ANSI/ASHRAE Addendum g to
ANSI/ASHRAE Standard 135-2008

ASHRAE ADDENDA

BACnet®—A Data Communication Protocol for Building Automation and Control Networks

Approved by the ASHRAE Standards Committee on June 26, 2010; by the ASHRAE Board of Directors on June 30, 2010; and by the American National Standards Institute on July 1, 2010.

This addendum was approved by a Standing Standard Project Committee (SSPC) for which the Standards Committee has established a documented program for regular publication of addenda or revisions, including procedures for timely, documented, consensus action on requests for change to any part of the standard. The change submittal form, instructions, and deadlines may be obtained in electronic form from the ASHRAE Web site (www.ashrae.org) or in paper form from the Manager of Standards.

The latest edition of an ASHRAE Standard may be purchased on the ASHRAE Web site (www.ashrae.org) or from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: orders@ashrae.org. Fax: 404-321-5478. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada). For reprint permission, go to www.ashrae.org/permissions.

ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

American Society of Heating, Refrigerating
and Air-Conditioning Engineers, Inc.
1791 Tullie Circle NE, Atlanta, GA 30329
www.ashrae.org

ASHRAE Standing Standard Project Committee 135
Cognizant TC: TC 1.4, Control Theory and Application
SPLS Liaison: Douglas T. Reindl

ASHRAE STANDARDS COMMITTEE 2009–2010

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus standard developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). *Consensus* is defined by the American National Standards Institute (ANSI), of which ASHRAE is a member and which has approved this standard as an ANS, as “substantial agreement reached by directly and materially affected interest categories. This signifies the concurrence of more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that an effort be made toward their resolution.” Compliance with this standard is voluntary until and unless a legal jurisdiction makes compliance mandatory through legislation.

ASHRAE obtains consensus through participation of its national and international members, associated societies, and public review.

ASHRAE Standards are prepared by a Project Committee appointed specifically for the purpose of writing the Standard. The Project Committee Chair and Vice-Chair must be members of ASHRAE; while other committee members may or may not be ASHRAE members, all must be technically qualified in the subject area of the Standard. Every effort is made to balance the concerned interests on all Project Committees.

The Assistant Director of Technology for Standards and Special Projects of ASHRAE should be contacted for:

a. interpretation of the contents of this Standard,
b. participation in the next review of the Standard,
c. offering constructive criticism for improving the Standard, or
d. permission to reprint portions of the Standard.

DISCLAIMER

ASHRAE uses its best efforts to promulgate Standards and Guidelines for the benefit of the public in light of available information and accepted industry practices. However, ASHRAE does not guarantee, certify, or assure the safety or performance of any products, components, or systems tested, installed, or operated in accordance with ASHRAE’s Standards or Guidelines or that any tests conducted under its Standards or Guidelines will be nonhazardous or free from risk.

ASHRAE INDUSTRIAL ADVERTISING POLICY ON STANDARDS

ASHRAE Standards and Guidelines are established to assist industry and the public by offering a uniform method of testing for rating purposes, by suggesting safe practices in designing and installing equipment, by providing proper definitions of this equipment, and by providing other information that may serve to guide the industry. The creation of ASHRAE Standards and Guidelines is determined by the need for them, and conformance to them is completely voluntary.

In referring to this Standard or Guideline and in marking of equipment and in advertising, no claim shall be made, either stated or implied, that the product has been approved by ASHRAE.

[This foreword and the "rationale" on the page 5 are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

Addendum 135*g* to ANSI/ASHRAE Standard 135-2008 contains a number of changes to the current standard. 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 changes are summarized below.

135-2008*g*-1. Update BACnet Network Security, p. 5.

In the following document, language added to existing clauses of ANSI/ASHRAE 135-2008 and addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are added, plain type is used throughout.

SSPC 135 is especially grateful for the dedicated efforts of the following people in developing this addendum: Carl Neilson and Alexander Andreyev, Coleman Brumley, James Butler, Clifford Copass, Philippe Goetz, David Holmberg, Frank Liese, and Dave Robin.

CONTENTS

24 NETWORK SECURITY 5
24.1 Overview 5
24.1.1 Security Layer 5
24.1.2 Shared Keys 5
24.1.3 Securing Messages 6
24.1.4 Network Security Policies 7
24.1.5 Device Level Security 7
24.1.6 Secure Tunnel Mode 7
24.1.7 User Authentication 8
24.1.8 Key Distribution 8
24.1.9 Deployment Options 8
24.1.10 Limitations and Attacks 8
24.1.11 Minimum Device Requirements 9
24.2 Security Wrapper 10
24.2.1 Security Header Protocol Control Information 10
24.2.2 Key Revision 11
24.2.3 Key Identifier 11
24.2.4 Source Device Instance 11
24.2.5 Message Id 11
24.2.6 Timestamp 11
24.2.7 Destination Device Instance 12
24.2.8 DNET/DLEN/DADR 12
24.2.9 SNET/SLEN/SADR 12
24.2.10 Authentication Mechanism 12
24.2.11 Authentication Data 13
24.2.12 Service Data 13
24.2.13 Padding 14
24.2.14 Signature 14
24.3 Security Messages 14
24.3.1 Challenge-Request 14
24.3.2 Security-Payload 16
24.3.3 Security-Response 18
24.3.4 Request-Key-Update 21
24.3.5 Update-Key-Set 23
24.3.6 Update-Distribution-Key 27
24.3.7 Request-Master-Key 29
24.3.8 Set-Master-Key 29
24.4 Securing an APDU 30
24.5 Securing an NPDU 31
24.6 Securing a BVLL 32
24.7 Securing Messages 34
24.7.1 Message Id 34
24.7.2 Timestamp 34
24.7.3 Device Identification 35
24.7.4 Message Signature 35
24.7.5 Encrypted Messages 36
24.8 Network Security Network Trust Levels 37
24.8.1 Trusted Networks 37
24.8.2 Non-trusted Networks 37
24.9 Network Security Policies 37
24.9.1 Plain-Non-Trusted 37
24.9.2 Plain-Trusted 38
24.9.3 Signed-Trusted 38
24.9.4 Encrypted-Trusted 38

24.10 Network Security 38
24.11 End-to-End Security 39
24.11.1 Determining Exceptional Security Requirements 39
24.12 Wrapping and Unwrapping Secure Messages 39
24.12.1 Wrapping and Unwrapping By Routers 39
24.12.2 Securing Response Messages 41
24.13 Authenticating Messages 41
24.13.1 Validating the Security Header Protocol Control Information 42
24.13.2 Validating the Signature 42
24.13.3 Validating the Source MAC Address 43
24.13.4 Validating the Destination Device Instance 43
24.13.5 Validating the Message Id 43
24.13.6 Validating the Timestamp 44
24.14 User Authentication 44
24.14.1 Proxied User Authentication 44
24.15 Time Synchronization Requirements 45
24.15.1 BACnet Time Synchronization Messages 45
24.15.2 Overcoming Non-synchronized Clocks 45
24.16 Integrating the Security Layer into the BACnet Stack 46
24.16.1 Secure PDU Sizes 46
24.16.2 Selecting Error Codes 47
24.16.3 Communicating Security Parameters 47
24.16.4 Detecting and Processing Security Errors 47
24.16.5 Security Errors in Network Layer Initiated Packets 52
24.16.6 Security Errors in BACnet/IP BVLL Initiated Packets 52
24.16.7 Data Hiding 52
24.16.8 Device Identity 53
24.17 BACnet Security In A NAT Environment 54
24.18 BACnet Security Proxy 54
24.19 Deploying Secure Device on Non-Security Aware Networks 54
24.20 Deploying Secure Single Network Installations 54
24.21 Security Keys 54
24.21.1 Key Identifiers 55
24.21.2 Key Sets 56
24.21.3 Key Distribution 56
24.22 Key Server 56
24.22.1 Key Generation 56
24.22.2 Distribution Method 57
24.22.3 Initial Key Distribution 58
24.22.4 Key Revision 59
24.22.5 Sites Without Key Servers 59
24.22.6 Multiple Key Servers 59
5.1.2 Unconfirmed Application Services 62
5.4.4 State Machine for Requesting BACnet User (client) 62
5.4.5 State Machine for Responding BACnet User (server) 68
6.1 Network Layer Service Specification 72
6.2.4 Network Layer Message Type 73
6.4.11 Challenge-Request 74
6.4.12 Security-Payload 74
6.4.13 Security-Response 75
6.4.14 Request-Key-Update 75
6.4.15 Update-Key-Set 75
6.4.16 Update-Distribution-Key 75
6.4.17 Request-Master-Key 75
6.4.18 Set-Master-Key 75
6.4.19 What-Is-Network-Number 75

6.4.20 Network-Number-Is......75
12.11.17 Max_APDU_Length_Accepted 76
12.X Network Security Object Type......77
12.X.1 Object_Identifier 77
12.X.2 Object_Name 77
12.X.3 Object_Type......77
12.X.4 Description......77
12.X.5 Base_Device_Security_Policy 78
12.X.6 Network_Access_Security_Policies......78
12.X.7 Security_Time_Window 78
12.X.8 Packet_Reorder_Time......78
12.X.9 Distribution_Key_Revision......78
12.X.10 Key_Sets 78
12.X.11 Last_Key_Server......78
12.X.12 Security_PDU_Timeout......78
12.X.13 Update_Key_Set_Timeout......79
12.X.14 Supported_Security_Algorithms......79
12.X.16 Profile_Name 79
18.5 Error Class – SECURITY 79
18.7 Error Class – COMMUNICATIONS......81
K.X BIBB - Network Security BIBBs 93
K.X.1 BIBB - Network Security - Secure Device (NS-SD)......93
K.X.2 BIBB - Network Security - Encrypted Device (NS-ED)......93
K.X.4 BIBB - Network Security - Multi-Application Device (NS-MAD) 93
K.X.5 BIBB - Network Security-Device Master Key-A (NS-DMK-A) 94
K.X.6 BIBB - Network Security-Device Master Key-B (NS-DMK-B)......94
K.X.7 BIBB - Network Security-Key Server (NS-KS)......94
K.X.8 BIBB - Network Security-Temporary Key Server (NS-TKS)......94
K.X.9 BIBB - Network Security-Secure Router (NS-SR) 95
K.X.10 BIBB - Network Security-Security Proxy (NS-SP)......95
R.1 Example of an Initial Key Distribution 98
R.2 Example of Device Startup......102
R.3 Examples of Secured Confirmed Requests 104
R.3.1 ReadProperty Example 104
R.3.2 ReadProperty Error Example......105
R.3.3 Segmented ReadProperty Example 106
R.4 Security Challenge Example......110
R.5 Secure-BVLL Example 112

135-2008*g*-1. Update BACnet Network Security.

Rationale

The existing BACnet Network Security architecture defined in Clause 24 is based on the 56-bit DES cryptographic standard and needs to be updated to meet the needs of today's security requirements.

Addendum 135-2008*g*-1

[Replace existing **Clause 24** in its entirety, p. 483, with the following new **Clause 24**.]

24 NETWORK SECURITY

This clause defines a security architecture for BACnet. Network security in BACnet is optional. The intent of this architecture is to provide peer entity, data origin, and operator authentication, as well as data confidentiality and integrity. Other aspects of communications security, such as authorization policies, access control lists, and non-repudiation, are not defined by this standard. Systems that require these functions may add them to BACnet by using the proprietary extensibility features provided for by this architecture, or by some other proprietary means.

24.1 Overview

The BACnet network security architecture provides device authentication, data hiding, and user authentication. This has been accomplished within the constraints that BACnet security should allow for:

(a) Application to all BACnet media types (BACnet/IP, MS/TP, etc.)
(b) Application to all BACnet device types (devices, routers, BBMDs)
(c) Application to all message types (broadcast, unicast, confirmed, and unconfirmed)
(d) Application to all message layers (BVLL, network, and application)
(e) Placing non-security-aware devices, if physically secure, behind a secure proxy firewall router
(f) Placing secure devices on non-security-aware networks.

To achieve these network security goals, the BACnet standard is extended with a set of network layer security messages. Other security standards, such as IPsec and Kerberos, were designed to operate only on TCP/IP networks and as such do not meet the above requirements. However, the BACnet security architecture was developed by applying the best security practices of those standards that fit the requirements listed above.

24.1.1 Security Layer

The security functionality is added into the BACnet stack as a set of network layer messages. As such, there is no actual security layer, although the discussion of security processing is easiest to understand if it is conceptually separated into a distinct layer. For this reason the security processing and the related messages are referred to as the security layer although in fact they are part of the network layer.

24.1.2 Shared Keys

The BACnet security model relies on the use of shared secrets called keys. Device and user authentication is achieved through the use of message signatures and shared signature keys. Data hiding is achieved through encryption of the secure payload and shared encryption keys.

In BACnet security, keys are always distributed as key pairs, where one half is the signature key and the other half is the encryption key. There are 6 types of key pairs: General-Network-Access, User-Authenticated, Application-Specific, Installation, Distribution, and Device-Master.

The General-Network-Access key is used for broadcast network layer messages, for encryption tunnels, and by user interface devices that cannot authenticate, or are not trusted to authenticate, a user. All devices must be given the General-Network-Access key pair to interoperate on a BACnet network. BACnet server devices that receive requests signed with the General-Network-Access key should assume that the User Id and User Role fields included in the message may not have been properly authenticated by the source device and may want to restrict access accordingly.

