BSR/ASHRAE Addendum *cg* to
ANSI/ASHRAE Standard 135-2020

Public Review Draft

Proposed Addendum *cg* to Standard 135-2020, BACnet® - A Data Communication Protocol for Building Automation and Control Networks

First Public Review (June 2022)
(Draft shows Proposed Changes to Current Standard)

This draft has been recommended for public review by the responsible project committee. To submit a comment on this proposed standard, go to the ASHRAE website at www.ashrae.org/standards-research--technology/public-review-drafts and access the online comment database. The draft is subject to modification until it is approved for publication by the Board of Directors and ANSI. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE website) remains in effect. The current edition of any standard may be purchased from the ASHRAE Online Store at www.ashrae.org/bookstore or by calling 404-636-8400 or 1-800-727-4723 (for orders in the U.S. or Canada).

This standard is under continuous maintenance. To propose a change to the current standard, use the change submittal form available on the ASHRAE website, www.ashrae.org.

The appearance of any technical data or editorial material in this public review document does not constitute endorsement, warranty, or guaranty by ASHRAE of any product, service, process, procedure, or design, and ASHRAE expressly disclaims such.

ASHRAE, 180 Technology Parkway NW, Peachtree Corners, GA 30092

[This foreword, the table of contents, the introduction, and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2020 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this document is provided for context only and is not open for public review comment except as it relates to the proposed changes.

The use of placeholders like XX, YY, ZZ, X1, X2, NN, x, n, ? etc. should not be interpreted as literal values of the final published version. These placeholders will be assigned actual numbers/letters only after final publication approval of the addendum.

135-2020*cg*-1 BACnet Architecture for Device Authentication and Authorization

Rationale

BACnet is fundamentally a "peer to peer" protocol. A message can be sent from any device to any device. This is very good for interoperability but can be undesirable from a security standpoint. Therefore, today, most BACnet/IP networks are protected by VPN/VLAN and most other network types are protected by "physical security." This generally protects against "outsiders" exerting unwelcomed control over BACnet devices but does not prevent damage from an "inside" BACnet device that is either malicious or misconfigured.

Various proprietary solutions have been created to limit which devices are allowed to perform certain operations on other devices, but, by their nature, they are not interoperable.

This proposed set of changes defines a standard and interoperable mechanism for authenticating and authorizing devices to take actions in other devices. It is based on the Internet standards OAuth 2.0, OpenID Connect, ACE-OAuth, and the suite of RFCs that define the data structures that support these protocols.

Note that BACnet Web Services (Annex W) also uses OAuth 2.0 and the same standard data structures. However, BACnet/WS is encoded in JSON and uses HTTP, whereas this solution is encoded in ASN and applies to the entire suite of binary-encoded services like WriteProperty, AtomicWriteFile, etc.

Also note that this is "device" authentication and authorization, not "user" authentication and authorization. The credentials to identify human users and the mechanism for such validation is not in scope here.

This addition to the standard is split into to two parts, one "normative" and one "informative". This allows the recommendations and deployment options to be discussed in the informative part while keeping the normative part more succinct.

[Change **Clause 3.2**, p. 2]

3.2 Terms Defined for this Standard

...
authentication: the act of verifying identity. *See also "device authentication".*

authorization (network security): the control of access to network resources based on known identity and access rules. *See also "device authorization".*
...

[Insert new entries to **Clause 3.2**, preserving the alphabetical order, p. 2]

identity: An identifier by which a device is uniquely identified in a BACnet internetwork. In this specification, device identity is determined by the device instance number.

protected operation: A BACnet service action (e.g., a WriteProperty to a particular object's property) that requires authorization to complete successfully.

device authentication: A mechanism for determining that the sender of a message is the device that it claims to be.

device authorization: Permission that is granted to a device to perform protected operations in another device.

authenticated device: A device that has proven its identity by some trusted means beyond its own claims.

authorized device: A device that has presented trusted permissions to perform a protected operation.

client (authorization context): A BACnet device that is attempting a protected operation and therefore needs authorization.

resource server: A BACnet device that contains a resource (e.g., a property) that may be protected from modification by unauthorized clients.

authorization server: A BACnet device that grants permissions to clients to perform a protected operation.

identity server: A BACnet device that attests to the identity of a client device.

identity token: A cryptographically signed data structure that binds a device instance number (the "identity") indirectly to private keying material held by the client. e.g., via the "subject" name of a X.509 certificate.

access token: A cryptographically signed data structure that grants authorization permissions ("scope") to a client.

token binding: The process of binding an access token or an identity token to a particular device so that it cannot successfully be used by another device.

scope (authorization context): the set of permissions granted to a device to perform a protected operation.

OAuth: The Open Authentication Framework for Internet services, RFC 6750 and related RFCs. In this standard, the term "OAuth" is shorthand for "OAuth 2.0".

ACE-OAuth: A constrained profile of OAuth that provides token caching and authorization "hints".

[Change **Clause 4.3**, p. 24]

4.3 Security

The principal security threats to BACnet systems are people who, intentionally or by accident, modify a device's configuration or control parameters. Problems due to a malfunctioning or misconfigured computer are outside the realm of security considerations*, with the exception that the network actions of such a device may be restricted by the authentication and authorization mechanisms defined in Annex XX*. One important place for security measures is the operator-machine interface. Since the operator-machine interface is not part of the communication protocol, vendors are free to include password protection, audit trails, or other controls to this interface as needed. In addition, write access to any properties that are not explicitly required to be "writable" by this standard may be restricted to modifications made ~~only in virtual terminal mode~~ *via Annex XX methods* or be prohibited entirely. ~~This permits vendors to protect key properties with a security mechanism that is as sophisticated as they consider appropriate.~~ *For added security, even properties that are required to be "writable" can be restricted to allowing modifications made only via the methods provided by Annex XX. However, enabling such restrictions shall be user configurable.*

[Add new **Annex XX**, p. 1403]

ANNEX XX - BACnet Device Authentication and Authorization (NORMATIVE)

(This annex is part of this standard and is required for its use.)

BACnet is fundamentally a "peer to peer" protocol. A message can be sent from any device to any device. As the default behavior, this is very good for interoperability. However, in many cases, it is desirable that access to some resources be restricted to only a subset of peers that have been given permission for such actions.

This annex defines a standard and interoperable mechanism for authenticating and authorizing devices to take actions in other devices. It is a BACnet adaptation of the established Internet standards OAuth 2.0, ACE-OAuth, OpenID Connect, and the IANA registries and RFCs that define the structures and semantics of the data that support those protocols.

While the mechanisms in this annex are based on OAuth 2.0 (hereafter simply "OAuth"), they are restricted to the "client credentials" flow and are used only for authenticating and authorizing devices. Authentication of human operators is not in scope. However, to support the auditing and logging of operations taken by the authorized devices, user and role identifiers can be included in the messages. For the mechanisms described in this annex, if the client device is trusted for a given operation, then the client's claim of the user and role identifiers is also trusted. The mechanism by which the authorized device determines the identity of a human user or role is a local matter. For example, some clients might use a simple PIN-based authentication while others might use sophisticated authentication methods such as LDAP, smart cards, etc.

Note that BACnet Web Services (Annex W) also uses OAuth. However, BACnet/WS is encoded in JSON, uses HTTP, and does not make restrictions on other OAuth flows, whereas the mechanisms in this annex are encoded in ASN.1, apply to the NPDU-based services defined in Clauses 13,14, 15, 16, and 17, and only authenticate and authorize devices.

Information for authentication and authorization is an optional addition to any NPDU and accompanies the NPDU in the form of "data attributes". See Clauses 5.1, 6.1, and AB.1.4.

XX.1 Overview

This clause provides a high-level summary of the concepts and dataflows used for authentication and authorization. The detailed requirements are provided by subsequent Clauses XX.2 and XX.3, and by secure datalinks, such as BACnet/SC in Annex AB. Implementation and interoperability guidelines are provided by informative Annex YY, and examples of data flow and data values are provided by Annex ZZ.

XX.1.1 Authentication

Authentication is the process of proving that you are who you say you are. In BACnet, the "identity" of a device is its device instance number. This identity is proven by a signed "identity token" that ties a unique device "subject name" to a unique device instance number.

Since all BACnet devices need to be able to determine the identity of any other BACnet device, there is a single "identity server" that issues and signs the identity tokens. All devices are configured with the keys needed to validate the signature of these tokens.

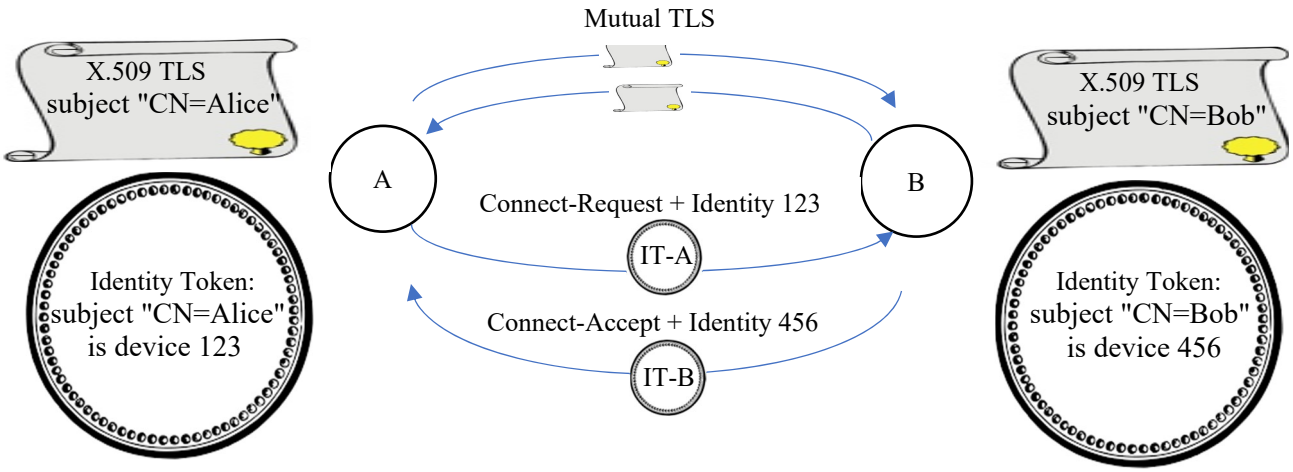
The identity server is the single most important part of BACnet security. The keys that are used to sign identity tokens shall be carefully protected from disclosure.

Authentication is enforced at the peer-to-peer datalink level. Therefore, each secure datalink defines the means to validate that the device has the right to use the "subject name" and therefore the right to use the identity token.

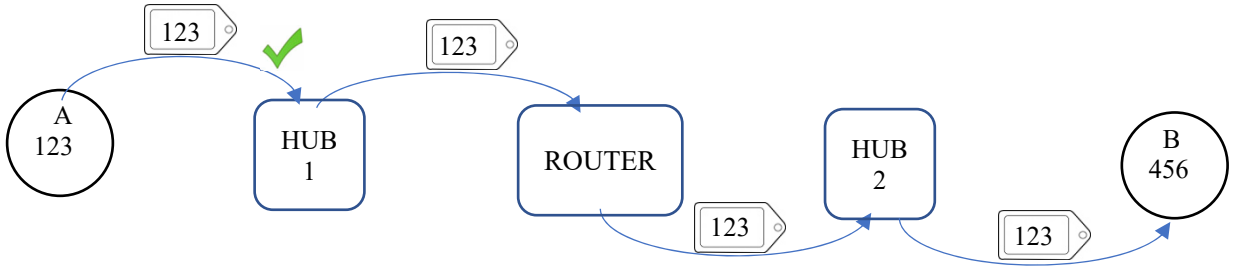
The following uses the TLS connections of BACnet/SC as an example.

After connecting with mutually authenticated TLS, the BACnet/SC initiating peer sends an identity token along with its Connect-Request message, and the accepting peer returns an identity token with the Connect-Accept message.

Both sides check that the subject in the identity token matches the 'subject' distinguished name in the TLS certificate. If they match, then each side will securely know the identity of the other side.



Identity is relayed by secure hubs and routers, e.g., a BACnet/SC to BACnet/SC router, so that the recipient can securely know who sent it, even though the recipient does not have direct access to the originator's TLS certificate or its identity token. To ensure that an identity claim is not forged, the first hub or router will verify that the claimed device instance matches the originator's identity token, and subsequent recipients will only accept the device instance information if it came from an authenticated hub or router.

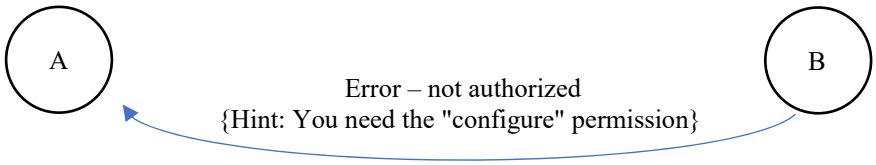


XX.1.2 Authorization

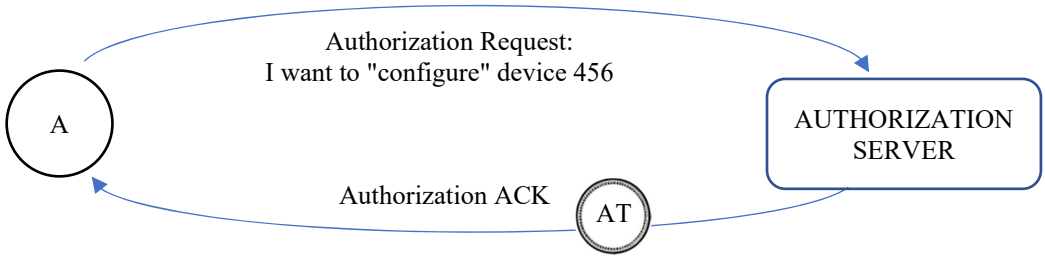
This authentication mechanism described above allows the recipient to securely know who sent the message, but it doesn't convey a sense of "permission" for what that sender can do. To avoid distributing access policy into every end device, a centralized "authorization server" is configured to know what actions a particular device is allowed to do to some other device.

These "permissions" are conveyed with "access tokens". Like the identity tokens, access tokens are signed and bound to the sender, so that the recipient knows that they have not been maliciously forged or stolen.

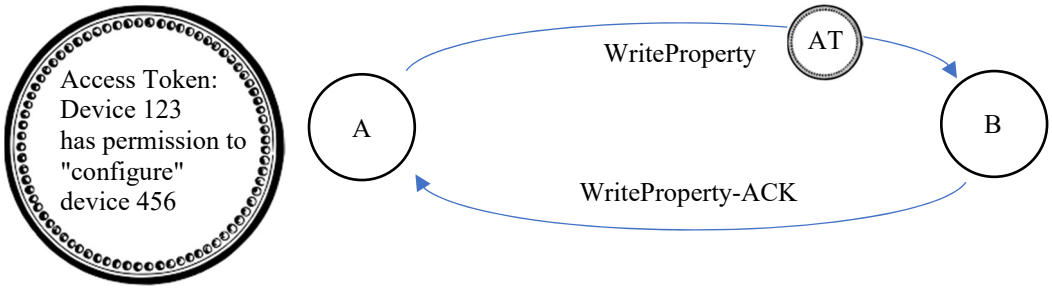
If an attempt to perform a restricted operation is tried without proper authorization, an error message is returned as usual. But, in addition the error code, the response can also contain a "hint" as to what permission is needed.



So now the client can ask for that permission from the authorization server for the target device. If the request is allowed, the authorization server will send back a signed access token that specifies the details of the permission.



The client now has an access token that it can send along with the message and the operation will succeed. The recipient trusts the access token because it is signed by its authorization server and it is bound to both the sender and the recipient, so it could not have been stolen or redirected.



XX.2 Device Authentication

The identity of a BACnet device is indicated by its device instance number, which is required to be unique internetwork-wide. See Clause 12.11.1. It is possible however for a device identifier to be unconfigured or misconfigured. Therefore, for a device's claim of its instance number to be trusted, an attestation from a trusted third party is required. This attestation is provided by an identity token provided by an identity server.

Identity tokens are public declarations. They can be conveyed by any means without security. However, they can only be successfully used and verified in a secure context, like TLS used by BACnet/SC. See Clause XX.2.7 for details of the identity tokens. The identity token is configured into the Identity_Token property of a device's Auth object either by the device itself using the AuthRequest Service or by some local means beyond the scope of this standard.

An identity token ties the device instance number to a 'subject' name, which is also required to be unique internetwork-wide. The device must be able to prove that it has the right to use the subject name by providing a cryptographic connection between the subject name and private keying material that it possesses. For example, in BACnet/SC, this connection is done by the X.509 certificate used for the TLS connection.

There is only one identity authority in a BACnet internetwork, so all identity tokens are signed by one of the two keys maintained by the identity server. This allows any BACnet device to validate the identity of any other BACnet device. Note that this contrasts with the authorization mechanism defined in Clause XX.3~~XX.2~~ which can support multiple authorization authorities within one internetwork.

To enable propagating authentication information by a device with a different identity (e.g., a hub or router), an extra permission ("hub", "router", or both) is added to the scope of the identity token.

XX.2.1 Authenticated Connections

For connection-oriented datalinks, like BACnet/SC, when a device initiates a connection to a peer device, it proves its identity by providing its identity token to the peer device as part of the connection establishment. Likewise, the accepting peer provides its own identity token in return. If the identity token is properly validated, the connection is marked as having an "authenticated" peer. If no identity token is provided, or the token is invalid, the peer will remain "unauthenticated". In this case, the connection can continue, but a device instance number claim by the peer shall not be trusted for authorization decisions.

For example, in BACnet/SC, after connecting with mutually authenticated TLS, the initiating peer sends an identity token along with its Connect-Request message, and the accepting peer returns an identity token with the Connect-Accept message. Both sides check that the subject in the identity token matches the subject in the TLS certificate. If they match, then it will securely know the device instance of the other side.

XX.2.2 Conveying Authentication

The secure identity of the sender of a message is conveyed in a "Secure Source" data attribute that contains the device instance of the sender of the message. If the identity of the sender has not been validated but is desired to be included for informational purposes, then it can be sent in a "Nonsecure Source" attribute. In BACnet/SC, these attribute values are conveyed in "Secure Source" and "Nonsecure Source" data options. See Clauses AB.2.3.X1 and AB.2.3.X2.

