLOW if (read_allowed is ALLOW) else DENY



def aud_entry_allows_current_node(aud_entry, node_instance_id, tls_server_cert_names,
use_serial_number_in_aud):
    if aud_entry == "*":
        return True

    if use_serial_number_in_aud:
        # aud_entry must contain node_instance_id as a substring
        if node_instance_id not in aud_entry:
            return False
        # and must also match TLS server certificate identity (CN or SAN)
        return aud_entry in tls_server_cert_names

    # use_serial_number_in_aud == False: aud entries are TLS cert names (with RFC 4592 wildcards)
    return matches_dns_wildcard(aud_entry, tls_server_cert_names)

def matches_dns_wildcard(pattern, cert_names):
    # Implement RFC 4592 DNS wildcard matching (e.g., *.example.com matches sub.example.com)
```


BY-ND

```
    for name in cert_names:
        if dns_wildcard_matches(pattern, name):
            return True
    return False

def dns_wildcard_matches(pattern, target):
    # Simplified: pattern like *.domain matches sub.domain
    # Full RFC 4592 implementation needed in real code
    if pattern.startswith("*."):
        domain = pattern[2:]
        return target.endswith("." + domain) and "." not in target[:-len("." + domain)]
    return pattern == target



def eval_indexed_attr(attr, aud, node_instance_id, tls_server_cert_names, use_serial_number_in_aud):
    # Returns one of {ALLOW, DENY, INVALID}
    #
    # Supported forms:
    # - ["*"]  => ALLOW
    # - [""]   => DENY
    # - [signed integers...] referencing aud indices:
    #     i >= 0: allow if aud[i] alone allows
    #     i <  0: deny  if aud[abs(i)] alone allows
    # - missing => DENY
    if attr is MISSING:
        return DENY

    if is_array(attr) and len(attr) == 1 and is_string(attr[0]):
        if attr[0] == "*":
            return ALLOW
        if attr[0] == "":
            return DENY
        return INVALID

    if not is_array_of_signed_integers(attr):
        return INVALID

    if len(attr) == 0:
        return INVALID

    # The array MUST be sorted to have positive integers first then negative integers (0 is positive).
    # Do not auto-sort; treat ordering violations as INVALID.
    list_pos = []
    list_neg = []
    seen_negative = False
    for i in attr:
        if i < 0:
            seen_negative = True
            list_neg.append(i)
        else:
            if seen_negative:
                return INVALID
            list_pos.append(i)

    # Evaluate allowlist/denylist semantics.
    #
```


CC BY-ND

```
    # - If there is at least one non-negative index, the list is an allowlist with optional deny
exceptions:
    #     * DENY unless at least one non-negative index matches.
    #     * If allowed, DENY if any negative index matches (deny exception).
    # - If the list contains only negative indices, it is a denylist:
    #     * ALLOW unless a negative index matches.

    def idx_matches(i):
        if abs(i) >= len(aud):
            return INVALID
        aud_entry = aud[abs(i)]
        return aud_entry_allows_current_node(aud_entry, node_instance_id, tls_server_cert_names,
use_serial_number_in_aud)

    # Phase 1: allow-list (if present)
    if len(list_pos) > 0:
        allowed = False
        for i in list_pos:
            m = idx_matches(i)
            if m is INVALID:
                return INVALID
            if m:
                allowed = True
                break
        if not allowed:
            return DENY

    # Phase 2: deny-list exceptions (applies both to allowlist and denylist-only)
    for i in list_neg:
        m = idx_matches(i)
        if m is INVALID:
            return INVALID
        if m:
            return DENY

    return ALLOW
```


CC BY-ND