The User-Authenticated key is distributed to client devices that are trusted to authenticate a user's identity by some means, or to devices that do not contain a user interface (where the user identity to use in BACnet messages is configured into the device and is not based on human interaction). This key is also distributed to BACnet server devices that restrict operations based on the identity of an authenticated user. Servers that receive requests that are signed with the User-Authenticated key can assume that the User Id and User Role fields included in the message has been properly authenticated by the client device, or was configured into a trusted device with no user interface. While the client device may restrict a user's actions based on its authorization policies prior to sending the message, the server device is also free to restrict access based on the received User Id and User Role.

An Application-Specific key may be used to provide security boundaries between application areas, such as access control and HVAC. Application-Specific keys are distributed only to those devices sharing a particular application and can thus be limited to highly secure communication. Devices using Application-Specific keys for highly secure communications should be designed to be able to restrict which services can be executed with lesser keys. For example, such devices might be configured to disallow time synchronization or network configuration via the General-Network-Access key.

Installation keys are distributed temporarily to small sets of devices, usually the configuration tool of a technician and a set of BACnet devices that require configuration. These keys are provided to allow temporary access to a specific set of controllers through a configuration tool that would not normally have access to the BACnet network. There may be multiple Installation keys in use by different devices simultaneously, so that different configuration tools could use different Installation keys, if desired.

The Distribution keys are used to distribute the General-Network-Access, User-Authenticated, and Application-Specific keys, which may change over time as needed to meet local security policies. They are also used to distribute the temporary Installation keys.

The Device-Master keys are used only for the distribution of the Distribution keys and remain the most secure of all key types because they are unique for every device and their use on the wire is very limited.

24.1.3 Securing Messages

Security is applied at the network layer by creating a new NPDU message type. Plain BACnet messages are secured by placing the NSDU portion of the message into the Payload of a Security-Payload message. Therefore, when a BACnet APDU is encapsulated with security information, it is transported as a network layer message and the control bit in the NPCI is changed to indicate that the message now contains a network layer message rather than an APDU. The security header will indicate that the encapsulated message is an APDU so that this information is not lost. Upon unwrapping this message, this control bit will change back so that the plain NPDU will once again indicate that it contains an APDU.

For NPDUs and BVLLs containing NPDUs, the portion of the message starting with the network layer Message_Type field is placed into the Payload of a Security-Payload message. For BVLL messages that do not contain an NPDU, the original BVLL is embedded in a Secure BVLL message.

The basic level of security that can be applied to a BACnet message consists of signing each message using HMAC (keyed-hash message authentication algorithm) and MD5 or SHA-256 (commonly used hash algorithms), and of marking each message with the source and destination Device instances, a Message Id and a timestamp. Including source and destination addresses and source and destination device instances assures that messages cannot be spoofed or redirected. However, this requires that all secure devices, even routers and BBMDs, contain an application layer and device object.

Message Id fulfills several purposes in securing BACnet messages. It is used to detect the replay of messages, to associate security responses with security requests, and along with the Timestamp field, to provide variability in otherwise identical messages.

Timestamp is used mainly for prevention of message replay but also serves as a source of variability in the message content so that messages that are repeated frequently do not generate the same signature. The clocks of secure devices must be loosely synchronized. If a timestamp on a message is outside the security time window, then an error is returned and clock issues need to be addressed. Within the security time window, Message Ids are checked to confirm that a message has not been replayed.

A higher level of security is provided by encrypting BACnet messages so that the content of the message cannot be determined without the possession of an appropriate key. Even the length of the message can be obscured by using a varying amount of hidden padding.

24.1.4 Network Security Policies

There are two network trust levels – trusted and non-trusted. Networks can be designated as trusted due to being physically secure, or due to the use of protocol security (signatures and/or encryption). Non-trusted networks are those which are both physically non-secure and not configured to require protocol security.

BACnet messages that do not have any security information in them are referred to as "plain" messages. Therefore, there are four corresponding network security policies: plain-trusted (requires physical security; no protocol security applied), signed-trusted (physical security not required; secured with signatures), encrypted-trusted (physical security not required; secured with encryption), and plain-non-trusted (not physically secure; no signature or encryption applied). A common example of a plain-trusted network is an MSTP network where all devices are locked up and no direct network connections are available outside of the locked space. Devices that do not support the BACnet security messages must reside only on plain-trusted networks for their communications to be trusted by secure devices. A common example of a plain-non-trusted network would be the corporate LAN. However, the LAN may be re-designated as signed-trusted or encrypted-trusted by requiring all BACnet devices on the LAN to implement BACnet security and sign/encrypt all messages.

24.1.5 Device Level Security

Secure Devices are not restricted to residing on trusted networks (plain-trusted, signed-trusted, or encrypted-trusted). Secure devices may be located on non-trusted networks and rely on end-to-end (device level) security for secure communications. While trusted networks are created by setting the security policy for a network, and all devices on a trusted network must be configured with the security policy of the network, end-to-end security is determined on a device by device and request by request basis.

Secure BACnet devices are configured with a base device security policy that dictates the device's minimum level of security for sending or receiving messages. This policy may be higher than, but not lower than, the network access security policy.

Incapable Devices (devices that are not capable of processing BACnet security messages or those that have been configured to not be able to process BACnet security messages) must reside on a plain network (plain-trusted or plain-non-trusted). Secure devices can also reside on this same network, but their Base_Device_Security_Policy property must be set to PLAIN if they need to communicate with the incapable devices. Even if the Base_Device_Security_Policy property is set to PLAIN for interoperability with incapable devices, the secure devices are free to use secured messages, for communicating with other secure devices, for any traffic that needs to be secured.

24.1.6 Secure Tunnel Mode

The standard allows for a tunnelling mode whereby plain and signed packets arriving at one end of the tunnel (e.g., router A on subnet A) can be tunnelled to another device (e.g., router B on subnet B across a non-physically-secure network segment). The tunnelling router applies encryption (and signature if needed) using the General-Network-Access key and forwards the packet along to the other end of tunnel. Control bits in the security header indicate that the packet has been tunnelled. If a packet is already encrypted, the tunnelling router passes the message as is.

To avoid inverted networks, it is recommended that only BACnet/IP be used for secure tunnels when connecting non-secure BACnet/IP or BACnet/Ethernet networks. BACnet/IP is preferred for secure tunnels since it is the only medium through which full size BACnet packets (1476 octets of APDU) can be transferred when security is enabled. In such installations, all BACnet products can take advantage of the secure tunnel, not just those that are security aware or only communicate with smaller PDU sizes.

24.1.7 User Authentication

The BACnet security architecture allows for multiple methods for user authentication. Currently only a single method of user authentication is defined: Proxied User Authentication.

Proxied User Authentication relies on site policy and trust of selected software to perform user authentication. To allow for some clients to be trusted to perform user authentication, and some clients that do not perform, or are not trusted to perform, user authentication, different security keys are provided. Clients with user interfaces that are trusted to perform user authentication are given the User-Authenticated key, or an Application-Specific key. Other clients that need access to the network but are not trusted to securely authenticate users are given the General-Network-Access key.

24.1.8 Key Distribution

BACnet security keys are distributed to all devices by a BACnet Key Server. The General-Network-Access, User-Authenticated, Application-Specific, and Installation keys are bundled into a set and distributed together with a single key revision number, each device receiving a specific set of keys appropriate for that device. While different devices may receive different key sets (differing in Application-Specific or Installation keys, for example), the key sets shall share the same revision number across all devices after a key distribution is complete.

Each BACnet device shall either have a unique factory-fixed Device-Master key, or support initiation of Request-Master-Key service and execution of the Set-Master-Key service. The Key Server will use a device's Device-Master key to securely provide the device with a device specific Distribution key. The Key Server will then use the Distribution key to send the device its set of security keys. Distribution keys are therefore revised separately from other keys, as they may change less frequently.

A full description of the key distribution protocol is defined in Clause 24.21.3.

All secure devices shall support the key distribution messages defined in this standard. In addition, they may also support proprietary mechanisms for setting keys. For example, an installation tool may configure an initial key set as part of its programming and commissioning operations.

24.1.9 Deployment Options

Security deployment always involves careful consideration for balancing costs, complexity, and time of configuration and maintenance against the likelihood of various attack scenarios and the sensitivity of the data or actions being protected. This standard provides for a continuum of protection from very simple and coarse grained to very powerful and fine grained.

Using the architecture defined here, very simple deployments can be made. Some deployments may not require a live Key Server. In these cases, the function of the Key Server is performed by the installation tool(s) and all devices are given infinite duration keys so that no Key Server is needed after installation. In addition, all key values can be set to be the same value if only a moderate level of security is needed to protect moderately critical resources.

Also using the architecture defined here, highly specific and highly secure deployment requirements can be met by segregating collections of devices using Application-Specific keys and tightly controlling the distribution of those keys to a limited number of devices. In addition, a live Key Server can be used to distribute expiring keys periodically according to site policy. User information is provided so that fine grained authorization policies (e.g., access control lists) can be based on the source device and/or the source user or process. The authentication mechanism can be extended to support complex proprietary methods, if required.

24.1.10 Limitations and Attacks

Highly secure communications between peer devices requires not only the knowledge of the proper key(s), but also the knowledge of a peer device's device instance number as well. This is because there are attack scenarios where it may be possible for the source and destination address information (SNET, SADR, DNET, DADR) to be altered. The relative ease or difficulty of these attacks is affected by the site's physical access policies and the skill and equipment of the attackers.

Altering the addressing information may be accomplished by gaining physical access to a secured device and changing its MAC address (e.g., by changing its address switches), by causing its IP address to change (e.g., by spoofed DHCP messages or a physically inserted NAT device), or by placing it on another network, either by physically moving the device or by remotely rewiring the networks.

Secure devices should not allow their instance numbers to be changed by physical switches after installation; device instance numbers should only be changeable via secured communications with a configuration tool. Therefore, the device instance number is the most trustworthy form of identifying the source or destination of a message, and highly secured communications should always include the destination device instance number (the source instance is always known and always included).

Devices receiving messages where the device instance of the destination is unknown should act accordingly based on their internal policies for the operation being requested. The device instance of the source is always known and may be used by the destination device's internal policies for determining how to handle these messages. In many cases, the knowledge by the destination of the authorized source instances may be sufficient to relieve the source of having to know the destination's instance.

There are also ways to avoid the condition of a source device not knowing the instance of a destination. For example, the device instance form of a recipient address should be used rather than the address form, and services like "Subscribe COV" should record the requesting device instance along with its address.

Secure devices should restrict the setting of their device instance number to communications that are secured with an Installation key, which may be temporary and unique to the device. Site policies should restrict user access to software that is authorized to change instance numbers in secure devices. But since this software is likely the same software that can completely reprogram the devices, this policy may already be in place. Site policies should also restrict physical access to highly secured devices so that their internal memory cannot be physically tampered with. Here again, this is likely to be an existing policy for such devices.

Many of the above attacks involve physical access to either secured devices themselves or to the wiring between devices. Given this opportunity, Denial of Service attacks are trivial and obvious and this standard does not address their prevention. However, to limit over-the-wire Denial of Service attacks, this standard allows some error conditions to be ignorable. For example, devices that want to hide from scanners are allowed to ignore messages that are using an unknown key or appear to be replayed.

In general, error responses are helpful for diagnosing or recovering from some forms of legitimate network problems, however, some devices may want to limit repeated error responses to repeated receipt of erroneous messages, which may actually be an attempt at a Denial of Service attack. Legitimate devices should be designed to recover from errors like outdated key sets or incorrect timestamps in a reasonable manner or should limit their rate of sending unsuccessful messages to avoid creating an inadvertent Denial of Service attack by repeatedly sending erroneous messages to other secure devices.

24.1.11 Minimum Device Requirements

In order to implement BACnet network security in a device, the device shall:

(a) have an application layer;
(b) support execution of WriteProperty;
(c) be able to track time;
(d) have non-volatile re-writable storage in which to retain some run-time and configuration data;
(e) not be an MS/TP slave.

In addition, it is recommended that secure devices have a real-time clock that is persistent across resets and extended power down periods.

24.2 Security Wrapper

All BACnet security messages use the same security wrapper consisting of a header, an optional body, and a required signature. The format of the wrapper is:

Table 24-1. Security Wrapper Format

Field Name	Size
Control	1 octet
Key Revision	1 octet
Key Identifier	2 octets
Source Device Instance	3 octets
Message Id	4 octets
Timestamp	4 octets
Destination Device Instance	3 octets
DNET	2 octets
DLEN	1 octet
DADR	Variable
SNET	2 octets
SLEN	1 octet
SADR	Variable
Authentication Mechanism	1 octet
Authentication Data	Variable
Service Data	Variable
Padding	Variable
Signature	16 octets

All multi-octet fields shall be conveyed with the most significant octet first. The DADR and SADR fields shall be encoded as described in Clause 6.

24.2.1 Security Header Protocol Control Information

Each security message NPDU shall start with a control octet that includes indications of the presence or absence of particular security header fields.