The "Nonsecure Source" attribute is not to be trusted for authorization decisions but may be useful for logging or debugging purposes. If routers know the identity of a device by some means, they can apply the "Nonsecure Source" attribute to a message as it passes through. The "Secure Source" shall only be applied if the source information has been validated securely or is inherently tamper-proof, such as when originating from a virtual network. See Annex H.1.1.

Note that the "Secure Source" data attribute is defined to be an "every segment" attribute. For segmented messages, if "Secure Source" is used, it shall be present and have the same value on every segment. See Clauses 5.2.1.1 and 5.4.3. If its presence or value differs on any segment, the message shall be aborted. See Clauses 5.4.4 and 5.4.5.

XX.2.3 Claiming Authentication

Not all messages require authentication, so the choice to include the "Secure Source" data attribute is made by the message originator. If an originating device wishes to authenticate a particular message, it shall apply the "Secure Source" attribute to the message. If the originating device does not wish to make such a distinction of when authentication is needed, it can simply include "Secure Source" on every message. Since this claim could be applied maliciously, the acceptance or rejection of a "Secure Source" claim will be determined by the receiver. See Clause XX.1.2.

Note that the authorization mechanism defined by Clause XX.3~~XX.2~~ requires that authentication accompanies the authorization.

XX.2.4 Accepting Authentication

The 'Secure Source' attribute shall be applied by the device originating the message, but it shall only be trusted by the receiver under the rule in this clause.

In the following description, the "sending device" means the device that immediately sends a BACnet packet to the "receiving device". Both can be different from the "originating device" and the "destination device" when hubs or routers are involved. Thus, the following rules apply to each step of the message's journey through hubs and routers.

When a "Secure Source" attribute is received at the datalink layer,

a) If the sending device is not authenticated to the receiving device, then the 'Secure Source' attribute shall either be dropped or turned into a 'Nonsecure Source' before being passed to the network layer.
b) If the sending device is authenticated to the receiving device and the 'Secure Source' value matches the sending device's authenticated value, then the 'Secure Source' attribute shall be accepted and passed to the network layer.
c) If the sending device is authenticated to the receiving device and the 'Secure Source' value does not match the sending device's authenticated value, then, if the scope of the sending device's identity contains "hub" or "router", the 'Secure Source' attribute shall be accepted and passed to the network layer, otherwise, the message shall be rejected or dropped and the connection to the sender shall be disconnected if possible. Such a mismatch is either erroneous or malicious and should be logged and/or reported since only hubs and routers are allowed to propagate a 'Secure Source' that is different from themselves. See Clause XX.2.5~~XX.1.5~~.

Since a 'Secure Source' attribute can be forged by bad actors and forwarded by older, unaware intermediaries, the security of the 'Secure Source' option is always enforced by newer devices at the receiving end of a connection. In summary, the above rules mean that:

a) An older destination will not understand the new attribute and will simply throw it away and behave as it always has.
b) Newer destinations will only accept the 'Secure Source' option if it came from an authenticated peer.
c) If the peer is authenticated, it can be trusted that -it- rejected anything that it received from an unauthenticated peer upstream, and so on, back to the original sender.
d) Therefore, if a 'Secure Source' option comes in from an authenticated peer, the destination knows that the entire chain from originator to destination is authenticated and that the 'Secure Source' can be trusted.

XX.2.5 Propagating Authentication

Identity is relayed by hubs or routers so that the destination can securely know who sent it, even though the destination does not have direct access to the originator's identity token. To ensure that an identity claim is not forged, the rules defined by Clause XX.2.4~~XX.1.4~~ are applied at each hop by the receiving datalink layer.

The first hub or router will verify that the claimed device instance matches the originator's identity token. Subsequent hops will verify that the message was sent from an authenticated hub or router. If any peer along the way is unauthenticated to the next peer in the chain this will result in the "Secure Source" attribute being dropped or changed into a "Nonsecure Source" attribute. This applies all the way to the final destination. For example, if the "Secure "Source" makes it all the way to the last hop, but the final sender is not authenticated to final receiver, the attribute would be dropped or changed by the final receiving datalink before being sent up the stack on that device. The use of "Nonsecure Source" allows logging or debugging by the destination without being used for authorization decisions. See the examples in Annex ZZ.

XX.2.6 Response Authentication

If a confirmed request message is received that has a "Secure Source" attribute, then any response, whether success or error, shall also have the "Secure Source" attribute applied, if possible, by the responding device. This allows the requesting device to validate that the response is from the intended target device.

It is sometimes necessary for a client device to establish the identity of a server device before sending sensitive information that should not be accidentally misdirected. In many cases, the client device will perform a device address binding query and determine the address of the server device with an authenticated I-Am. However, if the client device knows only the NET:MAC of the server device, it can determine the authenticated identity of the server device by sending it any ReadProperty (e.g., reading the Device object's Object_Identifier property using the wildcard instance) with an attached 'Secure Source' option and noting the 'Secure Source' option in the response.

XX.2.7 Identity Tokens

An identity token ties a unique "subject" name, for example, as provided by an X.509 certificate, to the device instance number that is attested and signed by the identity server. In this way, a device's claim of its instance number is validated by a trusted third party.

The data structure is a BACnetWebToken and it is signed by the identity server. The semantic meaning of the claims is defined by the IANA "JSON Web Token Claims" (JWT) registry, which contains references to the standards that define each claim. Many of the registered JWT claims are character strings for flexibility in various contexts, and it is expected that derived formats, like the CBOR Web Token, profile them to more restrictive types. For BACnet, the datatypes are profiled for compactness and range limitations.

Field	Datatype	Description
authorized-party	Unsigned	the attested instance number of the device.
confirmation	BACnetConfirmation	The 'key-id' field of this structure is the "subject" name.
scope	CharacterString	Contains at least "id" and may contain others, e.g., "id router"
audience	array of BACnetAudience	Constant audience: One member specifying "Group 1" (everyone).
issuer	Unsigned	The device instance of the identity server
issued-at	Unsigned	The time the token was issued (unix epoch seconds)
expiration	Unsigned	The time the token expires (unix epoch seconds)

When an identity token is sent to establish identity, the receiver shall verify the token by checking first that the 'key-id' field in the 'confirmation' field matches the 'subject' name of the sending device. This process is datalink specific. For example, for BACnet/SC, this is done by matching with the 'subject' distinguished name in the sender's TLS certificate. Then the signature is validated using the signing key of the Identity_Server property in the receiver's Auth object. If validated, the receiving peer will remember the 'authorized-party' field as the authenticated device instance of the sending peer.

The 'scope' of an identity token conveys information about the capabilities of the bearer that have been approved by the identity server. The possibilities are:

"id"	every identity token has this, and non-identity tokens do not.
"hub"	the port can function as a hub
"router"	the device can function as a router
"authz"	the device can function as an authorization server
"authn"	the device can function as an identity server

The identity token is created by the identity server based on its knowledge of the mapping between the device's subject name and the device instance. The identity token is either actively requested by the device using the AuthRequest service or is configured into the Auth object's Identity_Token property by some means. Note again that identity tokens are not secrets and conveying them does not need to be concealed.

For confirming a BACnet identity token, this standard uses the "kid" confirmation method of the "cnf" claim as defined in RFC 7800. This field is defined by the RFC to simply identity a key by some means. For BACnet, the private key is identified indirectly using the device's 'subject' name. The means by which the subject name references a private key is datalink specific. For datalinks that use TLS, e.g., BACnet/SC, the "proof of possession" of the corresponding private key is established by the X.509 certificate by the path illustrated below. Thus, only the device in possession of the correct private key can successfully use the identity token.

```
Identity Token           Certificate             Private-Key
------------------       ------------------      -------------
azp: 12345
cnf: {kid: "CN=..."} --> Subject: "CN=..."
                         Public-Key: 4C3E... --> x: 56E2... y: 56E2...
```

XX.2.8 Proving Identity

The identity of a device is proven at the datalink level, where the messages are directly sent and received between peers without intervening hubs or routers. An identity token is validated by checking the signature of the token using one of the keys in the Identity_Server property of the receiving device's Auth object and comparing to the token's device instance attestation to the

device instance claim made by the sending device. Each secure datalink defines the details of this validation process. For example, BACnet/SC uses the TLS connection to securely obtain the certificate's 'subject' name and then compares that to the name in an identity token provided as part of the connection establishment process. See Clause AB.X.1.

XX.3 Device Authorization

The 'Secure Source' data attribute provides the identity (device instance) of the originating device. In some simple cases, this might be sufficient to allow certain protected operations; for example, by using a simple allow-list of device instances in the destination device. However, it is generally not desirable to have authorization policies distributed into the end devices themselves, as synchronization and management of those policies becomes a burden for the user, and temporary authorization (e.g., service technician) requires making undesirable temporary and potentially widespread adjustments to the end devices needing service.

Therefore, this clause describes how authorization decisions can be consolidated into an authorization server that contains the authorization policies covering a large number of end devices. Client devices can request permission to perform a protected operation by requesting an access token from the Authorization Server. An access token contains an intended audience (where it can be used) as well as a scope (what actions can be done there). The audience is either a single device or a group. The scope can contain multiple permissions for various classes of actions that the client is allowed to perform.

When a client device wants to include an access token along with an NPDU, it includes it as a data attribute on the message. This attribute will get routed along with the NPDU as long as the message does not leave the internetwork of secure datalinks. For efficiency, access tokens can be cached and then either referenced explicitly or implied (referenced implicitly). See Clause XX.3.6~~XX.2.6~~

Note that access tokens and authenticated devices instances ('Secure Source') are independent. The authenticated device instance alone might be sufficient for some operations (e.g., secure I-Am); however, other operations might require an appropriate access token.

Even though authentication and authorization are structurally independent, access tokens are bound to the client device's instance so they cannot be stolen and used by another client device (i.e., they are not "bearer" tokens). Therefore, the receiving device will not accept an access token unless the sender is authenticated by an appropriate 'Secure Source"' attribute.

XX.3.1 Access Tokens

To perform protected operations on a target device, the client device needs to present an access token that is conveyed before, or simultaneously with, the requested operation. Access tokens are conveyed as a data attribute accompanying an NPDU. Access tokens can be cached to reduce bandwidth for repeated use. See Clause XX.3.6~~XX.2.6~~.

The data structure of an access token is a BACnetWebToken and it is signed by an authorization server that the resource server trusts. The semantic meaning of the claims is defined by the IANA "JSON Web Token Claims" registry, which contains references to the standards that define each claim. Many of the registered claims are character strings for flexibility in various contexts, and derived formats, like CBOR Web Token, profile them to more restrictive types. For BACnet, the datatypes are profiled for compactness and range limitations.

Field	Datatype	Description
confirmation	BACnetConfirmation	The confirmation (binding) of the token. The 'authorized-party' field of this structure is the device instance that the token is bound to.
scope	CharacterString	Space-separated list of permissions granted by the token
audience	array of BACnetAudience	The target audience of the token. Can be multiple devices and/or groups
issuer	Unsigned	The device instance of the identity server
issued-at	Unsigned	The time the token was issued (unix epoch seconds)
expiration	Unsigned	The time the token expires (unix epoch seconds)
subject	CharacterString	The "user" that requested the token. "0 0" means "the device itself".

If the token passes the validity checks described in Clause XX.3.2~~XX.2.2~~, then the recipient will compare the presented 'scope' values to the scope permissions required for the requested action. The naming and meaning of scopes in BACnet match those in BACnet/WS, as specified in W.3.5.

The 'subject' claim is intended to be used for auditing purposes only. All authorization decisions at the resource server should be based on 'scope'. The assumption is that the authorization server has already examined the user and role in the 'subject' has decided, based on its own policies, whether to issue the token or not. So, the resource server does not need to make further checks with its own policies. See Clause YY.4~~YY.2.4~~ for more about "users".

Access tokens are bound to the device that is authorized to use them. The 'confirmation' claim binds the access token to an identity token. The identity token is bound to the device in a manner that is datalink specific. For example, in BACnet/SC, the identity token is bound, via the X.509 certificate, to the private key used for TLS. In this way, only the device possessing that private key is allowed to use the access token.

```
Access Token          Identity Token           Certificate              Private-Key
------------          ------------------       -------------------      -------------
sub: "32 2"
cnf: {azp: 12345} --> azp: 12345
                      cnf: {kid: "CN=..."} --> Subject: "CN=..."
                                               Public-Key: ...     --> x: ... y: ...
```

XX.3.2 Audience

For added security, access tokens are not only bound to the device that can send them, but they are also bound to the device or group of devices that can receive them. The latter binding is done with the 'audience' field in the token. The audience is a list of BACnetAudience specifiers. Each one can specify a device or a device group and an optional "application" within that device or devices. A BACnetAudience construct contains a 'target' field which is a choice of a 'device' or 'group'. The 'device' choice is the instance number of the target device. The 'group' choice is a site-specific group number. Group numbers 0 and 1 are reserved to have special meanings. See Clause XX.3.9.

XX.3.3 Application

Some BACnet devices possess distinct software entities representing an "application", "processes", "API", "tenant", "vertical", etc. To support addressing this "sub-audience", each BACnetAudience construct also has an 'application' field. This is a CharacterString that identifies such an entity within the device. The values for the string are a local matter. The 'application' field is optional. If absent, it defaults to meaning "all applications". The valid combinations of 'scope' permissions and 'application' specifications is a local matter. Since this information might be unknown to the client, the "hint" mechanism can be useful for identifying what 'application' and 'scope' is required for a protected operation. See Clause XX.3.8~~XX.2.8~~.

XX.3.4 Scope

The permissions granted by an access token are contained in its 'scope' field. This has the semantics of the standard OAuth "scope" claim. It is a space-separated list of scope identifiers. See Clause W.3.5 for a description of standards scope identifiers and provisions for proprietary extensions. The valid combinations of 'scope' permissions and 'application' specifications is a local matter. Since this information might be unknown to the client, the "hint" mechanism can be useful for identifying what 'application' and 'scope' is required for a protected operation. See Clause XX.3.8~~XX.2.8~~.

XX.3.5 Validation

To validate an access token, the following steps are performed. This procedure is written as a "function" that "returns" a value that is an error code for the SECURITY error class. This does not imply that an Error PDU is sent with this code. The actions to take when an access token is invalid is context dependent and often, an invalid token will simply be ignored as if it had not been sent and the normal error response is returned indicating that a protected operations failed. The "hint" mechanism can be used to indicate that a correct access token is needed. See Clause XX.3.8~~XX.2.8~~.

When an access token is received, the receiving device shall:

1) check that a 'Secure Source' data attribute accompanies the access token. If not, then return the error code SOURCE_SECURITY_REQUIRED.
2) check that the receiving device's device instance or one of the groups in the Device_Groups property of the Auth object is present in one of the elements of the 'audience' field and that the 'application' field of that entry is recognized. If not, return INCORRECT_AUDIENCE.
3) compare the 'authorized-party' field of the token's 'confirmation' field with the 'Secure Source' data attribute. If there is a mismatch, return the error code INCORRECT_ INSTANCE.
4) check the access token header for a supported 'algorithm' field value. If the value is not supported or not accepted in this context, return the error code UNKNOWN_AUTHENTICATION_TYPE.
5) get the 'key-id' field of the token's header and verify that a key with that identifier is available in either the Primary_Authorization_Server or Failover_Authorization_Server property of the Auth object. If none is available, return the error code SECURITY_NOT_CONFIGURED.
6) validate the signature using the identified key. If the signature is invalid, then return the error code BAD_SIGNATURE.
7) if all tests pass, return the error code SUCCESS.

XX.3.6 Conveying

An access token is conveyed in a 'Token' data attribute. The 'Token' attribute is designated as a "first segment" attribute. The attribute consists of two fields: 'reference-identifier' and 'token'. The 'reference-identifier' field is a CharacterString whose content is not restricted with the exception that empty string and a single hyphen "-" are defined to have special meanings. See Clause XX.3.7~~XX.2.7~~. The 'token' field is a BACnetWebToken construct representing the value of the access token. The 'token' field is optional and can be omitted for the cache control described in Clause XX.3.7~~XX.2.7~~.

XX.3.7 Caching

The implementation of token caching is optional, but it is recommended to have at least one cache slot in a resource server to allow bandwidth savings for repeated operations from at least one client.

When a client sends an access token for the first time to a resource server, is indicates its desire for the token to be cached using the 'reference-identifier' field in the 'Token' attribute. If the 'reference-identifier' is equal to a single hyphen character ("-"), then the client is instructing the resource server not to cache the token. If the 'reference-identifier' is equal to an empty string (""), then the client is indicating that this is the "default token" that will be implicitly referenced by future messages.

If a client has previously sent a request to cache a token, it can send the brief 'Token Reference' attribute instead of the entire 'Token' attribute. The 'Token Reference' attribute consists of only the CharacterString that was used as a 'reference-identifier' in a previous 'Token' attribute. The 'Token' and 'Token Reference' attributes are mutually exclusive.

If no 'Token' or 'Token Reference' attribute is received with a message, then the "default token" will be implicitly referenced. If no default token has been cached, then the recipient will behave as if no token has been provided.

If a 'Token Reference' attribute is received but the referenced token is not found or has been purged, the recipient will behave as if no token has been provided.

If a 'Token' attribute is received and the 'reference-identifier' field matches an existing cache entry for that client and the 'token' field is present, the existing token will be replaced with the new one.

If a 'Token' attribute is received and the 'reference-identifier' field matches an existing cache entry for that client and the 'token' field is absent, the existing token will be removed from the cache.

If a 'Token' attribute is received and the 'reference-identifier' field is equal to a single hyphen character ("-") and the 'token' field is present, then that 'token' field shall be used for this one message and then discarded.

If a 'Token' attribute is received and the 'reference-identifier' field is equal to a single hyphen character ("-") and the 'token' field is absent, then all cached tokens for this client shall be removed from the cache.

See Clause YY.1 for recommendations on client-managed token reference identifiers.

XX.3.8 Hints

When a device returns an error indicating that a protected operation was not successful, for example, an error code of NOT_AUTHORIZED, it shall provide a 'Hint' data attribute with the response to authenticated devices indicate what kind of authorization is required for the requested operation. A 'Hint' attribute is designated as a "first segment" attribute.

If a resource is accessible with more than one audience or scope, then multiple 'Hint' attributes can be returned.

A 'Hint' data attribute contains the following fields:

Field	ACE-Oauth name	Datatype	Description
auth-server	"AS"	Unsigned	device instance of the authorization server
auth-server-alt	n/a	Unsigned	optional device instance of alternative authorization server
audience	"audience"	BACnetAudience	optional audience (defaults to this device)
scope	"scope"	CharacterString	scope required for the operation

The required 'auth-server' field contains the device instance number of the authorization server that can issue access tokens for this device. The optional 'auth-server-alt' field contains the device instance number of an alternate authorization server that can also issue access tokens for this device. Both the authorization servers provided in the 'Hint' shall have identical policies for issuing access tokens.

The 'audience' field is optional. If absent, it defaults to the device instance of the resource server and an empty 'application' string, indicating all applications on the resource server.

For operations that can potentially return multiple authorization errors in a single response, e.g., ReadPropertyMultiple, the returned hint attribute shall contain the authorization information needed for the first failed item.

XX.3.9 Groups

The members of the 'audience' field of an access token refer to either a single target device or to a group of devices. For a single device, the 'device' field is a device instance in the range 0-4194302. For group audiences, the 'group' field is a group number that is site specific in the range 2-4194302. Group 1 is defined to be the "everyone" group. The value 0 is not a valid group number and can be used to indicate "no group", e.g., in the Groups property of the Auth object.

A device shall only accept an access token if its own device instance is included in the 'device' field of an 'audience' member, or if the 'group' field an 'audience' member matches one of the group identifiers in the Groups property of the Auth object.

XX.3.10 Users

To meet the requirements of 'user-id' and 'user-role' in the Auditing object and their definition in Clause W.3.11, the value of 'user-id' and 'user-role' must be consistent site-wide. Therefore, a multi-user client shall support a configurable 'user-id' and 'user-role' for each of its users or shall support a method for retrieving that information for an authenticated user (e.g., LDAP). If the 'user-id' cannot be configured or otherwise obtained, then the 'user-id' in the authorization request shall be 0. If the 'user-

role' cannot be configured or otherwise obtained, then the 'user-role' in the authorization request shall be 1 if the action is known to be caused by a human or 0 otherwise. See Clause W.3.11.

XX.4 Data Types for Authentication and Authorization

Since the authentication and authorization mechanism defined in this annex is a profile of the Oauth and ACE-Oauth standards, several of the data structures used here mirror those defined by the relevant RFCs and IANA registries. This clause defines how these structures are used in BACnet.

XX.4.1 BACnetWebToken

A BACnetWebToken matches the structure, semantics, and capabilities of an RFC 7519 JSON Web Token, but is encoded in ASN rather than JSON. The token consists of three main fields: header, claims, and optional signature.

BACnet Field	Datatype
header	BACnetOSEHeader
claims	BACnetClaimsSet
signature	OctetString

The header corresponds to the semantics and capabilities of the JOSE header defined in RFC 7515 (JWS). The JWS payload in a JWT is a "Claims Set", as defined in RFC 7519. The signature shall be present if the 'algorithm' field in the header has a value other than "none".

XX.4.2 BACnetOSEHeader

This structure is used as the header of a BACnetWebToken, matching the definition of the JOSE header in RFC 7515 (JWS), RFC 7516 (JWE), and RFC 7519 (JWT).

BACnet Field	JOSE Name	Datatype
extension	n/a	SEQUENCE OF BACnetNameValue
type	"typ"	CharacterString
algorithm	"alg"	CharacterString
key-id	"kid"	CharacterString

For brevity in BACnet representations for common use cases, some claims are given default values by this standard and can be omitted from BACnet encodings even if they are indicated as "REQUIRED" by the appropriate RFCs.

The default value for 'type' is "BWT". Since this is almost always determined by context, as indicated by RFC 7519 section 5.1, 'type' will likely not be included for BACnet Authorization and Authentication services.

The default for 'algorithm' is "ES256". Support for the default values is required. Support for other values is a local matter requiring agreement between the communicating parties.

Devices shall not accept an 'algorithm' of "none" unless specifically configured to do so for a particular use case. All standard access tokens and identity tokens use an 'algorithm' other than "none".

The 'key-id' in the header can be used to match the 'key-id' member in a BACnetWebKey that can be used to verify the signature.

XX.4.3 BACnetClaimsSet

The body of a BACnetWebToken is a “Claims Set”, as defined by RFC 7519 (JWT). This is used for access tokens and identity tokens.

BACnet Field	JWT Claim	Datatype	Notes
extension	n/a	SEQUENCE OF BACnetNameValue	
issuer	"iss"	Unsigned	RFC 7519 – 4.1.1
audience	"aud"	SEQUENCE OF BACnetAudience	RFC 7519 – 4.1.3
scope	"scope"	CharacterString	RFC 6749 – 3.3
subject	"sub"	CharacterString	RFC 7519 – 4.1.2
confirmation	"cnf"	CharacterString	RFC 7800 – 3.1
expiration	"exp"	Unsigned	RFC 7519 – 4.1.4
issued-at	"iat"	Unsigned	RFC 7519 – 4.1.6
not-before	"nbf"	Unsigned	RFC 7519 – 4.1.5
no-cache	n/a	BOOLEAN	see below

The 'no-cache' field is defined by this standard so that authorization servers can instruct resource servers not to cache an access token when received from a client, even if the client does not specify "-" in the 'Token Reference' field of a 'Token' data attribute.

XX.4.4 BACnetHint

A failed service request can return a "hint" to indicate what authorization is required for the operation to succeed. The hint provides an indication of which authorization server to contact for an access token and the required scope and audience that is required. Refer to Section 5.3 of ACE-Oauth.

BACnet Field	ACE-Oauth Name	Datatype
extension	n/a	SEQUENCE OF BACnetNameValue
primary-as	"as"	Unsigned
failover-as	n/a	Unsigned
audience	"audience"	BACnetAudience
scope	"scope"	CharacterString

XX.4.5 BACnetWebKey

BACnet cryptographic keys are stored in a structure that corresponds to the semantics and capabilities of an RFC 7517 JSON Web Key but is encoded in ASN rather than JSON.

BACnet Field	JWK Claim	Datatype	Default Value
extension	n/a	SEQUENCE OF BACnetNameValue	
key-id	"kid"	CharacterString	
usage	"use"	CharacterString	"sig"
key-type	"kty"	CharacterString	"EC"
key-ops	"key-ops"	SEQUENCE OF CharacterString	["sign","verify"]
algorithm	"alg"	CharacterString	"ES256"
curve	"crv"	CharacterString	"P-256"
x	"x"	OctetString	
y	"y"	OctetString	
d	"d"	OctetString	
scope	"scope"	CharacterString	

For brevity in BACnet representations for common use cases, some claims are given default values by this standard and can be omitted from BACnet encodings even if they are indicated as "REQUIRED" by the appropriate RFCs. For future crypto-agility, a BACnetWebKey can contain alternate claims or non-default claim values, e.g., alternates selected from RFC 7518. Support for the default values is required. Support for other values is a local matter requiring agreement between the communicating parties.

In addition to the common parameters defined by RFC 7515, each BACnetWebKey can have members that are specific to the specified key type. RFC 7518 defines multiple kinds of cryptographic keys and their associated members. The default key type in this specification is "EC". Therefore, BACnetWebKey provides entries for "crv", "x", "y", and "d".

The 'key-id' field in this structure can be used to match this key with the signature in a BACnetWebToken which also has a 'key-id' field in its header. The 'key-id' field is a UTF8 string whose content is a local matter with the exception that, unless otherwise noted, its encoded value (the portion after the character set identifier) shall not exceed 16 octets.

XX.4.6 BACnetConfirmation

The BACnetConfirmation construct matches the semantics and capabilities of an RFC 7800 'cnf' claim. A token is bound to the device that is authorized to use it via one of the "confirmation methods" in the 'confirmation' field of the token.

The 'key-id' confirmation method is used by identity tokens to bind the token indirectly to a key in the possession of the device by referring to a "subject" name that is bound to a private key by some datalink-specific means, e.g., an X.509 certificate used for a TLS connection.

The 'authorized-party' confirmation method is used by access tokens to bind the token to a device instance number that is provided securely by some means, e.g., by a 'Secure Source' data attribute.

BACnet Field	JWT Claim	Datatype
extension	n/a	SEQUENCE OF BACnetNameValue
key-id	"kid"	CharacterString
authorized-party	"azp"	Unsigned

XX.4.7 BACnetSigner

The BACnetSigner construct is used to associate a key with the device that owns it. To allow key rotation, this construct optionally contains two keys. A device can hold multiple keys for different purposes, so this construct does not imply that these are the only keys held by the specified device. If the 'device' field contains 4194303, then the BACnetSigner is unconfigured and the keys shall be ignored.

BACnet Field	Datatype	Notes
device	Unsigned	
key1	BACnetWebKey	
key2	BACnetWebKey	optional

[Add new **Annex YY**, p. 1403]

ANNEX YY - Managing Authorization (INFORMATIVE)

(This annex is not part of the standard but is included for informative purposes only.)

This annex provides guidance and recommendations for interoperable applications of the authentication and authorization mechanisms defined in Annex XX.

YY.1 Managing Caching

There is no indication of whether a token has been cached or not, and a resource server is free to purge tokens from its cache for any reason at any time. Therefore, the use of a 'Token Reference' is considered an "optimistic" optimization by the client. If the use of a 'Token Reference' fails repeatedly, the client can conclude that caching is not reliably available, and that it needs to send the full 'Token' with every message. See also the specification of groups and group tokens in Clause XX.3.9~~XX.2.9~~.

A client device can define and manage 'reference-identifier' values in any manner that works for its design. For example, it can be a persistent or reproduceable identifier of a particular function and user within the client device. Simple devices that only use one token can simply use the default (empty string) 'reference-identifier'. Because the client assigns the meaning to the 'reference-identifier', it can be thought of as a "slot" identifier. So, the client can replace an existing "slot" without needing to remember the token previously sent for that slot. For example, the client might have lost the previous token during power cycle but uses a persistent 'reference-identifier' for a given function. By using repeatable 'reference-identifier' values, "abandoned" cache slots do not accumulate in the resource server.

Some examples of token cache requests by the client:

a) A simple client might only have one process and use one token. It would just use the default 'reference-identifier' (empty string) always. Whenever it gets a new token, it just sends it to the resource server, and the old one is automatically purged/replaced.
b) A client device might have two independent daemon threads that take actions with separate permissions, and neither has a "user". In this case, the 'reference-identifier' serves as a "sub-client" or "process" identifier.
c) A client might have the concept of "users", but only *one* is logged in at a time. In this case, all "users" would use the same 'reference-identifier' serially and would naturally want an old user's token to be replaced with the new user's token.
d) A complex multi-user client with multiple *simultaneous* users, might chose to do its token management by making its 'reference-identifier' related to the user-id. The 'reference-identifier' is a string to allow flexibility of the client to incorporate the user-id in addition to an internal function identifier, e.g., "trnd-42".
e) When a complex client is finished with a token, perhaps one that was retrieved for a temporary privilege elevation, it can remove the token from the resource server. e.g., it normally uses "ctrl-42" for control functions associated with user 42, but temporarily uses "conf-42" for a configuration operation for the same user. After the configuration operation is concluded, "conf-42" can be removed from the resource server.

YY.2 Managing Hints

A hint should contain the least privilege that is needed for the requested operation. By doing so, it will contain the resource and scope that the authorization server is most likely to grant to the client. Returning a hint that contains more privilege than is required for the requested operation might result in the client not being able to get an access token to complete the request.

YY.3 Managing Scope

The behaviors of both the client and the authorization server can affect the set of permissions that are granted by the 'scope' of an access token. BACnet supports both simple and complex clients and authorization servers. However, it is generally expected that an authorization server is a large, sophisticated device with a powerful user interface used for the configuration of policies that define which devices can perform which protected operations in which target devices, or groups of devices.

YY.3.1 Client Managed Scope

To reduce the number of tokens that a client need to manage, the client can control the scope that it asks for from the authorization server.

The general behavior for a protected operation is:

1) The client attempts the protected operation.
2) The operation fails and the resource server returns a 'Hint' attribute along with the error response.
3) The client uses the 'Hint' to contact the authorization server and asks for the required scope and audience.
4) The client retries the operation providing the token
5) The operation succeeds.

This results in a token for that specific protected operation, but if the client has many independent operations for a given resource server, this may result in too many tokens for the resource server to cache. Therefore, if a client already has a token for a given user at a resource server, and it receives a 'Hint' from that same resource server, then an additional scope entry is needed for a new operation, and the client can attempt to combine the new scope entry with the existing scope entries that it already has to obtain a single token that works for both operations.

For example, if the identity token already has scope of "adjust" and is given a hint that it needs scope of "configure", it can make an authorization request for scope "adjust configure". If the authorization server grants that request, then the client can use the new token in place of the original one. Note that a resource server will replace an existing token in its cache with a new one if the new token has the same 'reference-identifier', so the client does not have to specifically un-cache the old token.

Such scope combination can be done to reduce bandwidth and avoid token "thrashing" caused by limitations of the resource server's caching ability. However, it is making a trade-off between the doctrine of "least privilege" and performance. Therefore, it should only be done when the two scopes are needed by the same user for the same function within the client. It is a local matter how an advanced client handles the goal of "least privilege" when there is the possibility of multiple tokens for the same user at a resource server for multiple distinct functions within the client. For example, if a highly privileged operation is needed only temporarily (e.g., back-up/restore), the original lower-scoped token should be retained and then restored to the resource server after the operation is complete, or separate 'reference-identifier' could be used to separate two functions for the same user while retaining the cache of both tokens at the resource server.

YY.3.2 Authorization Server Managed Scope

A client device should not attempt to expand that scope beyond what it has already been granted since it may be denied an access token for the expanded scope by the authorization server. The authorization server, however, may have a "bigger picture" of the needs of a client. Therefore, when the client makes an authorization request for a given scope, it is possible that the authorization server has been configured to know that the client will very likely be following that shortly with a request for a different scope because it knows everything that the client will need for its normal operation. Issuing the two tokens separately could cause unnecessary token thrashing, caused either by resource servers with limited cache space or clients with limited token management capabilities.

An authorization server can return less scope than is requested, based on its policies. Since this can result in the client being unable to perform its actions, a reduction in scope should be reported for administrative analysis. When the requested scope cannot be granted, the authorization server shall not return an error, but rather shall reduce the scope to the level that can be granted, even if that becomes an empty string.

An authorization server shall only return an expanded scope if it is specifically configured to understand the 'purpose' parameter for the client or if it has been configured to return a fixed token for the combination of client, resource server, and audience, i.e., it shall not automatically "OR together" all the possible scopes that the client can have for the target resource server.

An authorization server can return a 'group' identifier instead of the requested 'device' identifier in the token's 'audience' if it is configured to know that the request requires talking to multiple devices that have been assigned to a group. This way, the client asks for a token for the first member of the group that it encounters, and the authorization server returns a token that will also work for the rest of the members that will be encountered in the future. A simple device that does not process hints can always "try" any token that it has when it receives an error. However, it is better for bandwidth and performance if a device that is expecting to talk to a group specifically asks for a group token.

YY.3.3 Use of the Purpose Parameter

The policies for "knowing what the client will need" in the authorization server is a local matter dependent on the capabilities and design of the authorization server. It can be based solely on the identity of the client and the resource server for simple single-function client devices. However, for advanced clients that may have multiple functions, the authorization server might need an indication of what function within the client is making the request. This indication is provided by the 'purpose' parameter in an authorization request.

Using the 'purpose' parameter allows very simple devices to not be required to first attempt an operation and then handle the 'Hint' attribute in an error response. For example, the following single-function simple device scenario is possible:

1) When the device powers up, it doesn't "attempt" to do anything. It just makes an authorization request to the authorization server, specifying its 'purpose' parameter without any requested target or scope.
2) The authorization server looks up the client and purpose in its database and returns a token containing the required audience and scope.
3) The device uses that token to accomplish that "purpose".
4) If it fails, this simple device just delays and tries the whole thing again. It doesn't need to process "hints" from the resource servers.

YY.4 Managing Users

The device-to-device authorization defined in Annex XX is simple and robust, but with a widely applied protocol like BACnet, is not uncommon to find an action that is initiated by a human operator. And some deployments, like pharmaceutical facilities, want the "who" to be audited along with the "what" and "where". Therefore, BACnet's implementation of the Oauth authorization flow includes the concept of an "authorized user". To support multiple users in BACnet, both the token request and the returned access tokens contain information that indicates the identity of the user that caused the action.

The authentication of "users", however, is not in scope of device authentication. While Annex XX is based on Oauth, it is restricted to the "client credentials" flow and does not use HTTP "redirects" to allow a human to "log in" to another server. Therefore, it is a local matter how a client device authenticates any users that may be causing the initiation of actions by the client. Some clients may be simple PIN-based authentication, and some may use sophisticated user authentication methods such as LDAP, smart cards, etc.

In an access token, the 'subject' field contains two parts, a 'user-id' and a 'user-role' identifier, as defined in Clause W.3.11. This claim identifies which user has been granted the access token. When a token is presented to a resource server, this information should be used for auditing purposes only. All authorization decisions at the resource server should be based on the 'audience' and 'scope' fields in the token. The assumption made here is that the authorization server has already decided whether to issue the token or not, so the resource server does not need to make further checks.

The trust the resource server has of the validity of the user-id and user-role fields is based on the fact that the authorization server granted the token and therefore trusts the client. If the authorization does not trust that the client properly authenticates its users, then it will not grant a token containing a 'subject' field other than "0 0", indicating "the device itself".

Note that the user-role part can be used when a specific user is not known. For example, a PIN based device might only know rolls, e.g., "occupant" and "technician". It could be programmed to use role 1 and 2 for these, and the authorization server would place a corresponding 'subject' in the token of "0 1" or "0 2". See Clause W.3.11 for the definition of user-id and user-role.

In an authorization request, a multi-user client sends the user-id and user-role of the user making the request to the authorization server. The authorization server consults its policies as to whether to issue an access token for that user or role, on that client, for the requested operation, on the target device. The authorization server may replace the user-id and/or user-role in the token based on its own configuration information.

For proper security and auditing integrity, a multi-user client must maintain a separate collection of access tokens for each user and shall not use a token requested for one user to perform an operation initiated by another user. The resource servers can

trust that the authorization server considers the client to be trustworthy enough to handle this properly since it is ultimately up to the authorization server to decide whether it gives out multiple tokens to a single client or not.