Bit 7: 1 indicates that the Payload contains a network layer message or a Secure-BVLL.
0 indicates that the Payload contains an application layer message

Bit 6: 1 indicates that the message is encrypted.
0 indicates that the message is not encrypted.
This bit is referred to as the 'encrypted flag' and shall always be 0 when calculating the signature for the message.

Bit 5: Reserved. Shall be 0.

Bit 4: 1 indicates that the Authentication Mechanism and Authentication Data fields are present.
0 indicates that the Authentication Mechanism and Authentication Data fields are absent.
The Authentication Mechanism and Authentication Data fields are optionally present on request messages but shall be absent from response messages (e.g., Complex Ack, Simple Ack, Security Response)

Bit 3: 1 indicates that the Security Wrapper should not be removed, except by the destination device. This bit shall be 1 if Bit 2 (the 'do-not-decrypt flag') is set to 1.
0 indicates that the Security Wrapper should be removed before placing the message on a plain network segment.
This bit is referred to as the 'do-not-unwrap flag'.

Bit 2: 1 indicates that encryption should not be removed, except by the destination device. If this bit is set to 1, then Bit 3 (the 'do-not-unwrap flag') shall be set to 1. This bit shall not be 1 when Bit 6 ('encrypted flag') is set to 0.
0 indicates that encryption should be removed before placing the message on a network segment that does not require encryption.
This bit is referred to as the 'do-not-decrypt flag'.

Bit 1: 1 indicates that the message was received from a plain-non-trusted network and that the security information was placed on the message by the router from the plain-non-trusted network to a trusted-signed or trusted-encrypted network. Routers should not route plain messages from plain-non-trusted network to a plain-trusted network.
0 indicates that the message originated on a trusted network, or that the originator applied the security header.
This bit is referred to as the 'non-trusted-source flag'.

Bit 0: 1 indicates that the message was secured by an intervening router.
0 indicates that the message was secured by the originator.
This bit is referred to as the 'secured-by-router flag'.

24.2.2 Key Revision

This field shall contain the key revision for the key identified by the Key Identifier field.

This field shall be 0 when the Key Identifier indicates the Device-Master key.

24.2.3 Key Identifier

The Key Identifier field specifies the key that is used to sign the message. If the do-not-decrypt flag has a value of 1, then it also specifies the key used to decrypt the message. If the do-not-decrypt flag has a value of 0, the General-Network-Access key is used to decrypt the message as it is the only key that is guaranteed to be known by intermediate routers (see Clause 24.21.1).

24.2.4 Source Device Instance

The Source Device Instance is the Device object instance of the device that applied security to the message.

The field shall be restricted to the range 0 through 4194302. This requires that all secure BACnet devices, even those that are only routers or BBMDs, contain an application layer and a Device object.

Note that this field cannot be used to identify the source device of a message when the secured-by-router flag is set as it will indicate the router's Device object instance.

24.2.5 Message Id

The Message Id is a 32 bit monotonically increasing counter value that is present in all secure messages. It is used for matching security responses to security requests, for preventing replay attacks, and along with the Timestamp provides variability between messages that might otherwise be identical.

In the normal course of operation, a device shall not generate more than one message with the same Message Id within the security time window.

If a device does not remember its Message Id across resets, then the device may have problems communicating for the first security time window period. Such a condition should be expected if the device resets within the first security time window period of a previous reset, and it always resets its Message Id counter to the same value on reset. Waiting 2 * Security_Time_Window seconds before communicating will overcome this problem.

24.2.6 Timestamp

The Timestamp field, an unsigned 32 bit integer, indicates the time of the message in UTC as seconds since 12:00 AM January 1, 1970 (standard Unix timestamp).

24.2.7 Destination Device Instance

The Destination Device Instance is the Device object instance of the destination device for the message. A value of 4194303 shall be used in all broadcast messages and when the device instance of the destination device is unknown to the device applying the security. Secure devices shall attempt to determine the Device object instance of the destination device, and only if attempts to determine the value fail, shall a secure device resort to the use of 4194303 in unicast messages.

24.2.8 DNET/DLEN/DADR

These fields contain the values of the fields with the same name from the NPCI portion of the message. They are always present and are included in the security header to allow the signing of the values.

When the message is to be placed onto the destination network, or is received from the destination network, the NPCI will not contain the DNET/DLEN/DADR fields. Regardless of whether the NPCI contains the DNET/DLEN/DADR fields, the security header shall contain these fields and they shall contain the correct destination address information.

For the Update-Key-Set, Update-Distribution-Key, and Set-Master-Key, the DNET shall be set to 0 in the security header if the SNET was 0 in the corresponding Request-Key-Update or Request-Master-Key message.

24.2.9 SNET/SLEN/SADR

These fields correspond to the fields with the same name from the NPCI portion of the message. They are always present and are included in the security header to allow the signing of the values. As such, the values must be known and filled in by the device. This is in contrast to non-secure BACnet messages where these fields in the NPCI are only present when added by a router when routing remote messages.

When a security header is placed in a message by a router on behalf of another device, these fields shall contain the address information of the originating device and not the address information of the router.

There are exceptions where the values are not known and cannot be filled in by the sending device. In the What-Is-Network message, the SNET shall be set to 0. In the Request-Key-Update, and Request-Master-Key, the SNET shall be set to 0 only when the device does not know its network number. However, the SLEN and SADR shall be set to valid values, if they are known. In the case where a device temporarily does not know its own SADR, such as a BACnet/IP device behind a NAT firewall, the SLEN shall be set to 0 and the SADR shall be empty. These devices shall learn their SADR by reading the destination address of any properly authenticated message sent to it.

When the message is to be placed onto the source network, or is received from the source network, the NPCI will not contain the SNET/SLEN/SADR fields. Regardless, the security header shall contain these fields and they shall contain the correct source address information.

24.2.10 Authentication Mechanism

If present, the Authentication Mechanism field is a 1 octet value that indicates the user authentication mechanism being used. This field shall be present when the User-Authenticated or an Application-Specific key is used and the PDU type is one that initiates a request (see below). It shall be absent when the Device-Master, Distribution, or Installation key is used. And it shall be optional when the General-Network-Access key is used.

User authentication information is not useful in responses or transmission control. User authentication is useful in any PDU type that initiates a request:

APDU PDU Types:	Confirmed-Request, Unconfirmed-Request,
NPDU PDU Types:	Initialize-Routing-Table, Establish-Connection-To-Network, Disconnect-Connection-To-Network,
BVLL Types:	Write-Broadcast-Distribution-Table, Read-Broadcast-Distribution-Table, Register-Foreign-Device, Read-Foreign-Device-Table, Delete-Foreign-Device-Table-Entry.

It shall not be included in all other PDU types, but if it is present a receiving device shall ignore it. If user authentication information is provided in a segmented APDU, the authentication information shall be the same in all segments. User authentication information in response PDUs and transmission control PDUs shall not be present.

The proxied user authentication mechanism is indicated by a value of 0 and is the only standardized mechanism at this time.

Values in the range 200 through 255 are reserved for vendor specific mechanisms.

24.2.11 Authentication Data

The Authentication Data field is a variable length identifier that provides authentication information in a format specific to the mechanism defined by the Authentication Mechanism field.

This may be used by the server's authorization mechanism to verify that the user is allowed to perform the requested action.

A client device that authenticates users may be given the User-Authenticated key or an Application-Specific key. It shall indicate the authenticated user's identity when initiating communication.

This field is always at least three octets in length. The first two octets are an unsigned integer (most significant octet first) indicating the numeric User Id for the user that is authenticated for this message. The third octet is the user's role or group. The user role values are site specific values dictated by site policy and are used to group access rights. Example roles are: HVAC operator, technician, etc.

User Id values represent either unique human users, or processes within a BACnet system. Assignment of the values is based on local site policy, but they should be unique across all BACnet devices, such that User Id 1234, for example, means the same regardless of its source or destination.

User Roles 0 and 1 are reserved to mean "the system itself". User Role 0 is used for programmed device-to-device communication that is not initiated by human action. A User Role of 1 is used for device-to-device communication that is initiated by an "unknown human", such as the changing of a setpoint based on button presses on a thermostat.

Other User Role values may also be used for device-to-device communication to indicate a particular subsystem that is performing the action, but those values are not restricted by this standard and are taken from the same set of numbers as are used for human users and groups. The values 0 and 1 are the only ones that are reserved specifically for this purpose and shall not be assigned to human user roles.

User Id 0 is reserved to indicate that the source user is unknown. It is commonly used in conjunction with User Role 0 or 1.

If the Authentication Mechanism has a value of 0, then the Authentication Data field contains no further information since the authentication has been performed by the source.

If the Authentication Mechanism has a value of 1 through 199, then the next 2 octets of this field shall be an unsigned integer (most significant octet first) indicating the length, in octets, of the entire field. The meaning of the remaining octets is not currently defined by this version of this standard.

If the Authentication Mechanism has a value of 200 through 255, then the next 2 octets of this field shall be an unsigned integer (most significant octet first) indicating the length, in octets, of the entire field. Following that, the next 2 octets shall be an unsigned integer (most significant octet first) indicating a BACnet Vendor Identifier. The meaning of the remaining octets is defined by that vendor.

24.2.12 Service Data

The Service Data field contains security specific data. Its content varies by message type.

The security NPCI message type values are:

X'0A': Challenge-Request

X'0B': Security-Payload
X'0C': Security-Response
X'0D': Request-Key-Update
X'0E': Update-Key-Set
X'0F': Update-Distribution-Key
X'10': Request-Master-Key
X'11': Set-Master-Key

24.2.13 Padding

The padding is present if and only if the message is encrypted. This field is sized to ensure that the length of the data being encrypted is a multiple of the encryption algorithm's block size. The padding field is added after the signature is calculated.

The last two octets of the padding field are a count (most significant octet first) that indicates the total number of octets of padding, including the count itself. The values of all remaining octets are unspecified.

Since the count includes itself, and cannot be zero, the padding field is always included if the message is encrypted.

The size of the padding field may be increased, by adding multiples of the block size to the minimum requirement, to allow devices to hide the true length of their encrypted messages.

24.2.14 Signature

The signature contains an HMAC of the message. See Clause 24.7.4 for details on generating the signature.

The signature (in whole or in part) is also used as the Initialization Vector for the encryption algorithm.

24.3 Security Messages

24.3.1 Challenge-Request

The Service Data for a Challenge-Request message has the following form:

Table 24-2. Challenge-Request Service Data

Message Field	Size	Description
Message Challenge	1 octet	When set to 1, this field indicates that the Challenge-Request is being sent in response to a message. Otherwise, the Challenge-Request is being sent for some other reason and the following fields contain random data.
Original Message Id	4 octets	The Message Id from the message that caused the device to issue the challenge.
Original Timestamp	4 octets	The timestamp from the message that caused the device to issue the challenge.

Any device that receives a secure BACnet message may, at the device's discretion, challenge the message source, unless the secure message was itself a Challenge-Request. Specific cases where a device may want to challenge a message source are as follows: on receipt of an I-Am or I-Am-Router-To-Network where the source address does not match a previously cached value, on receipt of a secure message where the source MAC address does not match the source address in the secured NPDU and both devices are on the same BACnet/IP network, on receipt of a message where SLEN in the security wrapper is 0, and on receipt of a unicast message where the Destination Device Instance is 4194303. When challenging a specific message, the Message Challenge field shall be set to 1.

A device may also arbitrarily Challenge another device simply by generating a Challenge-Request with a random Original Message Id, and any value for the Original Timestamp. When performing a challenge without reference to a specific message, the Message Challenge field shall be set to 0.

Upon receipt of a Challenge-Request that authenticates correctly according to Clause 24.13, with a Message Challenge field set to 1, the device shall attempt to verify that it originated the message identified by Original Message Id. If the device verifies that it did in fact send the message, it shall respond with a Security-Response with a Response Code of Success. If the device is unable to verify that it sent the specified message such as would occur if its Message Id cache were to overflow, then the device shall send a Security-Response with the error code cannotVerifyMessageId. If the device determines that it did not send the message the device shall send a Security-Response with the error code unknownSourceMessage. The device shall also reset any response timer for the challenged security message to Security_PDU_Timeout.

A device shall wait its Security_PDU_Timeout as specified in its Network Security object before cancelling a request due to a lack of response.

Upon receipt of a Challenge-Request that authenticates correctly according to Clause 24.13, with a Message Challenge field set to 0, a device shall respond with a Security-Response with a Response Code of Success.

If the device is unable to generate truly random data for Challenge requests with a Message Challenge field set to 0, the original Message Id and Original Timestamp fields can be set to values not previously used (ever). To do so, the device needs to remember the last used values for these fields across resets.

Broadcasts of this Message Type shall be ignored.

Messages of this type shall be sent with the data_expecting_reply bit set to 1 in the NPCI.

Devices that do not know their own MAC address, such as BACnet/IP devices behind a NAT firewall, may use the Challenge-Request message to determine their own address by examining the DADR in the Security-Response message.

The possible error codes returned in response to a Challenge-Request are listed in Table 24-3 below. For more information on selecting an error code to return, see Clause 24.16.2.