When a client device has the potential for multiple users taking actions at the same time, e.g., a web server, all the BACnet messages for all the users is originating from same client device. Therefore, the authorization server has to trust that the client device is keeping the users separate and is not letting one user access a token that was issued to another user, and resource servers have to trust that the token presented in a request matches the user that initiated the request. The degree to which the client is able to keep the separate tokens matched with the logged-in users, and isolated from other users (separate process memory spaces, per-thread TPM compartments, etc.), is not usually addressed by standards like Oauth, but is it acknowledged that authorization servers may choose to trust some clients more than others.

If the site policy allows that per-user auditing is not needed in any end devices, all clients could be configured to set user-id to 0 and user-role to 1 for all users. Therefore, these should be the default values. To further reduce the number of tokens in a multi-user environment, the authorization server can take advantage of the identification of the intended target device or group in the authorization request. If the authorization server knows that the targeted device/group does not need any auditing information at all, it can issue the same "0 0" token for all users requesting access to that device/group and caching in the target device will be optimized.

YY.5 Acquiring Access Tokens

A device typically acquires an access token by querying the Authorization Server (possibly after being rejected from the target device and receiving a "hint"). The authorization server must be reachable as a BACnet device and support the AuthRequest service defined in Clause 16.X. Alternative methods of requesting an access token (e.g., HTTP, alternate Oauth flows) may be used but are outside the scope of the specification in Annex XX.

Authorization servers, like hubs and routers, have an additional attestation, "authz", in the scope of their identity token. This lets directly connected clients know that they have connected to an authorization server and not accidentally to some other device. If needed, non-directly connected clients can read the Identity_Token property of the server's Auth object to determine its scope. If the 'auth-server' entry in a 'Hint' data attribute option points to a device that does not have "authz" in its scope, this would be considered an erroneous configuration.

Note that access tokens, like identity tokens, are effectively "public information". They are not "bearer tokens" and thus are not secrets. They are "sender-bound" and can only be used by the client identified in the token. Eavesdropping or theft is not a problem. And they are integrity protected with a signature so they cannot be modified. Therefore, the acquisition of them is not critical, and so the client to authorization server communications is not critical, other than possible denial of service by a bogus authorization server handing out unusable tokens.

YY.5.1 Acquiring Tokens On-Behalf-Of Another Device

For situations where a client cannot get its own access tokens for some reason, a "helper" device can request access tokens from the authorization server "on behalf of" the client that will eventually use them. To make this possible, the authorization request can include the client device instance as an explicit request parameter, rather than assuming that the requestor is to be the client. The authorization server's policies will decide whether to issue a token bound to a device other than the one that is making the request.

The method by which the "helper" provides the access tokens to the client is a local matter, and the method by which the "helper" knows what access tokens are needed by the client is also a local matter.

YY.6 Managing Groups

The use of non-standard scope identifiers for group operations can make issuance of access tokens for the group devices difficult or impossible depending on the capabilities and limitations of the authentication server and the makeup of the group. If different devices use different scopes for the same operation, the access token will need to contain all those scopes. This can lead to possibly undesirable or unsecure overlaps or conflicts. It is therefore strongly recommended that devices use the pre-defined standard scope identifiers defined in Clause W.3.5 for group operations, wherever possible.

Likewise, the use of a non-empty audience 'application' field for group operations makes issuance of a single access token for the group of devices impossible if the devices expect different application identifiers for the group operation. It is therefore strongly recommended that devices use an empty, or configurable, application identifier for their group operations wherever possible.

It is a requirement that all members of a group use the same authorization server so that the 'iss' in the access tokens is understood by all destinations and the signature can be verified.

It is also required that all members of a group understand a given access token (i.e., the same or compatible token version, no mandatory proprietary extensions, etc.). The means for verifying this when setting up a group is beyond the scope of this specification.

135-2020*cg*-2 Changes to BACnet/SC to Support Authentication and Authorization

Rationale

BACnet authentication is enforced at the datalink level, and authorization uses data attributes defined in Clause 5 and 6. Both require changes to BACnet/SC connectivity and transport mechanisms.

[Add new **Clauses AB.2.3.X1-6**, p. 1378]

AB.2.3.X1 Hello Header Option

The 'Hello' header option conveys revision and capability information along with an optional identity token that binds a device instance to the 'subject' distinguished name in the presenter's TLS certificate. The identity token may be absent. In this case, the absence of the token indicates to the connection peer that the concept of identity is understood but none is currently available or appropriate.

The 'Hello' destination option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Must Understand' = 0, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Device Instance	3-octets	The device instance, most significant octet first.
Capabilities	2-octets	A set of bit flags indicating capabilities of the protocol.
Identity Token	Variable	An optional identity token. Can be zero length.

The 'Capabilities' flags do not currently have meaning and all bits shall all be FALSE.

This header option, if present, shall be a destination option in the 'Destination Options' parameter of the Connect-Request and Connect-Accept messages.

AB.2.3.X2 Secure Source Header Option

The 'Secure Source' header option conveys a device instance of the originator of the message. If this claim is determined to be untrustworthy, its Header Option Type shall be changed to 'Nonsecure Source'.

The 'Secure Source' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 1, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Device Instance	3-octets	The device instance, most significant octet first.

AB.2.3.X3 Nonsecure Source Header Option

The 'Nonsecure Source' header option conveys either a previously claimed 'Secure Source' that has been invalidated somewhere along the path by receipt from an unauthenticated peer, or a device instance that has been applied by a router based on its knowledge of the device instance of the source. This option shall not be used for authentication decisions since it is not fully trusted. This option shall never be originated by a device.

The 'Nonsecure Source' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 1, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Device Instance	3-octets	The device instance, most significant octet first.

AB.2.3.X4 Hint Header Option

The 'Hint' header option conveys information to a client about what kind of access token is required.

The 'Hint' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 0, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Hint	Variable	A BACnetHint ASN.1 construction. See Clause 21.

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

The hint data contains an identification of which Authorization Server(s) to contact, the required resource path, and the required scope.

AB.2.3.X5 Token Header Option

The 'Token' header option specifies a BACnetWebToken that accompanies an NPDU.

The 'Token Identifier' field is a client-defined identifier for this token that can be used in a subsequent 'Token Reference' option. It consists of four octets whose contents are not restricted.

The 'Token' field is an optional ASN-tagged BACnetWebToken. It may be omitted if the intent is to remove a previously cached token.

The 'Token' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 0, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Token Identifier	4-octets	Client-defined reference identifier for the token.
Token	Variable	Optional BACnetWebToken ASN.1 construction. Can be zero length.

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

AB.2.3.X6 Token Reference Header Option

The 'Token Reference' header option specifies a token identifier of a previously sent, and possibly cached, BACnetWebToken that implicitly accompanies a NPDU.

The 'Reference Identifier' field is the client-defined identifier from a previously sent 'Token' header option. It consists of four octets whose contents are not restricted.

The 'Token Reference' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 0, 'Header Data Flag' = 1, 'Header Option Type' = n
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data		
Reference Identifier	4-octets	Client-defined reference to a previously sent token.

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

[Change **Clause AB.2.10**, p. 1381]

AB.2.10 Connect-Request

This unicast BVLC message is sent to the connection accepting peer node to request acceptance of the connection established. See Clause AB.6.2.

The Connect-Request message shall be sent with a 'Hello' option in the 'Destination Options' field. This option can contain an optional identity token to validate the 'Device Instance' field. See Clause AB.8.

Note that devices prior to Protocol_Revision X are not required to send the 'Hello' option.
...

[Change **Clause AB.2.11**, p. 1381]

AB.2.11 Connect-Accept

This unicast BVLC message is the response to the Connect-Request message. It is sent to the connection requesting peer node to confirm acceptance of the connection established. See Clause AB.6.2.

If the Connect-Request message was sent with a 'Hello' option, then the Connect-Accept message shall be sent with the accepting peer's information in a 'Hello' option in the 'Destination Options' field. This option can contain an optional identity token to validate the 'Device Instance' field. See Clause AB.8. If the Connect-Request does not have a 'Hello' option, this is an indication of an older initiating device, and the accepting peer shall not send one in its Connect-Accept message.

Note that devices prior to Protocol_Revision X are not required to send or receive the 'Hello' option.
...

[Add new **Clause AB.X**, p. 1410]

AB.X Connection Authentication

The authentication of a peer device is enforced at the datalink level since that is where the peer-to-peer messages occur with no intervening hubs or routers. BACnet/SC enables the identity claim specified in Clause XX.2.3~~XX.1.3~~ by enforcing the authentication and propagation rules specified in Clause XX.2.4~~XX.1.4~~.

The establishment of device authentication is optional; however, the device authorization functions described in Clause XX.3~~XX.2~~ will not work without device authentication.

After connection establishment with the Connect-Request and Connect-Accept messages, each connection instance information will contain an indication of whether the opposite peer is "authenticated" or "unauthenticated". This designation is based on

the opposite peer presenting a valid identity token during the connection establishment exchange. Identity is only exchanged with the Connect-Request and Connect-Accept messages. Therefore, once a connection is established, if an identity changes, the connection will need to be disconnected and reestablished.

To establish authentication with an accepting peer, the initiating peer shall send its identity token in the 'Hello' Option with the Connect-Request message, and the accepting peer shall send its token in return in the Connect-Accept message. See Clause XX.2.7~~XX.1.7~~ for definition of identity tokens.

If either peer does not yet have an identity token, or chooses not to include it, the 'Identity Token' field of the 'Hello' option shall be zero length. In this case, the receiving peer shall mark the sending peer as "unauthenticated".

If the initiating peer sent an identity token in the Connect-Request message, then the accepting peer shall validate the token with the rules in AB.X.1. If the token is valid, then the initiating peer shall be marked as "authenticated" in the connection context and its device instance shall be remembered. If the token is invalid, then a BVLC-Result NAK shall be returned indicating an 'Error Class' of SECURITY and 'Error Code' from the results from the validation in AB.X.1 and the connection shall be closed.

If the accepting peer sent an identity token in the Connect-Accept message, the initiating peer shall validate the token with the rules in AB.X.1. If the token is valid, then the accepting peer shall be marked as "authenticated" in the connection context and its device instance shall be remembered. If the token is invalid, then it is a local matter, based on its policies, whether the initiating peer proceeds with the connection, marking the accepting peer as "unauthenticated", or disconnects the connection.

It is possible that the initiating peer has determined by some means, such as a prior rejection, that its identity token is invalid or that the other peer cannot validate it. In this case, it can try the connection again by omitting identity token and continue with the connection in an "unauthenticated" mode. This allows a device with an invalid identity token or an unconfigured peer to nonetheless join the network for the purposes of notifying or correcting the situation.

AB.X.1 Validating an Identity Token

In this description, the terms "receiving peer" and "sending peer" are used since tokens are exchanged in both directions to/from the initiating peer, and the accepting peer and the rules are the same in both directions. This is written as a "function" that "returns" an error code. In the case of an accepting peer, these are used to send a BVLC-NAK response, but in the case of the initiating peer, the results are used internally.

When an identity token is received, the receiving peer shall:

1) compare the 'key-id' field of the token's 'confirmation' field with the X.509 'subject' distinguished name in the sending peer's TLS certificate. If there is a mismatch, return the error code INCORRECT_SUBJECT.
2) compare the token's 'authorized-party' field with the 'Device Instance' field in the sender's 'Hello' option. If there is a mismatch, return the error code INCORRECT_INSTANCE.
3) check the identity token header for a supported 'algorithm' field value. If the value is not supported or not accepted in this context, return the error code UNKNOWN_AUTHENTICATION_TYPE.
4) get the 'key-id' of the token's header and check that a key with that identifier is available in the Identity_Server property of the Auth object. If none is available, return the error code SECURITY_NOT_CONFIGURED.
5) validate the signature using the identified key. If the signature is invalid, then return the error code BAD_SIGNATURE.
6) if all tests pass, return the error code SUCCESS.

135-2020*cg*-3 AuthRequest Service

Rationale

Acquiring access tokens and identity tokens interoperably through the BACnet network requires a new service.

It is a single extensible service that initially has two functions: one for identity tokens and one for access tokens.

Identity servers and authorization servers might be separate devices, or they might be together in a single device - typically a large scale device (computer/server) with a database and full-featured user interface.

[Add new service Clause 16.X, p. 776]

16.X AuthRequest Service

The AuthRequest service is used by a BACnet-user to request an access token from an Authorization Server or an identity token from an Identity Server.

16.X.1 Structure

The structure of the AuthRequest service primitives is shown in Table 16-X1. The terminology and symbology used in this table are explained in Clause 5.6.

Table 16-X1. Structure of AuthRequest Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Request Parameters	M	M(=)		
Result (+)			S	S(=)
Response Values			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)
Error Details			U	U(=)

16.X.1.1 Argument

This parameter shall convey the parameters for the AuthRequest confirmed service request.

16.X.1.1.1 Request Parameters

This parameter, of type BACnetAuthRequest, provides the parameters required by the service, as defined in the Service Procedure below. The fields of this construct correspond to the OAuth request parameters defined in RFC 6749 and listed in the IANA "OAuth Parameters" registry, plus some BACnet extensions.

The fields of the BACnetAuthRequest are defined by the following clauses.

16.X.1.1.1.1 endpoint

The required 'endpoint' field serves as a sub-service selector and helps to verify that the optional fields are appropriate to the requested action. This serves a purpose similar to the path portion of an HTTP based protocol like OAuth. The 'endpoint' value "token" is analogous to the OAuth token endpoint and is used to request access tokens and identity tokens.

Servers shall only process 'endpoint' values that they understand. Simple endpoint names, like "authorize", "token", "refresh", "userinfo", etc., are reserved for definition by ASHRAE. Endpoint names defined by organizations other than ASHRAE shall use a prefix to ensure uniqueness. This prefix shall be either:

(a) A reversed registered DNS name, followed by a period character, e.g., "com.example.", or
(b) A BACnet vendor identifier in decimal, followed by a hyphen-minus character. E.g., "555-"

16.X.1.1.1.2 client-id

The required 'client-id' field contains the DeviceID of the device making the request.

16.X.1.1.1.3 extensions

The optional 'extensions' field contains proprietary extensions as name/value pairs. Servers shall ignore extensions that they do not understand.

16.X.1.1.1.4 response-type

The optional 'response-type' field identifies the kind of response expected from the token endpoint. This is equal to the OAuth "response_type" parameter. It is required for requests for access tokens or identity tokens.

The only registered values used by this standard are the IANA registered values of "token" and "id_token". Support for other possible values from the IANA "OAuth Authorization Endpoint Response Types" list is a local matter.

16.X.1.1.1.5 purpose

The optional 'purpose' field provides the Authorization Server with an identity token-defined identifier for the intended usage of the requested access token. This parameter can assist the Authorization Server to choose an appropriate audience based on its knowledge of the identity token's needs for the stated purpose. This field is mutually exclusive with the 'audience' and 'scope' field.

16.X.1.1.1.6 audience

The optional 'audience' field provides a list of one or more intended target(s) of an access token. This field is mutually exclusive with the 'purpose' field.

16.X.1.1.1.7 scope

The optional 'scope' field provides the requested scope (permissions) for the token. The server can return a different scope based on its authorization policies. If the scope returned in the token is different from the requested scope, the 'scope' field will be present in the Response Parameters. This field is mutually exclusive with the 'purpose' field.

16.X.1.1.1.8 req-cnf

The optional 'req-cnf' field provides the requested confirmation (binding) for the token. If absent for access token requests, the token will be bound to the 'client-id' using the 'authorized-party' member of the token's 'confirmation' claim.

16.X.1.1.1.9 subject

The optional 'subject' field provides the information about the actor that is making the request. The format and meaning are the same as the 'sub' claim in a BACnetWebToken. The Authorization Server uses this as input to its decision about what to include in the 'sub' claim in the token. This shall be absent for identity token requests.

16.X.1.1.2 Examples

This is a request for an access token for a stated purpose with unknown audience and scope:

```
request-type: "token"
client-id:    3333
purpose:      "brew beer"
```

This is a request made to access Resource Server 4444 with the stated scope for user-id 42.

```
request-type: "token"
client-id:    3333
audience:     [{device:4444}]
scope:        "control"
subject:      "42 0"
```

This is a request made by "helper" device 1111 on behalf of identity token 3333 to adjust and control the "lighting" application on Resource Server 4444.

request-type: "token"
client-id: 1111
audience: [{device: 4444, application: "lighting"}]
scope: "adjust control"
req-cnf: { authorized-party: 3333 }

This is a request made by identity token 3333 to retrieve an identity token for itself. Note that the requested scope here is "id router", since the device knows that it is configured to be a router, and the requested audience is absent, indicating the "everyone" group.

request-type: "id_token"
client-id: 3333
scope: "id router"

16.X.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters:

16.X.1.2.1 Response Values

This parameter, of type BACnetAuthResponse, provides the resultant values, as defined in the Service Procedure below. The members of this structure correspond to the OAuth response parameters defined in RFC 6749 Section 5.1, plus extensions.

The fields of the BACnetAuthResponse are defined by the following clauses.

16.X.1.2.1.1 token-type

The optional 'token-type' field provides the type of the returned token. The value is one of the IANA registered values for "OAuth Access Token Types". If absent, the default value is "PoP" and follows the definition in RFC 8747 and RFC 7800. Support for the default value is required. Bearer tokens are not allowed.

16.X.1.2.1.2 scope

The optional 'scope' field provides the resultant scope in the token. It shall be present if and only if the 'scope' in the token is different from the 'scope' in the Request Parameters,

16.X.1.2.1.3 access-token

The optional 'access-token' field provides the resultant access token value.

16.X.1.2.1.4 id-token

The optional 'access-token' field provides the resultant identity token value.

16.X.1.2.1.5 expires-in

The optional 'expires-in' field provides the number of seconds that remain before token expiration time.

16.X.1.2.1.6 no-cache

The optional 'no-cache' field indicates that the identity token should not try to cache the token in a Resource Server. If absent, it defaults to false.

16.X.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

16.X.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. For security related errors, the 'Error Class' shall be SECURITY and the 'Error Code' shall be appropriate to the conditions described in Clause 18.5.

16.X.1.3.2 Error Details

This optional parameter, of datatype BACnetAuthRequestError, contains additional information about why the service failed. The members of this structure correspond to the OAuth error responses defined in RFC 6749.

The fields of the BACnetAuthRequestError are defined by the following clauses.