Table 24-3. Challenge-Request Error Codes

Error Code	Ignorable	Description
securityNotConfigured	Yes	If the recipient is not configured for security on this port.
encryptionNotConfigured	Yes	If the Encrypted field is set to 1 and the receiving device is not configured to accept encrypted messages.
unknownKey	Yes	If the Key Identifier field indicates a security key that the receiving device does not know.
duplicateMessage	Yes	A message with the provided Message Id has already been received from the source device within the security time window.
unknownKeyRevision	Yes	If the Key Revision field indicates a revision that the receiving device does not know.
malformedMessage	Yes	If the message size is invalid, or security parameters are missing or malformed.
badSignature	Yes	If the signature is not correct. This error may also be indicated if a decryption error occurs.
badDestinationAddress	Yes	If the destination address information is missing or invalid.

badDestinationDeviceId	Yes	If the Destination Device Instance is not 4194303 and does not match the local device instance.
badSourceAddress	No	If the source address information (SNET/SLEN/SADR) is invalid.
unknownSourceMessage	No	The specified message was not sent by the client. While devices are not required to track all messages that have been sent, if a device is capable of detecting that it did not send the specified message, it shall use this error code to indicate that it was not the source of the challenged message. If the device cannot detect that it did not send the message, it shall not return this error code.
cannotVerifyMessageId	No	The device cannot accurately ascertain whether or not it sent the specified message.
badTimestamp	No	The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver.
destinationDeviceIdRequired	No	If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. How a device determines whether or not it requires the Destination Device Instance to be set correctly in any particular request is a local matter.
encryptionRequired	No	If the encrypted flag is set to 0 and the server's policy requires encryption.
sourceSecurityRequired	No	If the secured-by-router flag is 1 and end-to-end security is required, or the Do-not-decrypt flag is 0 and end-to-end encryption is required for the operation requested.

24.3.2 Security-Payload

The Service Data for a Security-Payload message has the following form:

Table 24-4. Security-Payload Service Data

Message Field	Size	Description
Payload Length	2 octets	The number of octets of payload
Payload	Variable	The secured NSDU

The Security-Payload message is used to transfer non-security related BACnet messages between communicating parties. As with all secure BACnet messages, the message is signed and may be optionally encrypted.

If the recipient of this message cannot process the message for one of the reasons listed below, and if the message was unicast, a negative Security Response may be returned to the sender with a Response Code as shown in the following table. Positive Security-Response messages are not generated in response to a Security-Payload message.

The data_expecting_reply bit in the NPCI is set based on the message in the Payload parameter. If the data_expecting_reply bit in the NPCI is not set, devices are not required to send Security-Responses when reportable errors occur, even if the error condition is not ignorable. If the data_expecting_reply bit is set, then a Security-Response shall be sent if a non-ignorable error condition occurs.

The possible error codes returned in response to a Security-Payload message are listed in Table 24-5 below. The 'Ignorable' column indicates whether the device is allowed to silently fail the request and not report the error condition to the requestor. For more information on selecting an error code to return, see Clause 24.16.2**.** Note that a device may ignore any request from a non-trusted source (non-trusted-source flag set by a router) even if the encountered error is not ignorable.

Table 24-5. Security-Payload Error Codes

Error Code	Ignorable	Description
securityNotConfigured	Yes	If the recipient is not configured for security on this port.
encryptionNotConfigured	Yes	If the Encrypted field is set to 1 and the receiving device is not configured to accept encrypted messages.
unknownKey	Yes	If the Key Identifier field indicates a security key that the receiving device does not know.
duplicateMessage	Yes	A message with the provided Message Id has already been received from the source device within the security time window.
unknownKeyRevision	Yes	If the Key Revision field indicates a revision that the receiving device does not know.
malformedMessage	Yes	If the message size is invalid, or security parameters are missing or malformed.
badSignature	Yes	If the signature is not correct. This error may also be indicated if a decryption error occurs.
badDestinationAddress	Yes	If the destination address information is missing or invalid.
badDestinationDeviceId	Yes	If the Destination Device Instance is not 4194303 and does not match the local device instance.
badSourceAddress	No	If the source address information (SNET/SLEN/SADR) is invalid.
badTimestamp	No	The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver.
destinationDeviceIdRequired	No	If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. How a device determines whether or not it requires the Destination Device Instance to be set correctly in any particular request is a local matter.
encryptionRequired	No	If the Encrypted field is set to 0 and the server's policy requires encryption.
sourceSecurityRequired	No	If the secured-by-router flag is 1 and end-to-end security is required, or the Do-not-

		decrypt flag is 0 and end-to-end encryption is required for the operation requested.
incorrectKey	No	The key provided to secure the message does not indicate sufficient authority to perform the requested operation.
unknownAuthenticationType	No	If the user authentication method in the message is unknown to the device.
accessDenied	No	The network layer or BVLL request was denied due to insufficient authorization. See Clause 24.14.1 for more details.

24.3.3 Security-Response

The Service Data for a Security-Response message has the following form:

Table 24-6. Security-Response Service Data

Message Field	Size	Description
Response Code	1 octet	The type of response (positive acknowledgement or error code).
Original Message Id	4 octets	The Message Id of the message that caused the response.
Original Timestamp	4 octets	The Timestamp of the message that caused the response.
Response Specific Parameters	Variable	The contents of this field are dependent on the value of the Response Code field.

This message is sent as a positive acknowledgement of another security message, or when a security error occurs and the reporting of that error is allowed by the security policy. A Security-Response message is not sent in response to a broadcast message, except by a Key Server in response to a broadcast Request-Key-Update is valid in all aspects, except for the timestamp. Certain errors may be suppressed if hiding from port scanners is desired.

The reaction of the recipient to this message is a local matter. Usually Security-Response messages that represent errors will be ignored, logged, or delivered to a human operator.

No errors shall be returned in response to a Security-Response message.

The Security-Response messages are sent in response to a security message and indicate a success or failure to process the message. All security responses shall be sent with the same level of security (signed or encrypted), with the same Key Identifier that the original message had except as described in Table 24-7. The Source Device Instance shall be equal to the Destination Device Instance of the original request, except if the Destination Device Instance was 4194303.

Table 24-7. Security-Response Security Level Exceptions

Response Code	Security Applied
securityNotConfigured	The Security-Response shall indicate the General-Network-Access key in the header's Key Identifier field, shall contain an all 0 signature, and shall not be encrypted. Clients receiving this error shall optionally report this error to a management entity, and then silently drop it.
encryptionNotConfigured	The Security-Response shall be secured with the General-Network-Access key, and shall not be encrypted.
unknownKeyRevision	The Security-Response shall be secured with the General-Network-Access key, and

	shall not be encrypted. If the local policy requires encryption, then no security response shall be generated.
unknownKey	The Security-Response shall be secured with the General-Network-Access key.

Security-Response messages shall not be sent in response to Security-Response messages.

Broadcasts of this Message Type shall be ignored.

Messages of this type shall be sent with the data_expecting_reply bit set to 0 in the NPCI.

The following list defines the allowable security response codes and indicates which ones are general security error codes and which ones are authorization error codes. For further information on the differentiation of error codes, see Clause 24.16.2.

Table 24-8. Security Response Codes

Value	Response Code	Type
X’00’	success	
X'01'	accessDenied	Authorization
X'02'	badDestinationAddress	General
X’03’	badDestinationDeviceId	General
X'04'	badSignature	General
X'05'	badSourceAddress	General
X'06'	badTimestamp	General
X'07'	cannotUseKey	General
X'08'	cannotVerifyMessageId	General
X'09'	correctKeyRevision	General
X'0A'	destinationDeviceIdRequired	Authorization
X'0B'	duplicateMessage	General
X'0C'	encryptionNotConfigured	General
X'0D'	encryptionRequired	Authorization
X'0E'	incorrectKey	Authorization
X’0F’	invalidKeyData	General
X’10’	keyUpdateInProgress	General
X’11’	malformedMessage	General
X’12’	notKeyServer	General
X'13'	securityNotConfigured	General
X'14'	sourceSecurityRequired	Authorization
X’15’	tooManyKeys	General
X'16'	unknownAuthenticationType	Authorization
X'17'	unknownKey	General
X'18'	unknownKeyRevision	General
X'19'	unknownSourceMessage	General

Descriptions of Response Specific Parameters follow. Response Codes for which no Response Specific Parameters are defined have no Response Specific Parameters and shall be transmitted without a Response Specific Parameters field.

24.3.3.1 badTimestamp

The Response Specific Parameters for a badTimestamp error are:

Table 24-9. badTimestamp Response-Specific Parameters

Message Field	Size	Description
Expected Timestamp	4 octets	The current time of the device that generated the Security-Response message.

Devices that generate a badTimestamp error shall set the Timestamp field in the security header to the value provided in the original message to ensure that it will be accepted by the destination device. This is the only case where the Timestamp field in the security header does not represent the current time in the generating device.

24.3.3.2 cannotUseKey

The Response Specific Parameters for a cannotUseKey error are:

Table 24-10. cannotUseKey Response-Specific Parameters

Message Field	Size	Description
Key	2 octets	The key identifier of the key that the device is incapable of using. If there is more than one key provided in the original request that the device cannot use, the first key encountered that the device cannot use shall be indicated.

24.3.3.3 incorrectKey

The Response Specific Parameters for an incorrectKey error are:

Table 24-11. incorrectKey Response-Specific Parameters

Message Field	Size	Description
Number Of Keys	1 octet	The number of Key Identifiers that follow
List Of Known Keys	n*2 octets	A list of n Key Identifiers that are known to the device.

The List Of Known Keys is populated with each of the Key Identifiers that the device knows. A device may optionally leave out of the List Of Known Keys any keys that the device knows will not grant sufficient access if the failed action is retried.

24.3.3.4 unknownAuthenticationType

The Response Specific Parameters for an unknownAuthenticationType error are:

Table 24-12. unknownAuthenticationType Response-Specific Parameters

Message Field	Size	Description
Original Authentication Mechanism	1 octet	The Authentication Mechanism from the original request.
Vendor Id	2 octets	If the value of the Original Authentication Mechanism is 200 through 255, then this shall be set to the Vendor Id provided with the Authentication Mechanism in the original message, otherwise this field shall be 0.

24.3.3.5 unknownKey

The Response Specific Parameters for a unknownKey error are:

Table 24-13. unknownKey Response-Specific Parameters

Message Field	Size	Description
Original Key	2 octets	The Key Identifier of the unknown key.

24.3.3.6 unknownKeyRevision

The Response Specific Parameters for an unknownKeyRevision error are:

Table 24-14. unknownKeyRevision Response-Specific Parameters

Message Field	Size	Description
Original Key Revision	1 octet	The Key revision that is unknown or otherwise invalid.

24.3.3.7 tooManyKeys

The Response Specific Parameters for a tooManyKeys error are:

Table 24-15. tooManyKeys Response-Specific Parameters

Message Field	Size	Description
Maximum Number Of Keys	1 octet	The maximum number of keys that this device is capable of be configured with.

24.3.3.8 invalidKeyData

The Response Specific Parameters for a invalidKeyData error are:

Table 24-16. invalidKeyData Response-Specific Parameters

Message Field	Size	Description
Key	2 octets	The Key Identifier of the key that contains invalid key data.

24.3.4 Request-Key-Update

The Service Data for a Request-Key-Update message has the following form:

Table 24-17. Request-Key-Update Service Data

Message Field	Size	Description
Set 1 Key Revision	1 octet	The Key Revision of the device's first key set.
Set 1 Activation Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set becomes valid, before which the device shall not accept or generate messages with keys from the set. A value of 0 shall indicate that the key set is valid immediately.
Set 1 Expiration Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set expires after which the device shall no longer accept or generate messages with keys from the set. An indefinite expiration time is encoded as X'FFFFFFFF'.
Set 2 Key Revision	1 octet	The Key Revision of the device's second key set.

Set 2 Activation Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set becomes valid before which the device shall not accept or generate messages with keys from the set. A value of 0 shall indicate that the key set is valid immediately.
Set 2 Expiration Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set expires after which the device shall no longer accept or generate messages with keys from the set. An indefinite expiration time is encoded as X'FFFFFFFF'.
Distribution Key Revision	1 octet	The revision for the Distribution key.

This security message is used by secure devices that either do not have a valid key set, or want to ensure that both of the device's key sets are still the most current.

If a secure device does not have a valid Distribution key, it shall secure this message with its Device-Master key thus indicating to the Key Server that a Distribution key is required. If a key set has not been received, the revision number field shall be 0. In such cases the time fields are meaningless and are ignored by the Key Server.

Upon receipt of a valid Request-Key-Update message secured with the device's Device-Master key, a Key Server device shall respond to the device with an Update-Distribution-Key message and then send an Update-Key-Set message to the device.

Upon receipt of a valid Request-Key-Update message secured with the device's Distribution key, a Key Server shall respond to the device with a Security-Response message with a Response Code of correctKeyRevision if the key revision data provided are the same as the Key Server has recorded for the device and the Key Server does not have an outstanding set of keys to send to the device. Otherwise the Key Server shall respond to the device with an Update-Key-Set message.

Key Servers shall not restrict execution of this service based on the Authentication Mechanism.

A device shall wait its Security_PDU_Timeout as specified in its Network Security object before cancelling a request due to a lack of response.

A device that receives no response, or receives an error of unknownKeyRevision in response to a Request-Key-Update secured with a Distribution key, should retry the request secured with the device's Device-Master key. This allows the device to obtain a new Distribution key and key set in those cases where the device and the Key Server have become out of sync.