16.X.1.3.2 error

The 'error' field corresponds to the "error" result defined in RFC 6749 and extended by other relevant RFCs.

16.X.1.3.2 error-description

The optional 'error-description' field corresponds to the "error_description" result defined in RFC 6749. It is a human readable description of the error.

16.X.1.3.2 error-uri

The optional 'error-uri' field corresponds to the "error_uri" result defined in RFC 6749. It is a URI identifying a web page with information about the error.

16.X.2 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall process the request according to the procedures in the following clauses. If the request can be granted, a 'Result(+)' primitive shall be generated. If the request fails, a 'Result(-)' primitive shall be generated, indicating the reason for the failure.

If the 'endpoint' value equals "token", then the server shall follow the procedure in Clause 16.X.2.1. Support for other 'endpoint' values is a local matter. If the value of the 'endpoint' cannot be processed by the server, a 'Result(-)' primitive shall be generated.

16.X.2.1 Token Request

If the server functions as an Authorization Server and the 'response-type' is equal to "token", then it shall process the request according to Clause 16.X.2.1.1. If the server functions as an Identity Server and the 'response-type' is equal to "id_token", then it shall process the request according to Clause 16.X.2.1.2. Support for other 'response-type' values is a local matter. If the value of the 'response-type' cannot be processed by the server, a 'Result(-)' primitive shall be generated.

If the device instance of the requestor is available (e.g., via a "Secure Source" data attribute), then the server shall verify that the 'client-id' field is equal to the device instance of the requestor.

All token responses shall have the following token claims:

a) 'issuer' is set to the device instance of this server.
b) 'audience' is set to the intended audience. Identity tokens will use group 1 ("everyone")
c) 'scope' is set. Can be empty string for access tokens and is at least "id" for identity tokens.
d) 'subject' is optional for access tokens and absent for identity tokens.
e) 'confirmation' is set appropriately for identity tokens or access tokens.
f) 'expiration' is set to the expiration time of the token.
g) 'authorized-party' is set if and only if the token is an identity token.

If a token is granted, the server shall generate a Result(+) with the appropriate values in the Response Parameters.

16.X.2.1 Identity Token Requests

The Identity Server can return an expanded scope, e.g., "id router", "id hub", or "id hub router", based on its knowledge of the device whose identity is being asserted. The requestor can optionally include additional scopes in its request, e.g., "id router". However, since the inclusion of "router" or "hub" is a critical part of the security of the network, the Identity Server shall never return a scope other than the singular "id" unless specifically configured to do so. Therefore, the inclusion of extra scope by the requestor is only for information purposes and can be used by the Identity Server to alert site administrators of a configuration mismatch. An identity token shall include, at a minimum, the "id" scope.

The 'authorized-party' field in the token shall be set to the device instance of that device whose identity is being asserted.

The 'key-id' field of the 'confirmation' field in the token shall be set to be the 'subject' Distinguished Name in the X.509 certificate of that device.

The requestor can request a token on-behalf-of another device by placing the device instance of that device in the 'authorized-party' field of the 'req-cnf', in the same manner as it would for requesting an access token for another device. In this case, the 'req-cnf' does not automatically become the 'conformation' in the token because the token's 'confirmation' for identity tokens uses the 'key-id' field to bind the token to the 'subject' in a certificate.

16.X.2.1 Access Token Requests

There are two kinds of requests that can be made for access tokens. One is based on the client's knowledge of known audience(s) for the token, and the other is based on a known "purpose" without the client's knowledge of a particular audience. For this reason, the 'audience' and 'purpose' parameters are mutually exclusive.

In the first case, the requestor provides one or more entries in the 'audience' parameter of the request and a requested 'scope'. The server shall consult its policies to determine whether to issue an access token for that/those audience(s) for the identity token and what scope values should be allowed.

In the second case, the requestor provides a string in the 'purpose' parameter. If this string is known to the server for the identity token device, it can issue a token containing appropriate audience(s) and scope based on its knowledge of the identity token device needs for that purpose. The server shall only issue an access token for the 'purpose' option if it has been configured to recognize the 'purpose' for that identity token. The 'purpose' string might not be configurable in the identity token and therefore the server must have the ability to match a purpose string value to a particular client.

If the requestor provides neither an 'audience' nor a 'purpose' parameter, this shall be interpreted by the server to mean a "default purpose", for simple devices that only need one access token for all their configured actions. The sever shall treat this as a 'purpose' request and shall only issue an access token if specifically configured to understand the "default purpose" of the client.

If a 'scope' parameter is provided in the request, it serves only as a requested scope; the server can return a larger, smaller, or completely different scope in the token, based on its configured policies. Requesting more than is allowed shall not result in an error response. If a scope component is requested but not permitted, then that component is simply left out of the resulting scope string. An empty scope string is considered a valid response and shall not result in an error response.

If the scope in the token does not match the requested scope, then the server shall indicate the changed scope in the 'scope' field of the response.

The 'authorized-party' field of the 'confirmation' field in the token shall be set to the device instance of the device that is allowed to use the token.

135-2020*cg*-4 Example Authentication and Authorization Message Flows

Rationale

The understanding of some message exchanges and data structures is aided by informative examples.

[Add new **Annex ZZ**, p. 1403]

ANNEX ZZ - Examples of Device Authentication and Authorization (INFORMATIVE)

(This annex is not part of the standard but is included for informative purposes only.)

The following examples use BACnet Secure Connect, Annex AB, as an example implementation of the authentication and authorization mechanisms defined in Annex XX.

ZZ.1 Establishing Identity

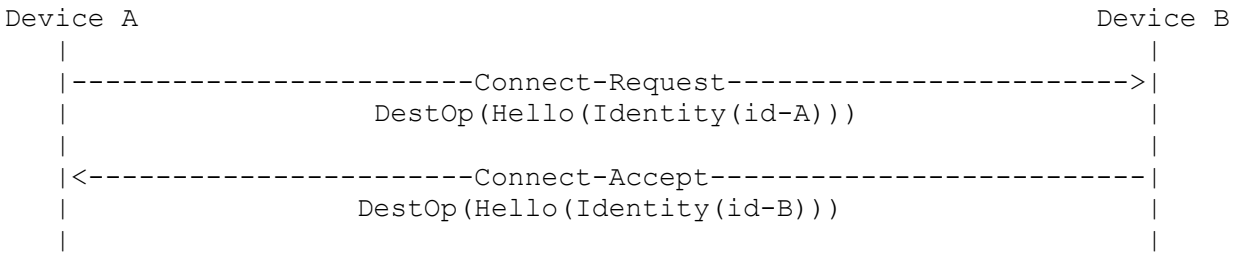
As part of the connection process, the initiating peer can send an identity proof to the accepting peer. The accepting peer will respond with its own identity proof or a negative BVLC-Result message if the initiator's identity claim was not accepted.

Identity is conveyed in both directions in a data option attached to the Connect-Request and Connect-Accept messages. At the end of the connection establishment, each side will mark the other side internally as either "authenticated" or "unauthenticated".

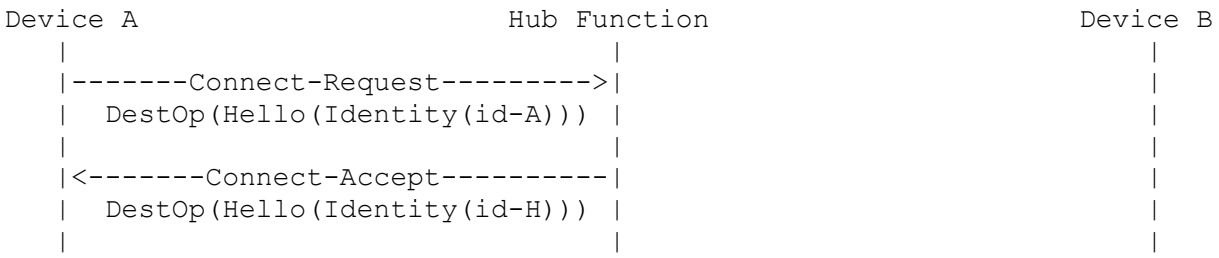
Either of the identity claims can be empty if an identity token is not yet available, but the presence of the 'Hello' option in the connection establishment messages lets the receiving peer know that the sending peer is at an implementation level that "understands" identity.

If the initiating peer sends a 'Hello' option with the Connect-Request message, then the accepting peer will send a 'Hello' option with the Connect-Accept message. Conversely, if the initiating peer does not send a 'Hello' option with the Connect-Request message, then the accepting peer will assume that it is an older device and will not send a 'Hello' option with its Connect-Accept message.

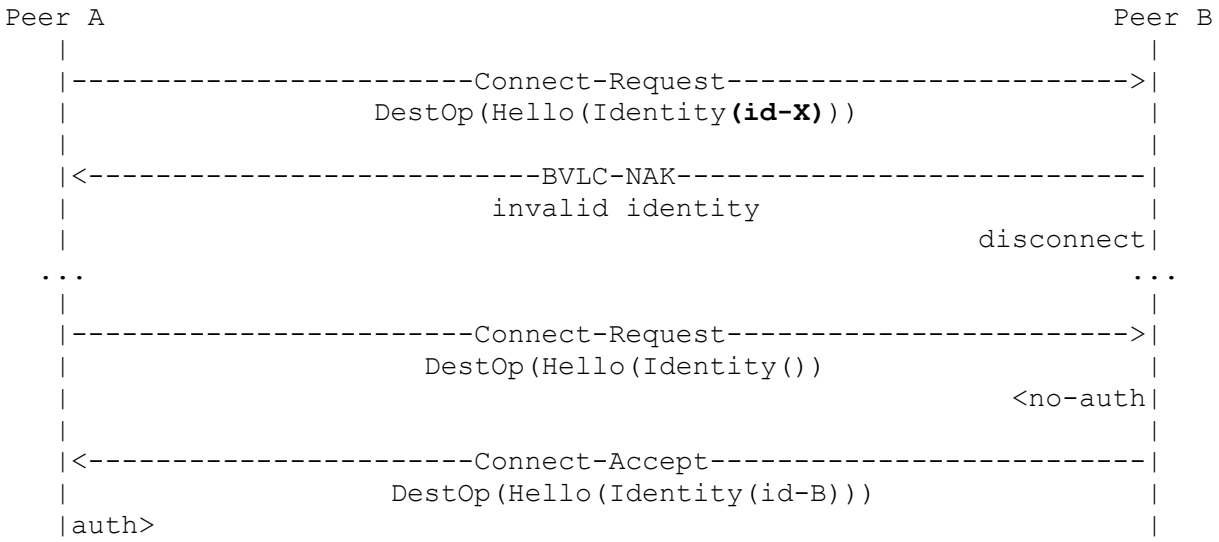
For the direct connect case, the initiating peer will send its identity to the accepting peer and the accepting peer will return its own identity.



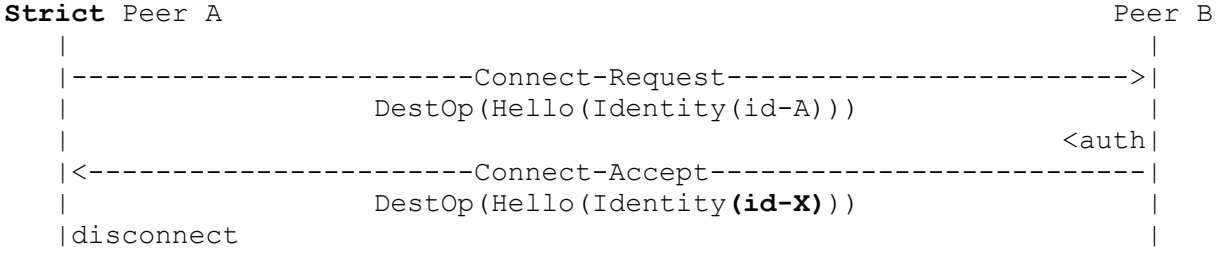
Hub Functions will respond with the identity of the device that contains the Hub Function.

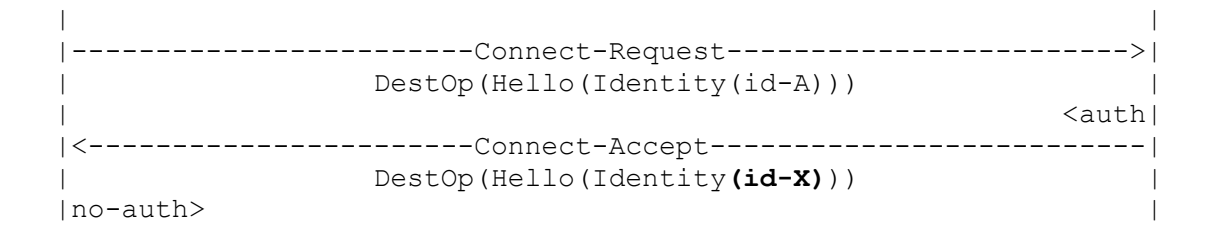


If the initiating peer's identity is not valid, the accepting peer will send a NAK and disconnect. The initiating peer may choose to try again without an identity token. This may be desirable in cases where the initiating peer's identity is not intentionally evil, just misconfigured. By allowing it to connect in an "unauthenticated" way, it allows it to communicate to someone that it is in need of service.



In the case where the accepting identity is invalid, the initiating peer may disconnect or continue and mark the accepting peer as unauthenticated, depending on its own settings.





Identity tokens are only exchanged with Connect-Request and Connect-Accept. Once a connection is established, if an identity changes, the connection will need to be disconnected and reestablished.

ZZ.2 Example Identity Token

```
header:
   key-id:           "3E21"      (refers to a key in a BACnetSigner)
claims:
   audience:         [{group:1}] (constant, the "everyone" group)
   authorized-party: 240105      (the possessor's device instance)
   confirmation:     {key-id: "O=Controls-R-Us, CN=GreatDevice, ..."}
   scope:            "id router"
   issuer:           249998     (the Identity Server's DeviceID)
   issued-at:        1426420800
   expiration:       1627537240
signature: DF6E76DFAE7AD77E39DFAEF5DF8EB7DB
           8EB6DF8E7ADF8E7AE78DF9DFAE7CDF6E
           FAE7CDB7EFAE39F3BDB7EB8E7CDF5EF8
           EB9F37D7BE3AE7CDFBE3AE7CDF8EF9EB
```

ZZ.3 Response Authentication

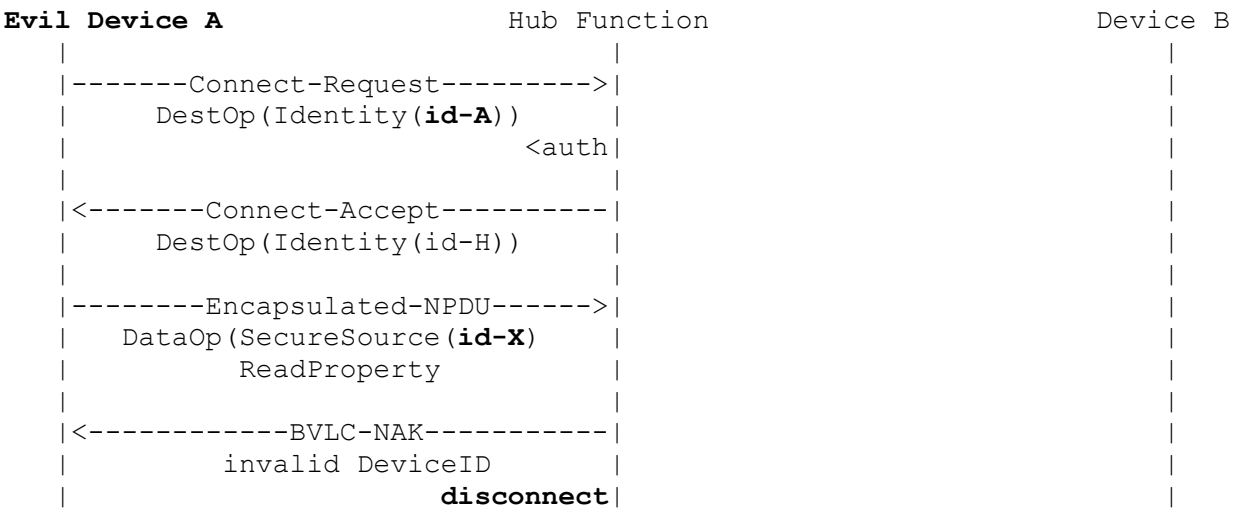
Responses must be authenticated as well. This way, a requester can have assurance that the answers are not spoofed. If a "Secure Source" data option was present on a request, the responding device must place its own "Secure Source" data option on the response.

```
Device A                   (hubs/routers not shown)                Device B
   |                                                                   |
   |-------------------------Encapsulated-NPDU------------------------>|
   |                      DataOp(SecureSource(id-A))                   |
   |                            ReadProperty                           |
   |                                                                   |
   |<------------------------Encapsulated-NPDU------------------------|
   |                      DataOp(SecureSource(id-B))                   |
   |                           ReadProperty-ACK                        |
```

ZZ.4 Validating Authentication

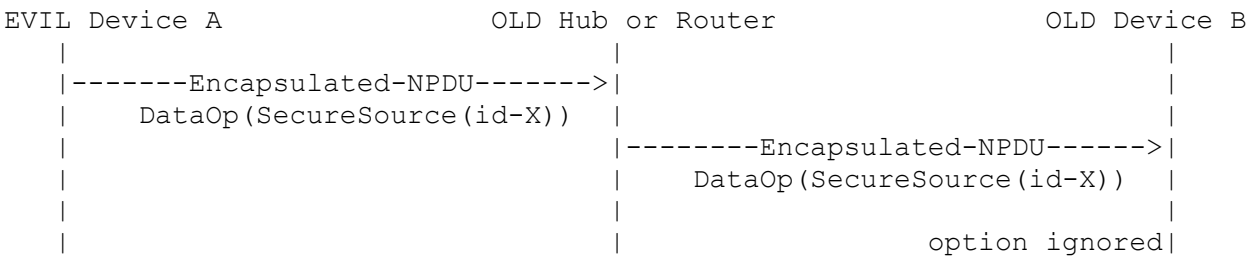
Since only a Direct Connect peer, Hub Function, or Hub Connector has a direct TLS connection to the originating peer, only they can validate the identity claim against the originator's certificate. Therefore, they are responsible for verifying that any claimed identity in incoming messages is correct before propagating the message.