Broadcasts of this Message Type shall be allowed. Non-key Server devices shall ignore broadcasts of this message.

Unicast messages of this type shall have the data_expecting_reply bit set to 1 in the NPCI.

The possible error codes returned in response to a Request-Key-Update are listed in Table 24-18 below. The 'Ignorable' column indicates whether the device is allowed to silently fail the request and not report the error condition to the requestor. For more information on selecting an error code to return, see Clause 24.16.2. When executing this service, incorrectKey shall be considered a general security error and not an authorization error.

Table 24-18. Request-Key-Update Error Codes

Error Code	Ignorable	Description
securityNotConfigured	Yes	If the recipient is not configured for security on this port

encryptionNotConfigured	Yes	If the Encrypted field is set to 1 and the server is not configured to accept encrypted messages.
incorrectKey	Yes	If the request is not secured with a Distribution key or Device-Master key.
duplicateMessage	Yes	A message with the provided Message Id has already been received from the source device within the security time window.
unknownKeyRevision	Yes	If the Key Revision field indicates a revision that the receiving device does not know.
malformedMessage	Yes	If the message size is invalid, or security parameters are missing or malformed.
badSignature	Yes	If the signature is not correct. This error may also be indicated if a decryption error occurs.
badDestinationAddress	Yes	If the destination address information is missing or invalid.
badDestinationDeviceId	Yes	If the Destination Device Instance is not 4194303 and does not match the local device instance.
badSourceAddress	No	If the source address information (SNET/SLEN/SADR) is invalid.
badTimestamp	No	The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver.
destinationDeviceIdRequired	No	If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. How a device determines whether or not it requires the Destination Device Instance to be set correctly in any particular request is a local matter.
encryptionRequired	No	If the Encrypted field is set to 0 and the server's policy requires encryption.
notKeyServer	No	If the message was unicast and the receiving device is not configured as a Key Server for the requesting device.
correctKeyRevision	No	The device's current keys are valid and no updates for the device are pending.

24.3.5 Update-Key-Set

The Service Data for an Update-Key-Set message has the following form:

Table 24-19. Update-Key-Set Service Data

Message Field	Size	Description
Control Flags	1 octet	Control flags for the Update-Key-Set message.
Set 1 Key Revision	1 octet	The Key Revision of the device's first key set.
Set 1 Activation Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which the device is allowed to start using the first key set. A value of 0 shall

		indicate that the key set is valid immediately.
Set 1 Expiration Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set expires after which the device shall no longer accept or generate message with keys from the set. An indefinite expiration time is encoded as X'FFFFFFFF'.
Set 1 Key Count	1 octet	The number of keys in the first key set.
Set 1 Keys	Variable	The first key set, consisting of a concatenated sequence of key entries.
Set 2 Key Revision	1 octet	The Key Revision of the device's second key set.
Set 2 Activation Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which the device is allowed to start using the second key set. A value of 0 shall indicate that the key set is valid immediately.
Set 2 Expiration Time	4 octets	The UTC time, in seconds since 12:00 AM January 1, 1970, at which this key set expires after which the device shall no longer accept or generate message with keys from the set. An indefinite expiration time is encoded as X'FFFFFFFF'.
Set 2 Key Count	1 octet	The number of keys in the second key set.
Set 2 Keys	Variable	The second key set, consisting of a concatenated sequence of key entries.

This security message is used to provide keys to secure devices. This message shall always be signed and encrypted with the destination device's Distribution key.

The message can be used to provide a new key set, to modify an existing key set, or to invalidate an existing key set.

The Control Flags parameter is 1 octet containing a number of control flags as follows:

Bit 7: 1 indicates that Set 1 Key Revision, Set 1 Activation Time and Set 1 Expiration Time parameters are present.
0 indicates that those parameters are not present.

Bit 6: 1 indicates that Set 1 Key Count and Set 1 Keys parameters are present.
0 indicates that those parameters are not present.

Bit 5: 1 indicates that Key Set 1 should be cleared before any updates are applied.
0 indicates that Key Set 1 should not be cleared before updates are applied.

Bit 4: 1 indicates that Set 2 Key Revision, Set 2 Activation Time and Set 2 Expiration Time parameters are present.
0 indicates that those parameters are not present.

Bit 3: 1 indicates that Set 2 Key Count and Set 2 Keys parameters are present.
0 indicates that those parameters are not present.

Bit 2: 1 indicates that Key Set 2 should be cleared before any updates are applied.
0 indicates that Key Set 2 should not be cleared before updates are applied.

Bit 1: 1 indicates more messages are expected in this sequence of update messages.
0 indicates that this is the final message in this sequence of update messages.

Bit 0: 1 indicates that the keys provided in the message shall be removed from the key sets.

0 indicates that the keys provided in the message shall be added to the key sets (or update the keys if they already exist in the key set).

Upon receipt of a valid Update-Key-Set message signed and encrypted with the device's Distribution key, the device shall apply the changes to its key set(s).

If key revision, activation time and expiration time are included in the message, they shall replace the key revision, activation time and expiration time values for the appropriate key set.

If keys are provided in the Update-Key-Set message, and Bit 0 indicates that keys shall be added or updated, the keys provided in the message shall be added to the appropriate key set if they do not already exist in the key set. If a key with the same Key Identifier already exists in the key set, then the value for the key shall be replaced with the new key value.

If keys are provided in the Update-Key-Set message, and Bit 0 indicates that keys shall be removed, any key with the matching Key Identifiers in the appropriate key set shall be removed. If a specified Key Identifier does not exist in the key set, the request shall succeed as if it had existed.

When Bit 0 is set, the Key Size for each key provided may be 0.

A device may optionally apply all changes to shadow key sets and only update the actual key sets when an Update-Key-Set message is received that has Bit 1 of the Control Flags parameter set to 0. As such, it is a local matter as to whether or not the modifications to the key sets take effect immediately, or when the final Update-Key-Set message is received.

When replacing key sets, a sequence of 1 or more Update-Key-Set messages is sent to the device. The first message in the sequence shall have bit 5 and/or bit 2 set to 1 to cause the existing key sets to be cleared. The final message in the sequence shall have bit 1 set to 0, and all other messages in the sequence shall have bit 1 set to 1. If the key set replacement can be described in a single message, then that message shall have bit 5 and/or bit 2 set to 1, and bit 1 set to 0.

This message can also be used to add, update or remove one or more keys from one or both of the key sets without replacing the complete key sets. In such cases, the initial message shall not have bit 5 and bit 2 of the Control Flags parameter set to 1.

When issuing multiple Update-Key-Set messages to a single device, the requesting device shall wait until a Security-Response message is received before issuing another Update-Key-Set message to the device. As these messages can take longer than normal to process, the requesting device shall wait at least Update_Key_Set_Timeout + APDU_Timeout as set in the destination device's Network Security and Device objects for a Security-Response from the device.

In general, when replacing key sets, one of the messages in the sequence shall include the key revisions, activation and expiration times; the same message need not provide these values for both key sets. The key revisions, activation and expiration times can also be updated in a device without modifying the key sets.

Update-Key-Set messages shall be executed in order, and the keys shall be applied in the order found in the Update-Key-Set message. If there are duplicate revisions, expiration dates, or key values, then the last value shall take precedence.

If a key cannot be added to the device's key set, because the local key set table is full, or the key is not usable by the device, the keys preceding the problem key in the message shall be added, and the other keys shall not be added.

Upon executing the first of a sequence of Update-Key-Set messages, a device shall record the address of the Key Server in the Last_Key_Server property of the Network Security object. The device shall only accept subsequent Update-Key-Set messages in the sequence from the same Key Server. Well formed Update-Key-Set messages from other Key Servers shall result in a keyUpdateInProgress.

If a device is waiting for more Update-Key-Set messages and none are received in 5 * (Update_Key_Set_Timeout + APDU_Timeout), the device shall consider the sequence of Update-Key-Set messages complete. It is a local matter as to whether the device updates its key sets based on the partial sequence it received, or whether it drops all changes. When a Key Server is unable to complete the sequence of updates, it shall retry the complete sequence of updates at a later time.

Devices shall not restrict execution of this service based on the authentication mechanism; knowledge of the device's Distribution key shall always be sufficient authorization.

Each key entry in the message shall be of the form:

Table 24-20. Key Entry Description

Message Field	Size	Description
Key Identifier	2 octets	The Key Identifier for the key pairs
Key Size	1 octet	The size of the key, in octets.
Key	Variable	The key value, consisting of the signature key followed by the encryption key.

The correct key sizes by algorithm are:

Table 24-21. Key Sizes by Algorithm

Hash algorithm (key size bytes)	Encryption algorithm (key size bytes)	Key field size
MD5 (16)	AES (16)	32 octets
SHA-256 (32)	AES (16)	48 octets

Broadcasts of this Message Type shall be ignored.

Messages of this type shall have the data_expecting_reply bit set to 1 in the NPCI.

The possible error codes returned in response to an Update-Key-Set are listed in Table 24-22 below. The 'Ignorable' column indicates whether the device is allowed to silently fail the request and not report the error condition to the requestor. For more information on selecting an error code to return, see Clause 24.16.2.

Table 24-22. Update-Key-Set Error Codes

Error Code	Ignorable	Description
securityNotConfigured	Yes	If the recipient is not configured for security on this port.
incorrectKey	Yes	If the request is not secured with a Distribution key.
duplicateMessage	Yes	A message with the provided Message Id has already been received from the source device within the security time window.
unknownKeyRevision	Yes	If the Key Revision field indicates a revision that the receiving device does not know.
malformedMessage	Yes	If the message size is invalid, or security parameters are missing or malformed.
badSignature	Yes	If the signature is not correct. This error may also be indicated if a decryption error occurs.
badDestinationAddress	Yes	If the destination address information is missing or invalid.
badDestinationDeviceId	Yes	If the Destination Device Instance is not 4194303 and does not match the local

		device instance.
badSourceAddress	No	If the source address information (SNET/SLEN/SADR) is invalid.
badTimestamp	No	The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver.
destinationDeviceIdRequired	No	If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. How a device determines whether or not it requires the Destination Device Instance to be set correctly in any particular request is a local matter.
encryptionRequired	No	If the Encrypted field is set to 0.
sourceSecurityRequired	No	If the secured-by-router flag is 1, or the Do-not-decrypt flag is 0.
keyUpdateInProgress	No	If an Update-Key-Set message is received from a different Key Server while waiting for more Update-Key-Set messages.
cannotUseKey	No	If the encryption or signature algorithm of any key provided in the key sets is based on an algorithm that the device does not support.
tooManyKeys	No	If the device cannot be configured with the number of keys provided for the key set.
invalidKeyData	No	If the key data provided for one of the keys does not match the size required for the specified algorithms.

24.3.6 Update-Distribution-Key

The Service Data for a Update-Distribution-Key message has the following form:

Table 24-23. Update-Distribution-Key Service Data

Message Field	Size	Description
Key Revision	1 octet	The Key Revision of the device's Distribution key.
Key	Variable	The new Distribution key.

This security message is used by the Key Server to provide Distribution keys to secure devices. This message shall always be signed and encrypted with the destination device's Device-Master key. The Key Revision field of the security header shall be ignored by the destination device (no revision is associated with the Device-Master Key).

This message is sent either to initially configure a Distribution key into a new device, to update a Distribution key in an existing device, or to provide a Distribution key at the request of a device.

Upon receiving a valid Update-Distribution-Key message signed and encrypted with the device's Device-Master key, a device shall replace its Distribution key and respond with a Security-Response message containing a Response Code of Success.

Devices shall not restrict execution of this service based on the authentication mechanism; knowledge of the device's Device-Master key shall always be sufficient authorization.

The Key field in the message shall be of the form specified in Table 24-20. The correct key sizes by algorithm are as given in Table 24-21.

As these messages can take longer than normal to process, the requesting device shall wait at least Update_Key_Set_Timeout + APDU_Timeout as set in the destination device's Network Security and Device objects for a Security-Response from the device.

Broadcasts of this Message Type shall be ignored.

Messages of this type shall have the data_expecting_reply bit set to 1 in the NPCI.

The possible error codes returned in response to an Update-Distribution-Key are listed in Table 24-24 below. The 'Ignorable' column indicates whether the device is allowed to silently fail the request and not report the error condition to the requestor. For more information on selecting an error code to return, see Clause 24.16.2**.**

Table 24-24. Update-Distribution-Key Error Codes

Error Code	Ignorable	Description
securityNotConfigured	Yes	If the recipient is not configured for security on this port.
incorrectKey	Yes	If the request is not secured with a Device-Master key.
duplicateMessage	Yes	A message with the provided Message Id has already been received from the source device within the security time window.
malformedMessage	Yes	If the message size is invalid, or security parameters are missing or malformed.
badSignature	Yes	If the signature is not correct. This error may also be indicated if a decryption error occurs.
badDestinationAddress	Yes	If the destination address information is missing or invalid.
badDestinationDeviceId	Yes	If the Destination Device Instance is not 4194303 and does not match the local device instance.
badSourceAddress	No	If the source address information (SNET/SLEN/SADR) is invalid.
badTimestamp	No	The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver.
destinationDeviceIdRequired	No	If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. How a device determines whether or not it requires the Destination Device Instance to be set correctly in any particular request is a local matter.
encryptionRequired	No	If the Encrypted field is set to 0.
sourceSecurityRequired	No	If the secured-by-router flag is 1, or the Do-not-decrypt flag is 0.
cannotUseKey	No	If the encryption or signature algorithm of

		the distribution key provided is based on an algorithm that the device does not support.
invalidKeyData	No	If the key data provided for one of the keys does not match the size required for the specified algorithms.