If a 'Secure Source' option from an "authenticated" non-routing node is asserted incorrectly, the receiving peer will send a NAK and disconnect since there is no valid reason for this error other than bad intent.

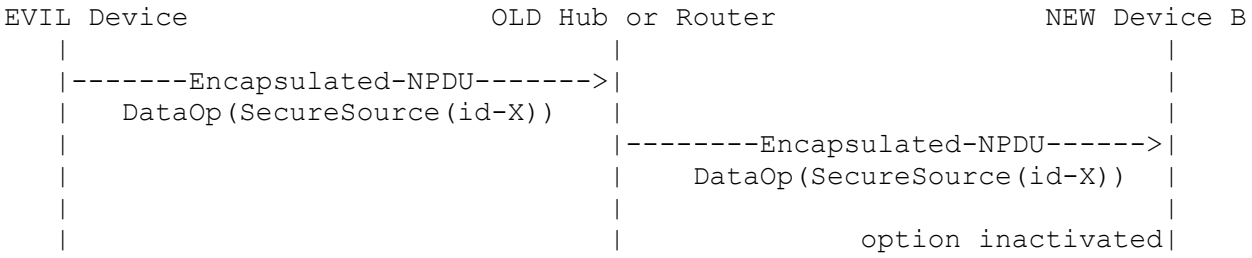


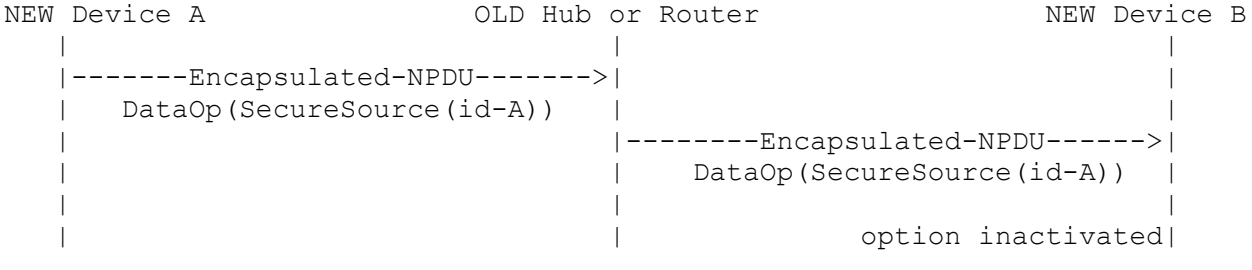
ZZ.5 Legacy Interactions

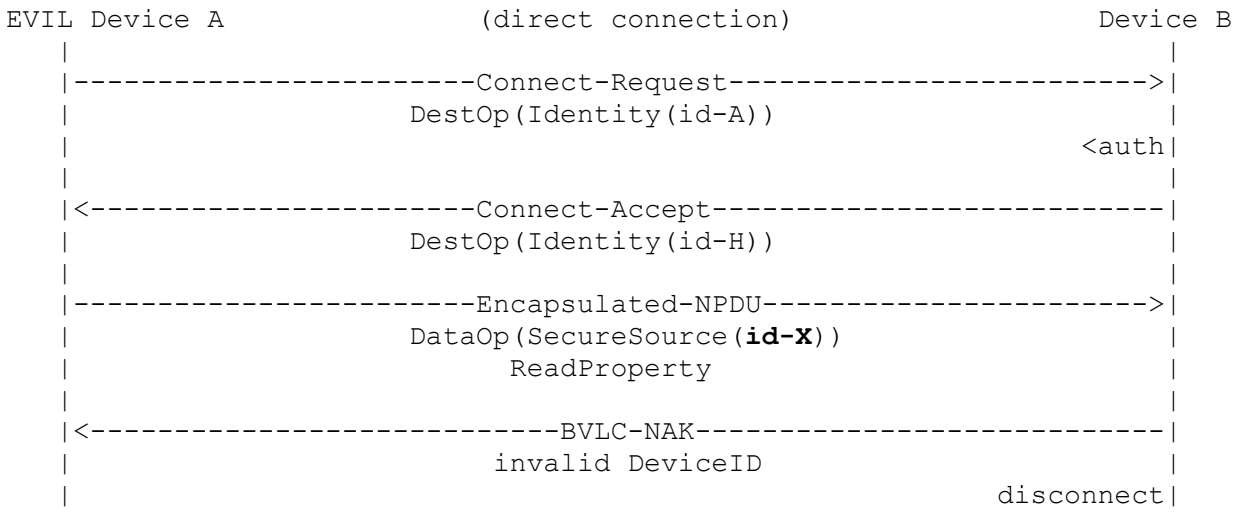
In the following, the evil device manages to forge an identity and get it all the way through to an old device. But nothing happens because the old device doesn't understand authentication and therefore grants no special privileges to the NPDU.



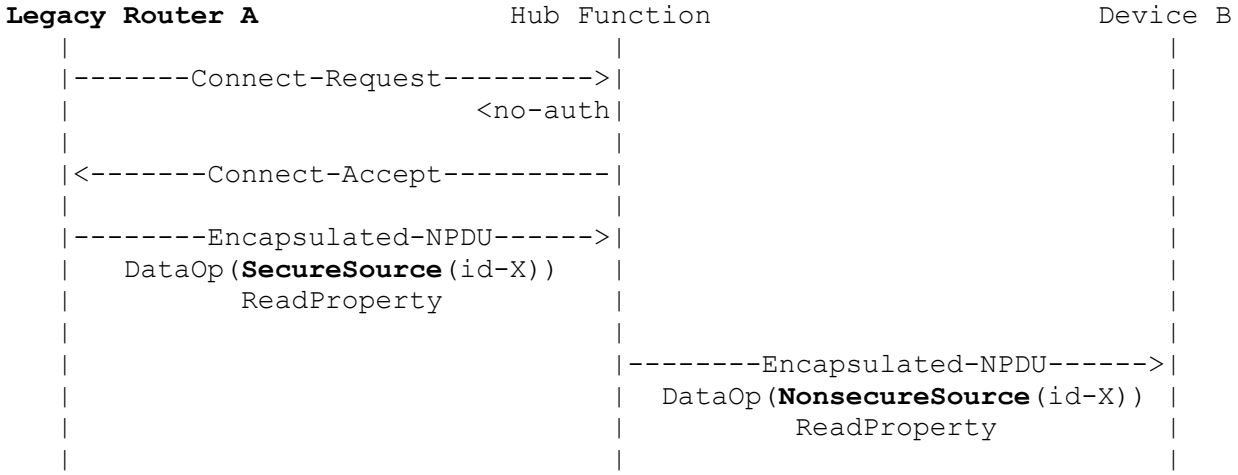
In the following two examples, either an evil device manages to forge an identity and get it all the way through to a new device, or a new and truthful device is simply behind an old hub or router. In either case, the new recipient cannot trust the identity since it came from an unauthenticated peer. So, the new recipient will inactivate the authentication information but should still accept the NPDU because there is no way to tell if the originator was truly evil or simply behind a legacy hub or router.

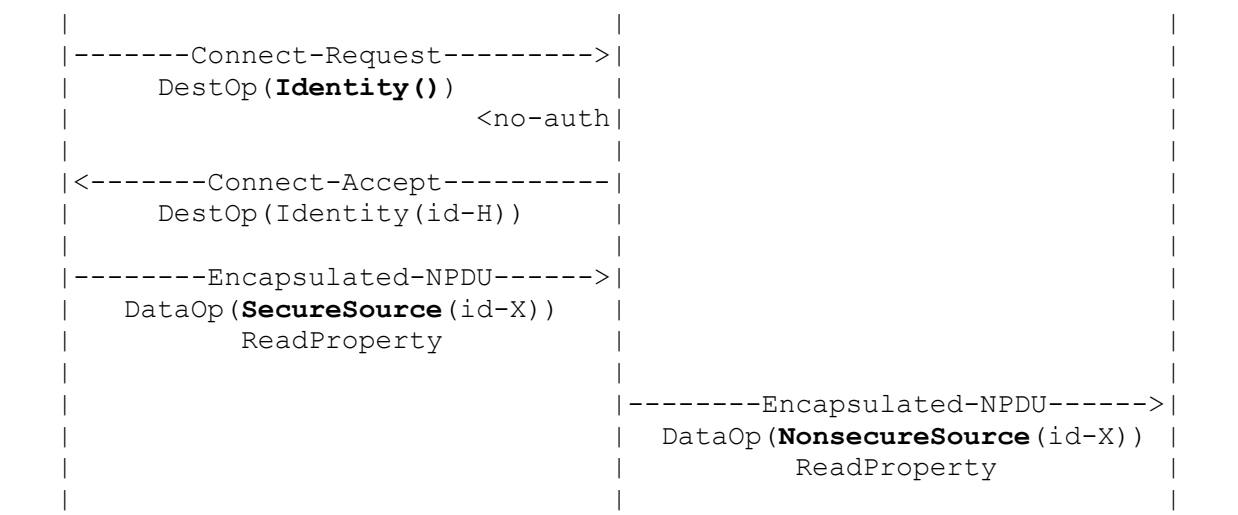




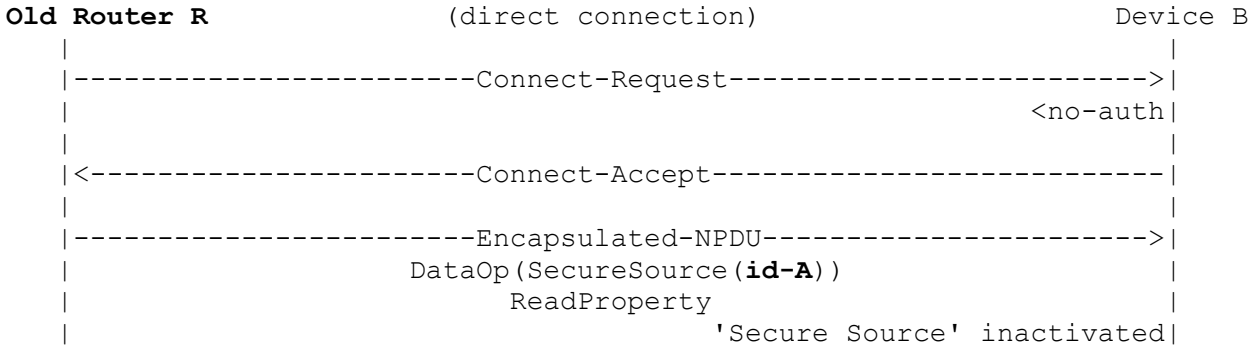


If a 'Secure Source' option is received from an "unauthenticated" node (either a legacy router or a new device without an identity token), then the receiving peer will change the option into a 'Nonsecure Source' option.



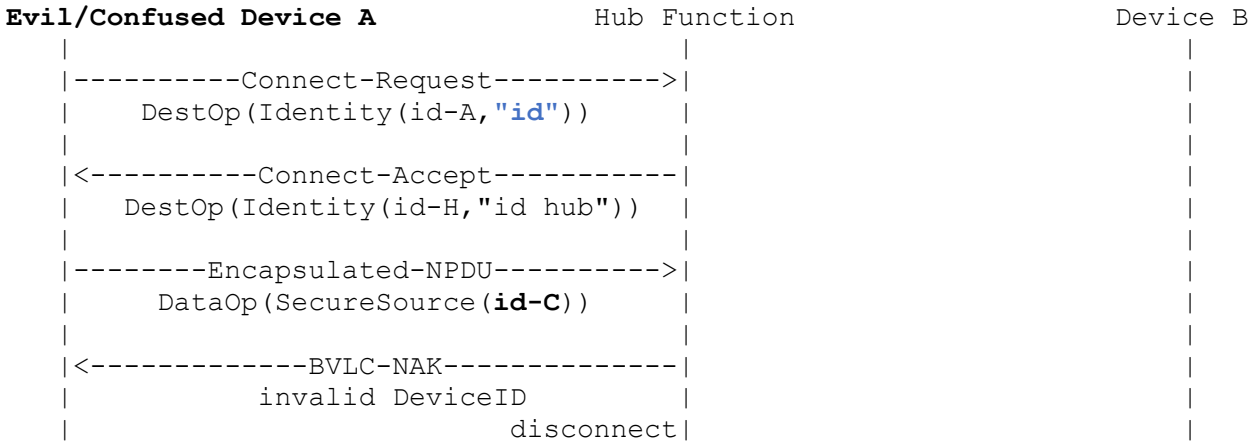


In the direct connect case, if a 'Secure Source' option is received from an "unauthenticated" node (either a legacy router or a new device without an identity token), then the receiving peer will internally remove the option, or change it into a Nonsecure Source' option, before passing it up the stack.

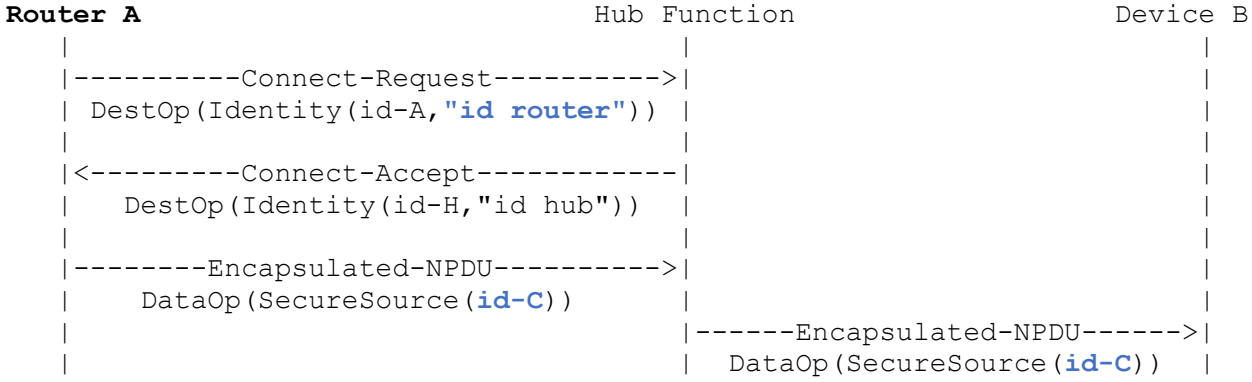


ZZ.6 Routing Authentication

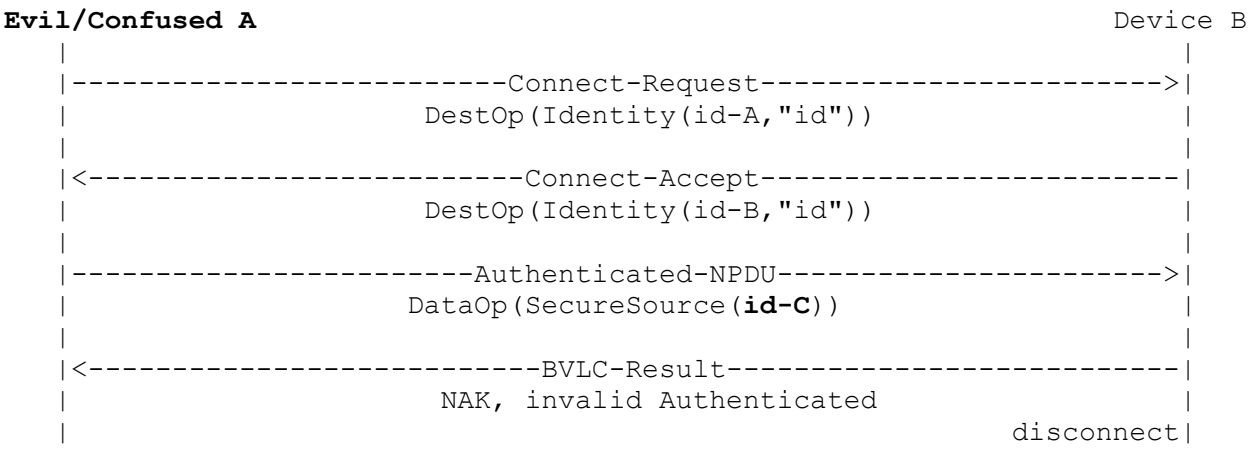
To enable propagating an authentication indication by a device with a different identity (i.e., a router), "router" is added to the scope claim of the router in its identity token. If the "router" scope is not present, then an attempt to send a 'Secure Source' option that does not match the validated identity of the sending peer will be considered a fraudulent or misconfigured node.



However, if the "router" scope is asserted, then the Hub Function will allow it.

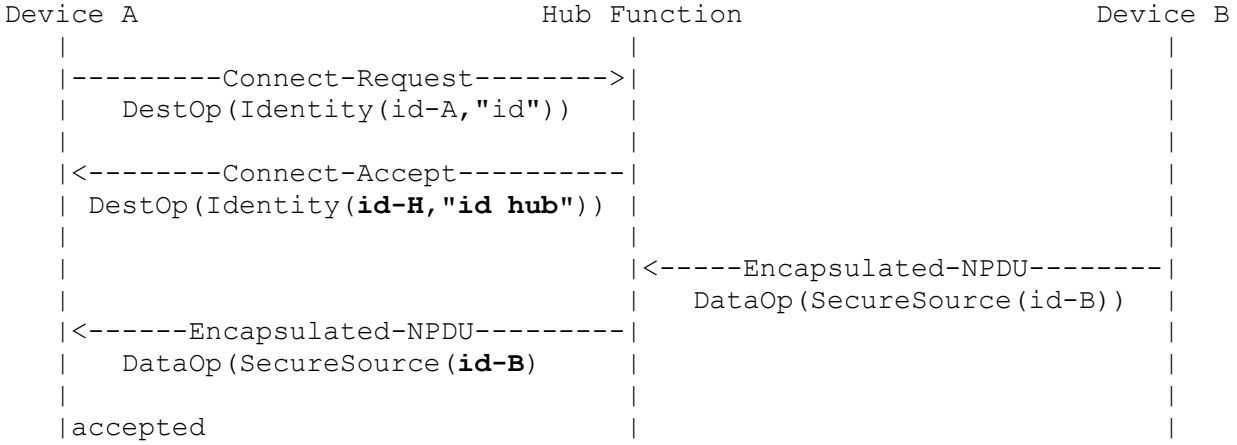


Likewise, in the Direct Connect case, if the sender does not identify with the "router" permission, then the receiving side will reject it.



ZZ.7 Forwarding Authentication

Like a router, a Hub Function needs to propagate 'Secure Source' options that do not match the device instance of the device containing the Hub Function. To enable this, another permission named "hub" is added to the scope claim of the hub.