24.3.7 Request-Master-Key

The Service Data for a Request-Master-Key message has the following form:

Table 24-25. Request-Master-Key Service Data

Message Field	Size	Description
Number of Supported Key Algorithms	1 octet	The number of encryption/signature algorithm pairs that follow.
Encryption and Signature Algorithms	Variable	Lists the encryption algorithms that device supports. Each algorithm is encoded in 1 octet as per Table 24-30

This security message is generated by a secure device to request a Device-Master key from a Key Server. It is expected that this message is only used when initially configuring a secure device to work with the Key Server and that the issuance of the request is the result of a physical interaction with the device. If a secure device times out waiting for a Set-Master-Key message, the length of the timeout shall be sufficient to allow for human interaction with the Key Server in order to allow the Key Server to respond to the request. This message is usually globally broadcast, but it may be unicast, or sent as a directed broadcast in order to get the message through legacy routers.

As the secure device cannot produce a signature, the message shall not include a valid signature (the signature shall be all zeros) nor be encrypted. The Key Identifier field shall be set to 0. This service might not work if either the Key Server or the destination device is connected to a network that requires encryption. Secure routers shall, by default, not drop these messages regardless of the network security policies although secure routers are allowed to be configured to drop these packets.

This service is inherently insecure and as such its use must be protected through safety precautions taken both by the implementers of secure BACnet devices and site setup personnel. The expectation is that site policy will dictate whether this service is to be used on a physically secure network, such as would be accomplished by attaching a BACnet device directly to the Key Server via a port dedicated for this purpose, or whether this service is allowed to be used on insecure networks during site setup.

A Key Server that receives this message, and is in a mode that allows it to respond to Request-Master-Key messages, or is instructed to respond by a user, shall record the algorithms that the device supports, and then generate a Set-Master-Key message in response. If the device does not support any signature or encryption algorithms allowed for use on the site, the Key Server shall report the deficiency to the user. The expectation is that a Key Server will only be in a mode that allows a response to this message if the Key Server has been specifically placed into such a mode. The method for placing a Key Server into a mode that will support execution of the Set-Master-Key service is a local matter. Key Servers are required to support such a mode.

Secure BACnet devices that are not configured as Key Servers, and Key Servers that are not in a mode that allows a response shall silently drop this message.

Messages of this type shall have the data_expecting_reply bit set to 0 in the NPCI.

24.3.8 Set-Master-Key

The Service Data for a Set-Master-Key message has the following form:

Table 24-26. Set-Master-Key Service Data

Message Field	Size	Description

Key	variable	The Device-Master key.

This security message is sent by a Key Server in response to a Request-Master-Key message. This message shall not be encrypted by the source device. The message shall be signed with the Device-Master key thus allowing values for all of security header fields.

Secure BACnet devices shall ignore messages of this type unless they are specifically placed into a mode in which they will accept these messages. It is recommended that secure BACnet devices restore themselves to factory defaults, excluding network communication parameters, when this service is executed in order to protect against theft of confidential information.

When a device receives this message, and this device has requested and is waiting for, a Device-Master key, the device shall accept the key from the message, and respond with a Security-Response with an error code of Success. In the case of failure, the device shall silently drop the request as it cannot secure a negative response and thus the response would not be able to be transmitted through the secure network to the Key Server.

See the Request-Master-Key description in Clause 24.3.7 for more details on the use of this service.

The Key field in the message shall be of the form specified in Table 24-20. The correct key sizes by algorithm are as given in Table 24-21.

Broadcasts of this Message Type shall be ignored.

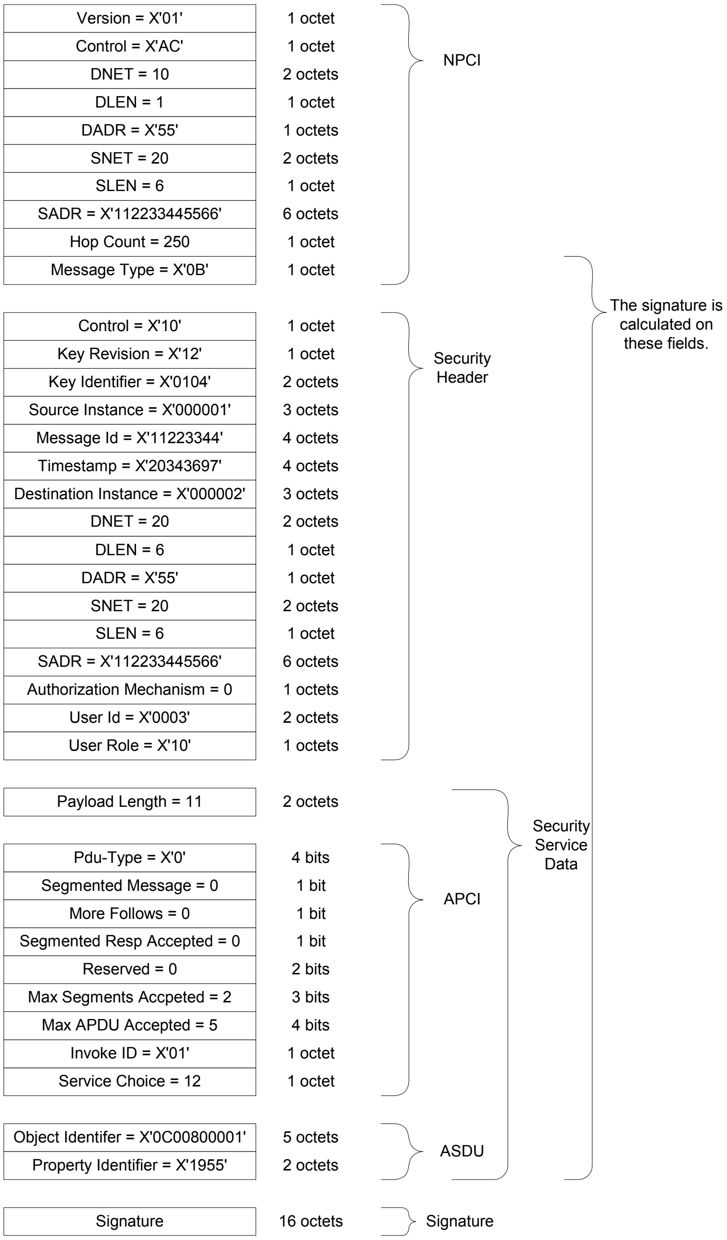
Messages of this type shall have the data_expecting_reply bit set to 1 in the NPCI.

24.4 Securing an APDU

When an APDU is to be sent securely, the APDU portion of the message is placed into the Payload parameter of a Security-Payload message.

Security is applied at the network layer by creating a new NPDU message type. Therefore, when a BACnet APDU is encapsulated with security information, it is transported as a network layer message so the control bit in the NPCI is changed to indicate that the message now contains a network layer message rather than an APDU. The security header will indicate that the encapsulated message is an APDU so that this information is not lost. Upon unwrapping this message, this control bit will change back so that the resulting NPDU will once again indicate that it contains an APDU.

An example of a secured APDU follows.

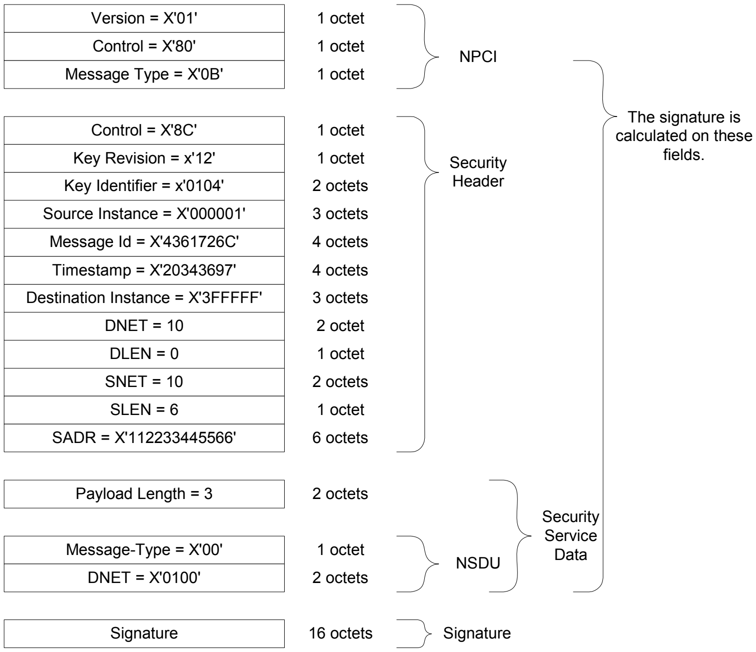


Version = X'01'	1 octet
Control = X'AC'	1 octet
DNET = 10	2 octets
DLEN = 1	1 octet
DADR = X'55'	1 octets
SNET = 20	2 octets
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Hop Count = 250	1 octet
Message Type = X'0B'	1 octet
Control = X'10'	1 octet
Key Revision = X'12'	1 octet
Key Identifier = X'0104'	2 octets
Source Instance = X'000001'	3 octets
Message Id = X'11223344'	4 octets
Timestamp = X'20343697'	4 octets
Destination Instance = X'000002'	3 octets
DNET = 20	2 octets
DLEN = 6	1 octet
DADR = X'55'	1 octet
SNET = 20	2 octets
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Authorization Mechanism = 0	1 octets
User Id = X'0003'	2 octets
User Role = X'10'	1 octets
Payload Length = 11	2 octets
Pdu-Type = X'0'	4 bits
Segmented Message = 0	1 bit
More Follows = 0	1 bit
Segmented Resp Accepted = 0	1 bit
Reserved = 0	2 bits
Max Segments Accpeted = 2	3 bits
Max APDU Accepted = 5	4 bits
Invoke ID = X'01'	1 octet
Service Choice = 12	1 octet
Object Identifer = X'0C00800001'	5 octets
Property Identifier = X'1955'	2 octets
Signature	16 octets

Figure 24-1. An example secured APDU (Read-Property).

24.5 Securing an NPDU

To secure an NPDU, the NSDU (NPDU payload starting with the Message-Type field) shall be placed into the Payload field of a Security-Payload message.



Version = X'01'	1 octet
Control = X'80'	1 octet
Message Type = X'0B'	1 octet
Control = X'8C'	1 octet
Key Revision = x'12'	1 octet
Key Identifier = x'0104'	2 octets
Source Instance = X'000001'	3 octets
Message Id = X'4361726C'	4 octets
Timestamp = X'20343697'	4 octets
Destination Instance = X'3FFFFF'	3 octets
DNET = 10	2 octet
DLEN = 0	1 octet
SNET = 10	2 octets
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Payload Length = 3	2 octets
Message-Type = X'00'	1 octet
DNET = X'0100'	2 octets
Signature	16 octets

Figure 24-2. An example secured NPDU (Who-Is-Router-To-Network).

24.6 Securing a BVLL

BVLLs are divided into 2 groups: those that contain an NPDU, such as Original-Unicast-NPDU or Distribute-Broadcast-To-Network, and those that do not, such as Register-Foreign-Device or BVLL-Result.

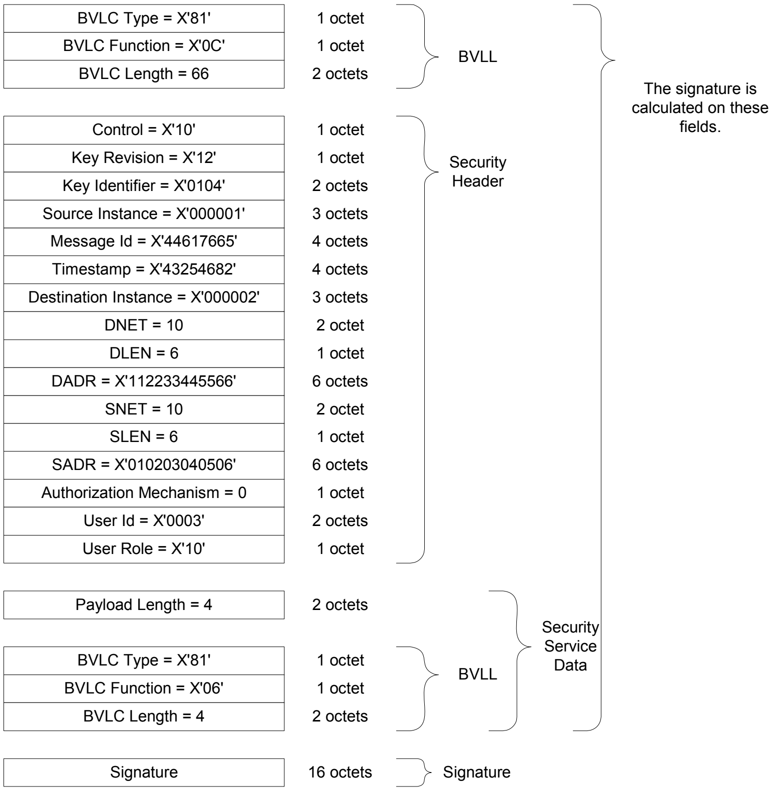
To secure a BVLL message that does not contain an NPDU, the original BVLL shall be placed in the Service Data field of a Security Wrapper, and that Security Wrapper shall be used as the service data for a Secure-BVLL (BVLC Function X'0C'), message, as shown in Figure 24-3.