When the Hub Connector is connecting to a hub and the "hub" scope is not present in the hub's identity token, then the Hub Connector will terminate the connection, assuming that it has connected to a misconfigured hub by mistake.


```
Device A                          Hub Function                          Device B
   |                                   |                                   |
   |---------Connect-Request-------->|                                   |
   |   DestOp(Identity(id-A,"id"))   |                                   |
   |                                   |                                   |
   |<--------Connect-Accept----------|                                   |
   |   DestOp(Identity(id-H,"id"))   |                                   |
   |                                   |                                   |
   |disconnect                         |                                   |
```

When the Hub Connector is connecting to a hub and the hub does not present an identity token, then the initiating device can decide whether to proceed with the hub being "unauthenticated" or not based on its own settings.

```
Device A                          Hub Function                          Device B
   |                                   |                                   |
   |---------Connect-Request-------->|                                   |
   |   DestOp(Identity(id-A,"id"))   |                                   |
   |                                   |                                   |
   |<--------Connect-Accept----------|                                   |
   |       DestOp(Identity())          |                                   |
   |                                   |                                   |
   |disconnect, or                     |                                   |
   |no-auth>                           |                                   |
```

ZZ.8 Example Access Token

Example Access Token

```
header:
   key-id:       "C54A"               (identifies the BACnetSigner)
claims:
   subject:      "32 2"              (user 32, role 2)
   confirmation: {authorized-party: 240105}  (the possessor's device instance)
   audience:     [{device: 240202}]  (the target device instance)
   issuer:       459999  (the authorization server's device instance)
   issued-at:    1426420900
   expiration:   1627538350
   scope:        "adjust config"
signature: 8EB6DF8E7ADF8E7AE78DF9DFAE7CDF6E
           DF6E76DFAE7AD77E39DFAEF5DF8EB7DB
           EB9F37D7BE3AE7CDFBE3AE7CDF8EF9EB
           FAE7CDB7EFAE39F3BDB7EB8E7CDF5EF8
```

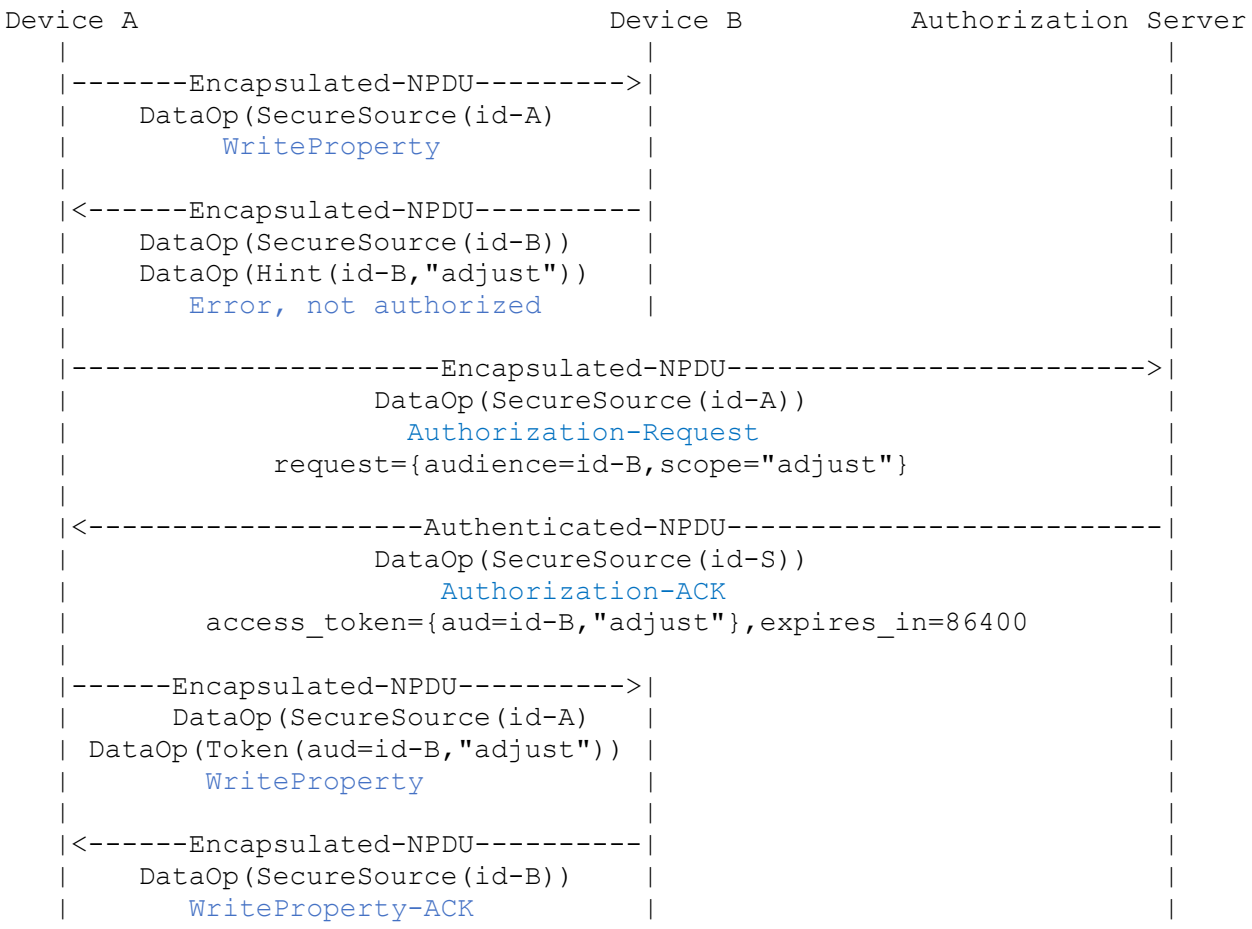
ZZ.9 BACnetWebKey

Example BACnetWebKey containing a public key of default type "EC" with the default "P-256" curve:

```
key-id: "C65F"
x:      X'30a0424cd21c2944838a2d75c92b37e76ea20d9f00893a3b4eee8a3c0aafec3e'
y:      X'e04b65e92456d9888b52b379bdfbd51ee869ef1f0fc65b6659695b6cce081723'
```


ZZ.10 Acquiring an Access Token Based on Hint

In this BACnet/SC example, the client requests a protected operation without proper authorization. The resource server returns a 'Hint' data option containing the information the client needs to obtain an access token



135-2020*cg*-5 Auth Object

Rationale

The authentication and authorization mechanisms require network visible and configurable properties. There is only one set of these properties per device, so they could have been placed in the Device object; however, like the Network Port, there are interdependencies between the properties, and misconfiguration of the properties can cause a serious problem. Therefore, the "Command" and "Command_Validation_Result" mechanisms from Addendum cc are used here in a new stand-alone object. Like the Device object, there is only one Auth object per device.

12.X Auth Object Type

The Auth object provides access to the configuration and properties related to the authentication and authorization of the device. There shall be exactly one Auth object in each BACnet device that supports the authentication and authorization mechanisms defined in Annex XX. The instance number of the object shall be 1. Since there can only be one Auth object, fixing its instance number allows it to be read without needing to discover it in the Object_List.

Property values which are required to maintain proper operation of authentication and authorization shall be retained across a device reset.

Auth objects may optionally support intrinsic reporting to facilitate the reporting of fault conditions. Auth objects that support intrinsic reporting shall apply the NONE event algorithm.

The object and its properties are summarized in Table 12-X1 and described in detail in this clause.

Table 12-X1. Properties of the Auth Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Reliability	BACnetReliability	R
Identity_Token	BACnetWebToken	R
Identity_Server	BACnetSigner	R
Authorization_Server	BACnetSigner	R
Authorization_Server_Alt	BACnetSigner	R
Allow_Factory_Authorization	BOOLEAN	O[4]
Device_Groups	BACnetARRAY of Unsigned	R
Changes_Pending	BOOLEAN	R
Command	BACnetNetworkPortCommand	O[1]
Current_Health	BACnetHealth	O
Command_Validation_Result	BACnetHealth	O
Event_Detection_Enable	BOOLEAN	O[2,3]
Notification_Class	Unsigned	O[2,3]
Event_Enable	BACnetEventTransitionBits	O[2,3]
Acked_Transitions	BACnetEventTransitionBits	O[2,3]
Notify_Type	BACnetNotifyType	O[2,3]
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O[2,3]
Event_Message_Texts	BACnetARRAY[3] of CharacterString	O[3]
Event_Message_Texts_Config	BACnetARRAY[3] of CharacterString	O[3]
Event_State	BACnetEventState	O[2]

Reliability_Evaluation_Inhibit	BOOLEAN	O
Property_List	BACnetARRAY[N] of BACnetPropertyIdentifier	R
Tags	BACnetARRAY[N] of BACnetNameValue	O
Profile_Location	CharacterString	O
Profile_Name	CharacterString	O

[1] Shall be present if, and only if, the object supports execution of any of the values of the Command property. If present, this property shall be writable.
[2] These properties are required if the object supports intrinsic reporting.
[3] These properties shall be present only if the object supports intrinsic reporting.
[4] Required to be present and configurable if the device is capable of accepting access tokens from a "factory authorization server"

12.X.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. For the Auth object, the object instance number shall be 1.

12.X.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.X.3 Object_Type

This read-only property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be AUTH.

12.X.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.X.5 Status_Flags

This read-only property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of the object. The four flags are:

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical TRUE (1) if the Event_State property is present and does not have a value of NORMAL, otherwise logical FALSE (0).
FAULT	Logical TRUE (1) if the Reliability property does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	The value of this flag shall be logical FALSE (0).
OUT_OF_SERVICE	The value of this flag shall be logical FALSE (0).

12.X.6 Reliability

This property, of type BACnetReliability, provides an indication of whether the Auth object is "reliable" as far as the BACnet device can determine and, if not, why.

If the 'Result' field of the Current_Health property does not have an error code of SUCCESS, then the Reliability property shall have a value that is appropriate to the error, such as CONFIGURATION_ERROR.

12.X.7 Identity_Token

This property, of type BACnetWebToken, contains the identity token for the device. See Clause XX.2.7~~XX.1.7~~. If the device has not been configured with an identity token yet, attempts to read this property shall result in the return of a Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_NOT_INITIALIZED. The use of 4194303 in the 'issuer' field shall indicate that no on-line authentication server is used on this site.

If the device instance in the identity_token does not match the Object_Identifier property of the Device object, it is a local matter whether the device adopts the device instance of the identity token or continues with the mismatch. If the two remain mismatched, the Auth object shall indicate this error in its Reliability and Current_Health properties.

12.X.8 Identity_Server

This property, of type BACnetSigner, contains device instance number and the signing keys of the authentication server. The use of 4194303 shall indicate that no on-line authentication server is used on this site. The authentication server's keys are used to validate the identity tokens of other devices. See Clause XX.2.8~~XX.1.8~~. The device shall attempt to validate the signature of its own Identity_Token property with the keys in this property when a VALIDATE_CHANGES command is received.

12.X.9 Authorization_Server

This property, of type BACnetSigner, contains the authorization server that can issue access tokens for this device. These keys contained in this property are used to validate access tokens provided to this device by other devices. See Clause XX.3.5~~XX.2.5~~. If this property is not configured, the instance number of the BACnetSigner shall be 4194303.

12.X.10 Authorization_Server_Alt

This property, of type BACnetSigner, contains an alternate authorization server that can issue access tokens for this device. These keys contained in this property are used to validate access tokens provided to this device by other devices. See Clause XX.3.5~~XX.2.5~~. If this property is not configured, the instance number of the BACnetSigner shall be 4194303. The use of an alternate is optional. If an alternate is used, then both the authorization servers shall have identical policies for issuing access tokens.

12.X.11 Allow_Factory_Authorization

This property, of type BOOLEAN, indicates whether the device is enabled to accept access tokens from a proprietary "Factory Authorization Server" for "factory service" scenarios. The configuration of the signing keys for such a server is outside the scope of this standard; therefore, if the device has a possibility of accepting such tokens, then this property is required to be present and configurable so the owner can choose to enable and disable the behavior.

12.X.12 Device_Groups

This property, of type BACnetARRAY of Unsigned, contains the collection of authorization group numbers that this device is a member of. See Clause XX.3.9~~XX.2.9~~. This array shall be configurable and shall support a minimum of 5 entries. The group number 1 is reserved to mean "everyone" and shall not appear in this array. The number 0 is not assignable as a group number and is used in this array to indicate an unused array entry.

12.X.13 Changes_Pending

This property, of type BOOLEAN, indicates whether the configuration settings in the Auth object map to the current configuration settings. A value of FALSE indicates that the configuration settings reflect the current configuration information. A value of TRUE indicates the configuration settings have been modified but have not been activated on the object.

When a value is written to a property that requires activation, the value of the Changes_Pending property shall automatically be set to TRUE, indicating that the current property values are not the values actively in use.

It is necessary for the client to initiate a ReinitializeDevice service request with a 'Reinitialized State of Device' of ACTIVATE_CHANGES or WARMSTART in order to activate the currently visible configuration settings. This interaction is necessary to support atomic updating of multiple properties when modifying an Auth object configuration.

It is a local matter as to whether or not resetting the device by means other than a ReinitializeDevice service with a 'Reinitialized State of Device' of ACTIVATE_CHANGES or WARMSTART discards pending changes, activates pending changes, or leaves the changes pending.

12.X.14 Command

This property, of type BACnetAuthCommand, is used to request that the Auth object perform various actions.

When this property is written, the sequence of operations shall be as follows:

1) Perform any necessary validation. If Result(-) is returned, this property shall retain the value that it had before the write was attempted and no change shall be made to any other property of the object.
2) If validation succeeds, this property shall be set to the value written and a Result(+) be returned.
3) The device shall begin performing the requested command actions.
4) When the object is able to accept another command, the Command property shall be set to IDLE. This may occur immediately, when the actions have completed, or when the actions have proceeded to a point that allows the implementation to accept another command. The exact timing is a local matter.

When this property has a value other than IDLE, any attempt to write to it shall result in the return of a Result(-) with an 'Error Class' of OBJECT and an 'Error Code' of BUSY.

Writing a value of IDLE to this property shall result in the return of a Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE.

If the value of the Changes_Pending property is TRUE, writing a supported command other than DISCARD_CHANGES or VALIDATE_CHANGES to the Command property shall result in the return of a Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of INVALID_VALUE_IN_THIS_STATE.

Any of the following commands may be written to this property:

DISCARD_CHANGES	If the object supports this command, this object shall revert to the set of property values that were contained in this object when Changes_Pending was last equal to FALSE. Changes_Pending shall be set to FALSE, and Command shall be set to IDLE. If the object does not support this command, writing this value shall result in the return of a Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED. Changes_Pending shall remain TRUE, and Command shall be set to IDLE.
VALIDATE_CHANGES	If this object supports this command, this command shall initiate a validation of the values of the properties of this object as specified in each property. If a property is present but not used it shall not be validated. The value of the Command_Validation_Result property shall be updated to indicate the validation result. The Command shall remain unchanged until the validation has been completed at which point the Command shall be set to IDLE. This Command shall not affect the Changes_Pending property. If the object supports pending changes, then the object shall support this command. If the object does not support this command, writing this command shall return Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to allow command values other than those defined by the standard. For proprietary extensions of this enumeration, see Clause 23.1 of this standard.

A proprietary command failure shall result in the value of the Reliability property being set to PROPRIETARY_COMMAND_FAILURE and the value of this property being set to IDLE.

It is a local matter whether the value of this property remains at the proprietary value until the proprietary action has completed (whether in success or failure), and then returns to IDLE; or whether the property returns to IDLE once the action has been initiated and the object is prepared to accept another command.

This enumerated value is extensible. Writing an unknown value to this property shall result in the return of a Result(-) with an 'Error Class' of PROPERTY and an 'Error Code' of VALUE_OUT_OF_RANGE.

12.X.15 Current_Health

This read-only property, of type BACnetHealth, indicates the current health of this object. It is a local matter when and how often this object is evaluated. When multiple error conditions exist, it is a local matter which is reported in this property.

This property shall not report errors caused by pending changes.

The BACnetHealth structure has the following fields:

Timestamp	This field, of type BACnetDateTime, indicates the local date and time when this property was last updated. If the device has no knowledge of local time or date, this field shall contain an unspecified datetime.
Result	This field, of type Error, indicates the most recent error for this port. If the object is not currently in error, this field shall contain 'Error Class' of OBJECT and 'Error Code' of SUCCESS.
Property	This optional field, of type BACnetPropertyIdentifier, indicates the property that caused the error. If this object is not currently in error, this field shall be absent.
Details	This optional field, of type CharacterString, provides details about the most recent error. If the object is not currently in error, this field shall be absent.

12.X.16 Command_Validation_Result

This read-only property, of type BACnetHealth, indicates any errors detected as a result of writing VALIDATE_CHANGES to the Command property. This property shall be present if the VALIDATE_CHANGES command is supported. See Clause 12.X.14. The BACnetHealth structure is described in Clause 12.X.15.

12.X.17 Notification_Class

This property, of type Unsigned, shall specify the instance of the Notification Class object to use for event-notification-distribution.

12.X.18 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable the distribution of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL notifications (see Clause 13.2.5). A device is allowed to restrict the set of supported values for this property but shall support (T, T, T) at a minimum.

12.X.19 Acked_Transitions

This read-only property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the acknowledgment state for TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1.5). Each flag shall have the value TRUE if no event of that type has ever occurred for the object.

12.X.20 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. The value of the property is used as the value of the 'Notify Type' service parameter in event notifications generated by the object.

12.X.21 Event_Time_Stamps

This read-only property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events (see Clause 13.2.2.1). Timestamps of type BACnetDateTime shall have X'FF' in each octet, and Sequence Number timestamps shall have the value 0 if no event of that type has ever occurred for the object.

12.X.22 Event_Message_Texts

This read-only property, of type BACnetARRAY[3] of CharacterString, shall convey the message text values of the last TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively (see Clause 13.2.2.1). If a particular type of event has yet to occur, an empty string shall be stored in the respective array element.

12.X.23 Event_Message_Texts_Config

This property, of type BACnetARRAY[3] of CharacterString, contains the character strings which are the basis for the 'Message Text' parameter for the event notifications of TO_OFFNORMAL, TO_FAULT, and TO_NORMAL events, respectively, generated by this object. The character strings may optionally contain proprietary text substitution codes to incorporate dynamic information such as date and time or other information.

12.X.24 Event_State

This read-only property, of type BACnetEventState, is included in order to provide a way to determine whether this object has an active event state associated with it (see Clause 13.2.2.1). If the object supports event reporting, then the Event_State property shall indicate the event state of the object. If the object does not support event reporting, then the value of this property shall be NORMAL.

12.X.25 Reliability_Evaluation_Inhibit

This property, of type BOOLEAN, indicates whether (TRUE) or not (FALSE) reliability-evaluation is disabled in the object. This property is a runtime override that allows temporary disabling of reliability-evaluation.

When reliability-evaluation is disabled, the Reliability property shall have the value NO_FAULT_DETECTED unless Out_Of_Service is TRUE and an alternate value has been written to the Reliability property.

12.X.26 Property_List

This read-only property is a BACnetARRAY of property identifiers, one property identifier for each property that exists within the object. The Object_Name, Object_Type, Object_Identifier, and Property_List properties are not included in the list.

12.X.27 Audit_Level

This property, of type BACnetAuditLevel, specifies the level of auditing to perform for the specific object. If this property has the value DEFAULT, or if this property is not present in the object, then the audit level value for the object shall be taken from the Audit_Level property of the object's associated Audit Reporter object.

For details on auditing and the use of this property, see Clause 19.6.3.

If this property is present and not configurable, the value shall not be NONE.

12.X.28 Auditable_Operations

This property, of type BACnetAuditOperationFlags, specifies the operations that the device will report for this object.

If present and not configurable, the values of the READ, NOTIFICATION, and SUBSCRIPTION bits shall be 0, and the WRITE, CREATE, DELETE, and ACKNOWLEDGE_ALARM bits shall be 1.

If this property is not present, and the device supports auditing, then the Auditable_Operations property of the object's associated Audit Reporter object shall control the operations that are auditable for this object.

For details on auditing and the use of this property, see Clause 19.6.3.

12.X.29 Tags

This property, of type BACnetARRAY of BACnetNameValue, is a collection of tags for the object. See Clause Y.1.4 for restrictions on the string values used for the names of these tags and for a description of tagging and the mechanism by which tags are defined.

Each entry in the array is a BACnetNameValue construct which consists of the tag name and an optional value. If the tag is defined to be a "semantic tag" then it has no value, and the "value" field of the BACnetNameValue shall be absent.

While some tags may be known in advance when a device is manufactured, it is recommended that implementations consider that this kind of information might not be known until a device is deployed and to provide a means of configuration or writability of this property.

12.X.30 Profile_Location

This property, of type CharacterString, is the URI of the location of an xdd file (See Clause X.2) containing the definition of the CSML type specified by the Profile_Name property and possible other information (See Annex X). The URI is restricted to using only the "http", "https", and "bacnet" URI schemes. See Clause Q.8 for the definition of the "bacnet" URI scheme.

If a Profile_Location value is not provided for a particular object, then the client shall use the Profile_Location of the Device object, if provided, to find the definition of the Profile_Name.

12.X.31 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name shall begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the profile name shall indicate the organization that publishes and maintains the profile. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. If the Profile_Location property of this object or the Device object is present and nonempty, then the value of this property shall be the name of a CSML type defined in an xdd file referred to by the Profile_Location property.

135-2020*cg*-6 Changes to Existing Services Support Authentication and Authorization

Rationale

Existing services need new error possibilities added to their situation tables.

[Change **Clauses 13.13.1.3.1, 13,19.1.3.1, 13.20.1.3.1, 14.1.4.1, 14.2.4.1, 15.1.1.3.1, 15.2.1.3.1, 15.3.1.3.1, 15.4.1.3.1, 15.5.1.3.1, 15.7.1.3.1, 15.8.1.3.1, 15.9.1.3.1, 15.10.1.3.1, 16.1.1.3.1, 16.4.1.3.1**, pp. 702-765]

X.X.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. *For security related errors, the 'Error Class' shall be SECURITY and the 'Error Code' shall be appropriate to the conditions described in Clause 18.5. Otherwise, the following situations shall apply.*
...

[Change **Clauses 16.2.1.3.1, 17.2.1.3.1, 17.3.1.3.1, 17.4.1.3.1**, pp. 762-785]

X.X.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) an 'Error Code'. See Clause 18. *For security related errors, the 'Error Class' shall be SECURITY and the 'Error Code' shall be appropriate to the conditions described in Clause 18.5.*

135-2020*cg*-7 BACnet/WS Changes

Rationale

Additions are needed to the definitions of scope in Annex W.

[Add rows to **Table W-1**, p. 1228]

Table W-1. Predefined Scopes

Scope Identifier	Meaning
view	View (view private data - public data does not need authorization)
adjust	Adjust setpoints
control	Runtime Control (write values for the purpose of controlling actions)
override	Command Override (placing objects out of service, commanding at priorities that would limit runtime control, etc.)
config	Configuration and Programming (configuration and programming actions, adding objects etc.)
bind	Configuration of external references for which the server device will use its own credentials to read or write.
install	Installation (more technical configuration actions, adding IO points, uploading different application programs)
auth	Configuration of authorization-related data.
id	*Indicates an identity token*
hub	*The device can function as a hub (allowed to forward 'Secure Source')*
router	*The device can function as a router (allowed to forward 'Secure Source')*
authz	*The device can function as an authorization server*
authn	*The device can function as an identity server*

135-2020*cg*-8 Error and Abort Codes for Authentication and Authorization

Rationale

This addendum section defines new error codes for the Error datatype and Abort reasons for Abort PDUs.

[Change **Clause 18.5**, p. 793]

18.5 Error Class - SECURITY

...

BAD_SIGNATURE - The signature in a secure message was incorrect. ~~(Reserved for future use)~~

...

SOURCE_SECURITY_REQUIRED - The operation requested requires that the source secure or encrypt the request. ~~(Reserved for future use)~~

...

UNKNOWN_AUTHENTICATION_TYPE - The authentication method in a secure message is unknown to the receiving device. ~~(Reserved for future use)~~

...

INCORRECT_AUDIENCE - *The audience specified in a security token does not match the recipient.*

INCORRECT_SUBJECT - *The subject name specified in a security token does not match the recipient.*

INCORRECT_INSTANCE - *The device instance specified in a security token does not match the recipient.*

[Insert new values into **Clause 21** Error production's 'error-code' field, p 863]

```
error-code          ENUMERATED { -- see below for numerical order
                      ...
                      incorrect-audience                (n),
                      incorrect-subject                 (n+1),
                      incorrect-instance                (n+2),
                      ...
                      -- see incorrect-audience         (n),
                      -- see incorrect-subject          (n+1),
                      -- see incorrect-instance         (n+2),
```


135-2020*cg*-9 New ASN.1 types for BACnet Authentication and Authorization

Rationale

This section defines the new data types and other changes to Clause 21 required for the other sections of this addendum.

[Add new entry in **BACnet-Confirmed-Service-Request**, p. 858]

```
BACnet-Confirmed-Service-Request ::= CHOICE {
        ...
-- Security Services
        auth-request                    [n]   AuthRequest-Request,
        ...
-- Services added after 2020
        -- auth-request                 [n]   see Security Services
        }
```

[Add new entry in **BACnet-Confirmed-Service-ACK**, p. 859]

```
BACnet-Confirmed-Service-ACK ::= CHOICE {
        ...
-- Security Services
        auth-request                    [n]   AuthRequest-ACK,
        ...
-- Services added after 2020
        -- auth-request                 [n]   see Security Services
        }
```

[Add new Clause 21.2.X p. 866]

21.2.X Confirmed Security Services

```
AuthRequest-Request ::= SEQUENCE { -- IANA "OAuth Parameters"
        extensions      [0] SEQUENCE OF BACnetNameValue OPTIONAL, -- BACnet
        endpoint        CharacterString,                  -- RFC 6749 URI path
        client-id       Unsigned,                         -- RFC 6749
        response-type   [1] CharacterString OPTIONAL,    -- RFC 6749
        audience        [2] SEQUENCE OF BACnetAudience OPTIONAL,  -- RFC 8693
        purpose         [3] CharacterString OPTIONAL,     -- BACnet
        scope           [4] CharacterString OPTIONAL,     -- RFC 6749
        req-cnf         [5] BACnetConfirmation OPTIONAL,   -- ietf-ace-oauth-params
        subject         [6] CharacterString OPTIONAL      -- BACnet
        -- Only the IANA parameters with defined interoperable uses are included.
        -- Proprietary extensions shall use 'extensions' for other parameters.
        }

AuthRequest-ACK ::= SEQUENCE {
        extensions      [0] SEQUENCE OF BACnetNameValue OPTIONAL, -- BACnet
        token-type      [1] CharacterString OPTIONAL,             -- RFC 6749
        access-token    [2] BACnetWebToken OPTIONAL,              -- RFC 6749
        id-token        [3] BACnetWebToken OPTIONAL,              -- OpenID Connect
        scope           [5] CharacterString OPTIONAL,             -- RFC 6749
        expires-in      [7] Unsigned OPTIONAL,                    -- RFC 6749
```


no-cache [8] BOOLEAN OPTIONAL -- BACnet
-- Only the IANA parameters with defined interoperable uses are included.
-- Proprietary extensions shall use 'extensions' for other parameters.
}

[Add new entry to **BACnetObjectType**, p. 919]

BACnetObjectType ::= ENUMERATED{
...
audit-reporter (62),
auth (n),
averaging (18),
...
-- numerical order reference
...
~~-- see audit-reporter (62)~~
-- see audit-reporter (62),
-- see auth (n)
}

[Add new entry to **BACnetObjectTypesSupported**, p. 922]

BACnetObjectTypesSupported ::= BIT STRING { -- see below for numerical order
...
~~audit-reporter (62)~~
audit-reporter (62),
auth (n)
}

[Add new entries to **BACnetPropertyIdentifier**, p. 925]

BACnetPropertyIdentifier ::= ENUMERATED { -- see below for numerical order
...
all-writes-successful (9),
allow-factory-authorization (x),
allow-group-delay-inhibit (365),
...
authorization-mode (261),
authorization-server (x+1),
authorization-server-alt (x+2),
auto-slave-discovery (169),
...
device-address-binding (30),
device-groups (x+3),
device-type (31),
...
higher-deck (468),
identity-token (x+4),
identity-server (x+5),
in-process (47),

...
-- -numerical order reference
...
-- see allow-factory-authorization (x),
-- see authorization-server (x+1),


```
        -- see authorization-server-alt     (x+2),
        -- see device-groups                 (x+3),
        -- see identity-token                (x+4),
        -- see identity-server               (x+5),
        ...
        }
```

[Add new entry to **BACnetServicesSupported**, p. 946]

```
BACnetServicesSupported ::= BIT STRING {
        ...
-- Security Services
        -- auth-request     (n),
        ...
-- Services added after 2020
        auth-request        (n), -- Security Service
        ...
        }
```

[Add new ASN.1 productions in **Clause 21** maintaining the alphabetical order, pp. 639]

```
BACnetWebToken ::= SEQUENCE {
        header          [0] BACnetOSEHeader,
        claims          [1] BACnetClaimsSet,
        signature       [3] OCTETSTRING OPTIONAL
        }

BACnetOSEHeader ::= SEQUENCE { -- RFC 7515,7516,7519 JOSE header
        -- selected from IANA "JSON Web Signature and Encryption Header Parameters"
        extension       [0] SEQUENCE OF BACnetNameValue OPTIONAL,
        type            [1] CharacterString OPTIONAL, -- default "BWT"
        algorithm       [2] CharacterString OPTIONAL, -- default "ES256"
        key-id          [3] CharacterString OPTIONAL
        }

BACnetClaimsSet ::= SEQUENCE { -- RFC 7519, JSON Web Token Claims
        extension       [0] SEQUENCE OF BACnetNameValue OPTIONAL,
        issuer          [1] Unsigned OPTIONAL,
        audience        [2] SEQUENCE OF BACnetAudience OPTIONAL,
        scope           [3] CharacterString OPTIONAL,
        subject         [4] CharacterString OPTIONAL,
        confirmation    [5] BACnetConfirmation OPTIONAL,
        expiration      [6] Unsigned OPTIONAL,
        issued-at       [7] Unsigned OPTIONAL,
        not-before      [8] Unsigned OPTIONAL,
        no-cache        [9] BOOLEAN OPTIONAL -- RFC 7519 4.3 Private Claims
        }

BACnetHint ::= SEQUENCE {
        extension       [0] SEQUENCE OF BACnetNameValue OPTIONAL,
        auth-server     [1] Unsigned,
        auth-server-alt [2] Unsigned OPTIONAL,
        audience        [3] BACnetAudience OPTIONAL,
        scope           [4] CharacterString
        }
```



```
BACnetAudience ::= SEQUENCE {
        target   CHOICE {
                   device              [0] Unsigned (0..4194302),
                   group               [1] Unsigned (1..65535)
                   },
        application         [2] CharacterString OPTIONAL
        }


BACnetWebKey ::= SEQUENCE { -- RFC 7517 JSON Web Key
        extension           [0] SEQUENCE OF BACnetNameValue OPTIONAL,
        key-id              [1] CharacterString OPTIONAL,
        usage               [2] CharacterString OPTIONAL,
        key-type            [3] CharacterString OPTIONAL,
        algorithm           [4] CharacterString OPTIONAL,
        curve               [5] CharacterString OPTIONAL,
        x                   [6] OCTET STRING OPTIONAL,
        y                   [7] OCTET STRING OPTIONAL,
        d                   [8] OCTET STRING OPTIONAL,
        key-ops             [9] SEQUENCE OF CharacterString OPTIONAL
        }

BACnetConfirmation ::= SEQUENCE { -- RFC 7800, JWT Confirmation Methods
        extension           [0] SEQUENCE OF BACnetNameValue OPTIONAL,
        key-id              [1] CharacterString OPTIONAL,
        authorized-party    [2] Unsigned OPTIONAL
        }

BACnetSigner ::= SEQUENCE {
        device              Unsigned,
        key1                [1] BACnetWebKey,
        key2                [2] BACnetWebKey OPTIONAL
        }


BACnetAuthRequestError ::= SEQUENCE { -- RFC 6749
        error               CharacterString,
        error-description   [0] CharacterString OPTIONAL,
        error-uri           [1] CharacterString OPTIONAL,
        }
```


135-2020*cg*-10 Interoperability Specifications for Authentication and Authorization

Rationale

This section modifies the PICS statement to describe the security capabilities of the device and defines the BIBBs and Device Profiles for device authentication and authorization.

[Change **Annex A**, p. 694]

BACnet Protocol Implementation Conformance Statement

…

BACnet Standardized Device Profiles (Annex L):

…
☐ ***BACnet Identity Server (B-IS)***
☐ ***BACnet Authorization Server (B-AS)***
…

…

BACnet Authentication and Authorization Options (Annex XX):

☐ *Supports "Secure Source" device identity*
☐ *Automatically retrieves identity token from Identity Server*
☐ *Supports protected resources that require authorization*
☐ *Automatically retrieves access tokens from Authorization Server*
☐ *Supports being a member of a device group. Maximum number of groups supported:* _____

[Add new **Clause K.X**, p. 1108]

K.X Authentication and Authorization BIBBs

K.X.1 BIBB - DeviceIdentityRequest-B (AUTH-DIR-B)

The A device requires an attestation of its identity and requests an identity token from the B device.

BACnet Service	Initiate	Execute
AuthRequest, "identity" endpoint		x

K.X.2 BIBB - DeviceIdentityRequest -A (AUTH-DIR-A)

The A device requires an attestation of its identity and requests an identity token from the B device.

BACnet Service	Initiate	Execute
AuthRequest, "identity" endpoint	x	

K.X.3 BIBB - DeviceAuthorizationRequest-B (AUTH-DAR-B)

The A device is an authorization client and requests an access token from the B device.

BACnet Service	Initiate	Execute
AuthRequest, "access" endpoint		x

K.X.4 BIBB - DeviceAuthorizationRequest -A (AUTH-DAR-A)

The A device is an authorization client and requests an access token from the B device.

BACnet Service	Initiate	Execute
AuthRequest, "access" endpoint	x	

[Change **Annex L**, p.1109]

...
BACnet device profiles are categorized into families:

...

- Elevator Controllers. This family is composed of B-AEC, B-EC, and B-EM.
- *Authorization and Authentication Servers. This family is composed of B-IS and B-AS.*
- Miscellaneous. This family is composed of B-RTR, B-GW, B-BBMD, B-ACDC, B-ACCR, and B-SCHUB.

[Add new **Clauses L.X,** p. 1134]

L.X Authentication and Authorization Profiles

The following table indicates which BIBBs shall be supported by the device types of this family, for each interoperability area.

B-IS	**B-AS**
AUTH-DIR-B	AUTH-DAR-B

L.X.1 Identity Server Profiles

A B-IS device performs the function of an Identity Server, capable of issuing identity tokens to clients.

Identity Attestation

- Maintains a database of mappings between device "subject" names and device instance number.
- Ability to create signed identity tokens that attest to the device instance number for a given device "subject" name.
- Ability to respond with the appropriate identity token when requested.

L.X.1 Authorization Server Profiles

A B-AS device performs the function of an Authorization Server, capable of issuing access tokens to clients.

Permission Granting

- Maintains a database of permissions and policies defining which client devices can perform specific protected operations on specific resources on target devices or device groups.
- Ability to create signed access tokens that grant permission to a client device for a given target device or group.
- Ability to respond with the appropriate access token when requested.

[Add a new entry to **History of Revisions**, p. 1429]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

1 HISTORY OF REVISIONS

…	…	**…**
1	X	**Addendum *cg* to ANSI/ASHRAE 135-2020** Approved by the ASHRAE Standards Committee MONTH X, 20XX; by the ASHRAE Board of Directors MONTH X, 20XX; and by the American National Standards Institute MONTH X, 20XX. 1. Changes to BACnet Architecture to support Device Authentication and Authorization. 2. Modifications to Annex AB related to Authentication and Authorization. 3. Add AuthRequest Service. 4. Add example Authentication and Authorization Message Flows. 5. Add Auth Object Type. 6. Modify existing services to specify error codes related to Authentication and Authorization. 7. Add new scopes to Annex W. 8. Add new Error and Abort codes for Authentication and Authorization. 9. Define new Clause 21 data structures for Authentication and Authorization. 10. Add new BIBBs to Annex K and Device Profiles to Annex L.