The ability to generate and consume Security Wrappers requires that the sender and receiver of secured BVLL messages are full BACnet devices with all the requirements thereof.

The Secure-BVLL message consists of the following fields:

Table 24-27. Secure-BVLL Message Fields

Field	Size	Description
BVLC Type	1-octet	BVLL for BACnet/IP (value = X'81')
BVLC Function	1-octet	Secure-BVLL (value = X'0C')
BVLC Length	2-octets	Length L, in octets, of the Secure-BVLL message and its contents up to and including the Signature. It includes the padding if the Secure-BVLL is encrypted.
Security Wrapper	variable	As described in the Security Wrapper clause.



BVLC Type = X'81'	1 octet
BVLC Function = X'0C'	1 octet
BVLC Length = 66	2 octets
Control = X'10'	1 octet
Key Revision = X'12'	1 octet
Key Identifier = X'0104'	2 octets
Source Instance = X'000001'	3 octets
Message Id = X'44617665'	4 octets
Timestamp = X'43254682'	4 octets
Destination Instance = X'000002'	3 octets
DNET = 10	2 octet
DLEN = 6	1 octet
DADR = X'112233445566'	6 octets
SNET = 10	2 octet
SLEN = 6	1 octet
SADR = X'010203040506'	6 octets
Authorization Mechanism = 0	1 octet
User Id = X'0003'	2 octets
User Role = X'10'	1 octet
Payload Length = 4	2 octets
BVLC Type = X'81'	1 octet
BVLC Function = X'06'	1 octet
BVLC Length = 4	2 octets
Signature	16 octets

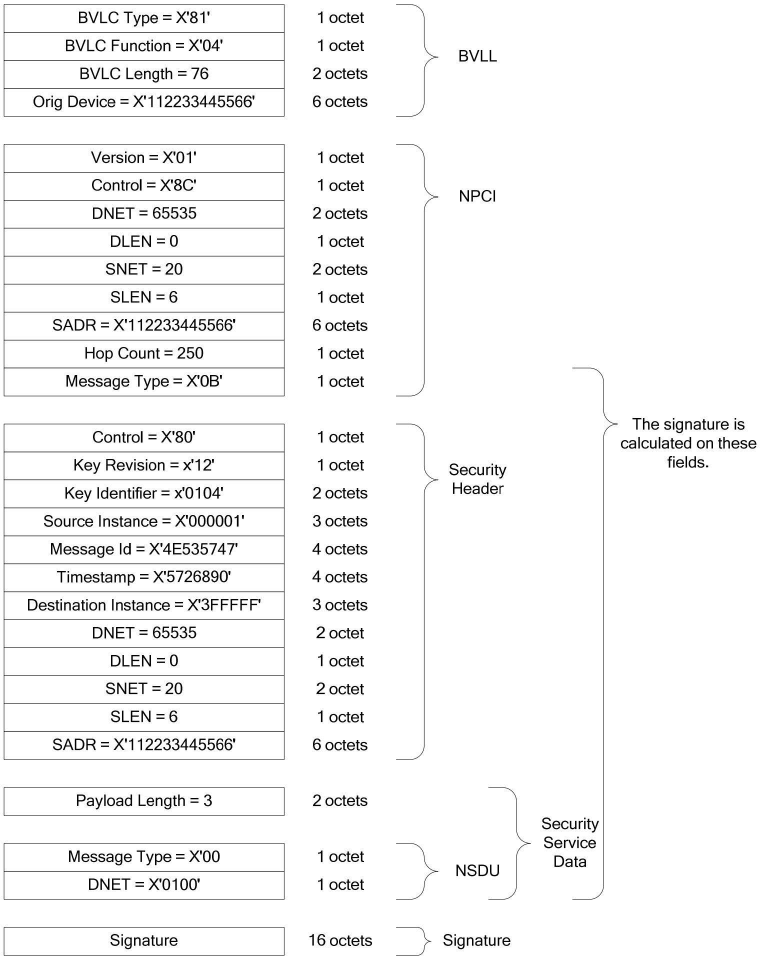
Figure 24-3. An example secured BVLL (Read-Foreign-Device-Table).

BVLL messages that contain an NPDU are transmitted without modification as specified in Annex J. However, there is no loss of security capability because those NPDUs may be secure NPDUs, containing their own Security Wrappers, based on the security policies of the network and sending device. If plain (non-secured) NPDUs are not desired in BVLL messages, then the network security policy of the BACnet/IP network should not be plain.

BBMDs do not perform wrapping/unwrapping functions on forwarded NPDUs the way routers do. The Network Security Policy that guides such operations in routers applies to an entire BACnet network, and BBMDs only serve to facilitate broadcasts between distant parts of a single network, which is governed by a single network security policy.

The possible error codes returned in response to a Secure-BVLL message are the same as for the Security-Payload message and are listed in Table 24-5. The 'Ignorable' column indicates whether the device is allowed to silently fail the request and not report the error condition to the requestor. For more information on selecting an error code to return, see Clause 24.16.2**.**

An example of a BVLL message containing a secure NPDU is shown in Figure 24-4.



BVLC Type = X'81'	1 octet
BVLC Function = X'04'	1 octet
BVLC Length = 76	2 octets
Orig Device = X'112233445566'	6 octets
Version = X'01'	1 octet
Control = X'8C'	1 octet
DNET = 65535	2 octets
DLEN = 0	1 octet
SNET = 20	2 octets
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Hop Count = 250	1 octet
Message Type = X'0B'	1 octet
Control = X'80'	1 octet
Key Revision = x'12'	1 octet
Key Identifier = x'0104'	2 octets
Source Instance = X'000001'	3 octets
Message Id = X'4E535747'	4 octets
Timestamp = X'5726890'	4 octets
Destination Instance = X'3FFFFF'	3 octets
DNET = 65535	2 octet
DLEN = 0	1 octet
SNET = 20	2 octet
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Payload Length = 3	2 octets
Message Type = X'00	1 octet
DNET = X'0100'	1 octet
Signature	16 octets

Figure 24-4. An example BVLL containing a Secured NPDU (Who-Is-Router-To-Network).

24.7 Securing Messages

The basic level of security that can be applied to a BACnet message consists of signing each message with an HMAC, and of marking each message with the source and destination device instances, a Message Id and a timestamp.

24.7.1 Message Id

The security header contains a monotonically increasing 32 bit Message Id which fulfills three purposes in securing BACnet messages. It is used to detect the replay of messages, to associate security responses with security requests and, along with the Timestamp field, to provide variability in otherwise identical messages.

To thwart replay, the Message Id is required to be unique across the security time window. This allows devices to track all received Message Id and Timestamp pairs over the security time window in order to detect replaying of messages.

To allow for a more compact Message Id cache, the Message Id is required to be monotonically increasing across the security time window.

24.7.2 Timestamp

The timestamp in the security header is used to detect the replay of messages.

The timestamp of a received message is compared to the device's local clock. If the timestamp is not within the security time window, then it is not accepted.

Timestamp along with Message Id provide variability in the message content so that messages that are repeated frequently do not generate the same signature.

24.7.3 Device Identification

All secure messages include the Source Device Instance, Destination Device Instance, SNET, SLEN, SADR, DNET, DLEN, and DADR fields in the security header. The inclusion of these values allows the security signature to be calculated on the source and destination device identities in order to stop redirection and identity switching attacks.

Requiring SNET in every secure message imposes the requirement that all secure BACnet devices know their network numbers.

When the device that applies the security wrapper does not know the device instance of the destination device, and attempts to determine the value fail, or the protection provided by the device instance is not required for the operation requested, the value 4194303 shall be used.

When replying to a secure request, the Destination Device Instance can be set to the Source Device Instance of the original request except when the secured-by-router flag is true. In this case the Source Device Instance identifies the router that applied the security, not the device that originated the message.

24.7.4 Message Signature

The signature included in all secure messages is computed using HMAC and a secure hash algorithm.

24.7.4.1 Secure Hash Algorithms

The first step in generating the signature is to generate a hash of the message using the secure hash algorithm specified by the security key. The hash shall be computed over the message contents starting with the octet before the security header (the Message Type field from the containing NPCI) and ending with the last octet of the Service Data field. For Secure-BVLL messages the hash shall be computed over the message contents starting with the first octet of the BVLCI (BVLC Type) and ending with the last octet of the Service Data field.

The signature is always calculated before encryption or after decryption. Before calculating the signature, the encrypted flag of the control octet must be set to 0. Any padding required for encryption is not included in the input text of the signature.

A hash algorithm is considered "secure" because it is computationally infeasible either to find a message that corresponds to a given message digest, or to find two different messages that produce the same message digest. Any change to a message will, with a very high probability, result in a different message digest and thus verification failure.

The output of the secure hash algorithm will be used as the input to one of the keyed-hash message authentication code (HMAC) algorithms that follows.

24.7.4.1.1 MD5

The MD5 hash algorithm is defined by RFC1321. MD5 is included in the BACnet security framework as it is less computationally expensive than SHA-256. Sites whose security policies do not require SHA-256 may get better performance by using MD5. All secure BACnet devices shall support MD5 for use as a secure hash algorithm.

24.7.4.1.2 SHA-256

The SHA-256 hash algorithm is defined in NIST FIPS180-2. This algorithm is more computationally intensive than MD5, but is included in the BACnet security framework for use at sites that require it. All secure BACnet devices shall support SHA-256 for use as a secure hash algorithm.

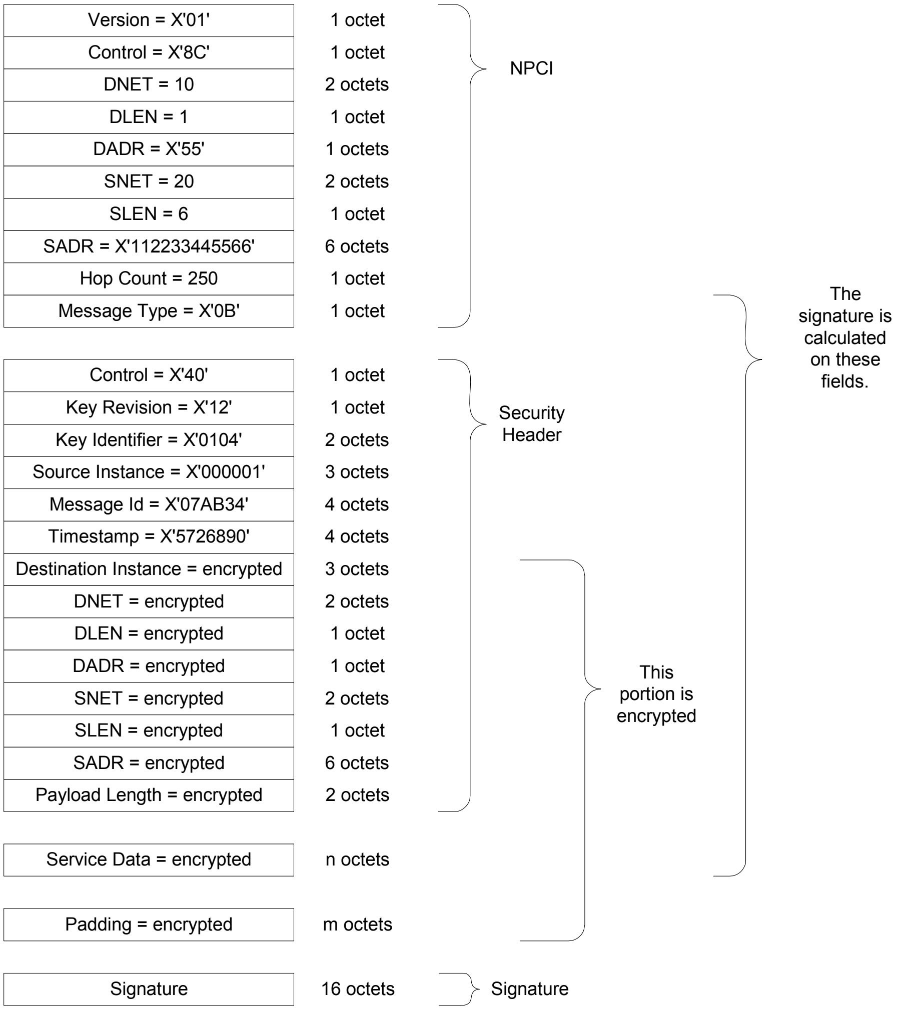
24.7.4.2 HMAC

The HMAC algorithm is defined in section 5 of NIST FIPS198a. The leftmost 16 bytes of the HMAC output shall be used as the signature in secure BACnet messages.

24.7.5 Encrypted Messages

The BACnet security architecture allows for the encryption of a message's payload. The encryption of a BACnet message is done using AES. The definition of AES is contained in NIST FIPS 197.

When encrypting a BACnet message, the text to be encrypted starts with the Destination Device Instance and includes all fields in the security wrapper up to and including the Padding field.



Version = X'01'	1 octet
Control = X'8C'	1 octet
DNET = 10	2 octets
DLEN = 1	1 octet
DADR = X'55'	1 octets
SNET = 20	2 octets
SLEN = 6	1 octet
SADR = X'112233445566'	6 octets
Hop Count = 250	1 octet
Message Type = X'0B'	1 octet
Control = X'40'	1 octet
Key Revision = X'12'	1 octet
Key Identifier = X'0104'	2 octets
Source Instance = X'000001'	3 octets
Message Id = X'07AB34'	4 octets
Timestamp = X'5726890'	4 octets
Destination Instance = encrypted	3 octets
DNET = encrypted	2 octets
DLEN = encrypted	1 octet
DADR = encrypted	1 octet
SNET = encrypted	2 octets
SLEN = encrypted	1 octet
SADR = encrypted	6 octets
Payload Length = encrypted	2 octets
Service Data = encrypted	n octets
Padding = encrypted	m octets
Signature	16 octets

NPCI
Security Header
This portion is encrypted
The signature is calculated on these fields.
Signature

Figure 24-5. An example encrypted message.

24.7.5.1 Cipher Block Chaining (CBC)

In order to encrypt a complete BACnet message, cipher block chaining (CBC) mode is used. The standard formula for CBC is:

$$C_i = E(K, P_i \oplus C_{i-1}) \qquad \text{for } i = 1,\dots,k$$

where

C_0 is the initialization vector,
C_i is the ith block of output,

K is the encryption key,
P_i is the i^{th} block of the portion of the message to encrypt.
$\oplus$ is the XOR operation.
E is the encryption function.
k is the number of blocks that make up the input message.

The initialization vector consists of the first *B* bytes of the message signature where *B* is the block size. For AES the block size is 16. If *B* is larger than the size of the signature, then the signature shall be repeated a sufficient number of times to be larger than or equal to *B*.

24.8 Network Security Network Trust Levels

There are two trust levels that describe the capabilities and security requirements of a device.

24.8.1 Trusted Networks

A trusted network consists of devices that are trusted either inherently, or because all communications are secured by the protocol.

When devices are inherently trusted, access to the devices and networks must be controlled. Depending on the installation policy, this may mean that there are no PCs and there are no direct network connections outside of locked space, or it may be that all devices are in locked cabinets and all network cabling is in metal conduit. The exact requirements will be determined on an installation by installation basis. Non-secured messages exchanged on these networks are trusted.

24.8.2 Non-trusted Networks

A non-trusted network is one where access to the network is not controlled, and as such no non-secured messages exchanged on the network can be trusted. Non-secured messages received from non-trusted networks must not be trusted. In order to identify such messages when they are routed onto trusted networks, the non-trusted-source flag is set to 1. This allows devices to identify and optionally ignore or drop messages from non-trusted networks.

Secure BACnet routers shall be configurable to route non-secured messages from non-trusted networks onto trusted networks. The mechanism for enabling and disabling this feature is a local matter.

24.9 Network Security Policies

There are four standard network security policy levels. The following definitions are presented in increasing order of security policy level.

The policy level for a network specifies the minimum level of security required for messages on the network. It also specifies the default level of security for messages that are forwarded onto a network. A secure router will change the security on a message when it is forwarded onto a network with a different security policy if not restricted by the do-not-unwrap and do-not-decrypt flags.

24.9.1 Plain-Non-Trusted

Plain-non-trusted networks have no security requirements, and are not physically secure. Devices on these networks are allowed to generate and consume messages that are not signed. Secure devices on plain-non-trusted networks should assume that all plain messages are suspect and only execute requests contained in plain messages that are considered harmless.

The router that connects a plain-non-trusted network to a network with a different security policy must be a secure router. If the router's security policy allows it to route non-secured messages from the plain-non-trusted network onto a secure network, the router shall add the security wrapper and set the non-trusted-source flag.

24.9.2 Plain-Trusted

Plain-trusted networks have no security requirements but are physically secure. Devices on these networks are allowed to generate and consume messages that are not signed or encrypted. Devices on plain-trusted networks may assume that all plain messages are from sources that are allowed to communicate on the network.

Devices on plain-trusted networks should treat plain messages in the same manner as secure messages that use the General-Network-Access key.

24.9.3 Signed-Trusted

Signed-trusted networks require that all messages be at least signed. All devices connected to signed-trusted networks must therefore be secure devices. All plain messages received on a signed-trusted network shall be silently dropped.

24.9.4 Encrypted-Trusted

Encrypted-trusted networks require that all messages be encrypted. All devices connected to encrypted-trusted networks must therefore be secure devices and support encryption. All non-encrypted messages received on an encrypted-trusted network shall be silently dropped with the notable exceptions of Set-Master-Key, Request-Master-Key and certain Security-Response messages.

24.10 Network Security

The BACnet security architecture allows both for secure networks and for end-to-end security. Secure networks are created by setting the security policy for the network and configuring all devices on the secure network with the security policy of the network. This is accomplished by setting the Network_Access_Security_Policies property of the Network Security object in all devices on the network. In addition to the Network_Access_Security_Policies property, Devices also have a Base_Device_Security_Policy property that indicates the minimum level of security that the device requires for non-network infrastructure requests. This policy dictates the device's minimum level of security for sending or receiving messages.

Messages whose minimum security level is controlled by the Network_Access_Security_Policies are:

Who-Is-Router-To-Network,
I-Am-Router-To-Network,
I-Could-Be-Router-To-Network,
Reject-Message-To-Network,
Router-Busy-To-Network,
Router-Available-To-Network,
Establish-Connection-To-Network,
Disconnect-Connection-To-Network,
What-Is-Network-Number,
Network-Number-Is,
Register-Foreign-Device,
Who-Is,
I-Am

The minimum security level required for all other messages is specified by the Base_Device_Security_Policy.

Each security enabled router contains a network policy table that defines the security policy for each directly connected network. The network security policy table is defined by the Network_Access_Security_Policies property of the local Network Security object.

Where the Base_Device_Security_Policy property differs from an entry in the network security policy table, the higher of the two values shall dictate the minimum level of security for non-infrastructure communication to or from that network.

In contrast, end-to-end security is determined on a device by device and request by request basis. A security enabled device may contain a more detailed security policy to guide end-to-end secure communications. This policy may require security based on any number of factors, such as the service being requested, the object or property being operated on, the source of the request, etc. Note that when a device is located on a non-trusted plain network, the only way for the device to communicate to devices located on trusted plain networks is to use end-to-end security.

The limitations on device security policy are:

> No secure devices shall require that requests be plain. If a device receives a well-formed valid secure request, the device is not allowed to return an error indicating that the message should have been sent plain.
>
> No secure device connected to a network protected by a security proxy (see Clause 24.18) shall have a device policy that requires broadcast messages be secured.
>
> If a network contains any incapable devices, then the local policy for the network must be 'plain-trusted' or 'plain-non-trusted'. If a network contains any devices that do not support encryption, then the local policy for the network must not be 'encrypted-trusted'.
>
> The security policy of a network specifies the minimum security that messages must have if they are to be accepted from the network. A network that is physically secure might allow plain messages to be sent between local devices for efficiency reasons, but a router from that network might be configured to require signed or encrypted messages for communication beyond the local network.

24.11 End-to-End Security

In order to ensure that devices can determine the security expectations of peers, a response to a request shall use the same level of security as the request. For requests that result in a broadcast response, such as the unconfirmed application service Who-Is, the responding device is allowed to duplicate and send the response at other security levels as long as the other security levels do not violate the device's base local security policy or the network security policy over which the message is sent. The choice to send duplicates at other security levels is a local matter. Note that a device that is incapable of consuming a broadcast request due to its security level shall not respond in any way to the broadcast request.

24.11.1 Determining Exceptional Security Requirements

A device is allowed to require a higher security level on a per-service, per-object, or per-property basis.

Where the base device security policies and network security policies of communicating devices do not indicate the need for encryption, a secure device that is not configured to know ahead of time the sensitivity of a particular piece of data that is to be written to another secure device should attempt to read that data first before writing it without encryption. This is to allow the receiving device an opportunity to indicate that the data requires encryption by returning the error class SECURITY and error code ENCRYPTION_REQUIRED. Upon receiving this error, the client device now knows that the data should not be transmitted in the clear. If the sending device is configured to know ahead of time which data needs encryption, then pre-reading is not necessary.

24.12 Wrapping and Unwrapping Secure Messages

Messages can be secured either at the source device or en route in a router. Where the security is added or upgraded depends on the security policies and capabilities of the source device, destination device, and intervening networks.

24.12.1 Wrapping and Unwrapping By Routers

The application and removal of secure wrappers to/from BACnet messages occur at different points in a BACnet network. Each secure device along the communication path will evaluate the level of security required for the message and whether or not the message shall be forwarded along its route, or dropped. The rules shall be evaluated in the order presented, and only the first matching shall be applied to a message.

When unwrapping messages, a router shall validate the security protocol control octet, signature and timestamp, and verify uniqueness of the Message Id across the timestamp window. If the security protocol control octet, signature or timestamp is invalid, or the Message Id is not unique, then the message shall not be routed. In such a case, the router shall either remain quiet, or respond with an appropriate security error code as described in Clause 24.3.

A router may optionally validate the source MAC address, and/or destination device instance, This validation may be done when security is passed on untouched, removed, or modified. If the validation fails, the router shall silently drop the packet.

When routing messages without changing the security of the packet, a router may optionally validate the security protocol control octet, signature and timestamp, and verify uniqueness of the Message Id across the timestamp window. If the validation fails, the router shall silently drop the packet. Routers may optionally pass Security-Response messages secured according to Table 24-7's exceptions even when the security of the message does not meet the minimum requirements of the source network.

The rules governing wrapping and routing of messages are:

A router not configured for security processing (i.e., contains no configured security policy table, or does not support security NPDUs) shall route all messages according to this standard's Clause 6 rules.

The remaining rules only apply to routers configured for security processing:

A message shall be dropped if any of the following are true:

a. The message is plain and was received on a network with a local policy of 'signed' or 'encrypted'.
b. The message is not encrypted and was received on a network with a local policy of 'encrypted'.
c. The message is signed or encrypted, the non-trusted-source flag is set and the message is to be placed onto a plain-trusted network.

A plain message shall be signed with the General-Network-Access key if the security policy of the outgoing port is 'signed'. The do-not-unwrap flag shall be set to 0 and the secured-by-router flag shall be set to 1. If the message was received on a non-trusted network, then the non-trusted-source flag shall be set to 1.

A plain message shall be signed and encrypted with the General-Network-Access key if the security policy of the outgoing port is 'encrypted'. The do-not-decrypt flag shall be set to 0 and the secured-by-router flag shall be set to 1. If the message was received on a non-trusted network, then the non-trusted-source flag shall be set to 1.

A signed message shall be encrypted using the General-Network-Access key if the security policy of the outgoing port is 'encrypted'. The do-not-decrypt flag shall be set to 0.

A signed message shall be unwrapped if the security policy of the outgoing port is 'plain' and the do-not-unwrap flag is set to 0.

An encrypted message shall be decrypted and unwrapped if the security policy of the outgoing port is 'plain' and the do-not-decrypt and do-not-unwrap flags are set to 0.

An encrypted message shall be decrypted, but not unwrapped, if the security policy of the outgoing port is not 'encrypted' and the do-not-decrypt flag is set to 0.

All other messages shall be forwarded as is.

24.12.1.1 Routing Security Errors onto Plain Networks

When a router receives a security error that requires routing and the router has determined that the security wrapper should be removed before routing the message on, the router shall generate and forward a Reject-Message-To-Network message to the destination device. The reject reason shall be code 5 indicating that a security error stopped the original message from being processed by its destination device.

24.12.1.2 Routing To and From Plain Networks

Routers to plain networks shall enhance the security for the plain network and those devices interacting with them by providing bi-directional device id/address translation for all devices on the plain network.

The most secure form of device identification in a secure BACnet internetwork is the device instance. When messages are routed to and from a plain network, the device instance information is lost. To overcome this problem, secure routers to plain networks shall use device id/address translation replacing the SADR/DADR in messages with the device's instance.

When applying a security wrapper to route an incoming plain message onto a secure network, the router shall set, in the outgoing secure message, an SADR, in both the NPCI and the security wrapper, of length 3 that contains the source device's device instance, most significant octet first. If the message is a unicast message, the incoming DADR is expected to be a translated address, 3 octets in length, containing the destination device's instance. The router is responsible for locating the correct outgoing DADR for the specified destination device instance, and the correct device instance to use for the outgoing translated SADR for the source device.

When removing a security wrapper to route a secure message onto a plain network, the router shall set, in the outgoing plain message, an SADR, in the NPCI, of length 3, that contains the source device's device instance, most significant octet first. This source device instance will be available in the security wrapper of the incoming secure message, either in the Source Device Instance field if the security wrapper was applied by the source device, or in the SADR field if the security wrapper was applied by a router. If the message is a unicast message, the DADR is expected to be a translated address, 3 octets in length, containing the destination device's instance. The router is responsible for locating the correct outgoing untranslated DADR for the specified device on the plain network.

While the process of locating the correct DADR is a local matter, should a router be required to generate a request to find a device, the request should be restricted to the DNET specified in the request. If the DADR in a unicast request that is to be routed from a secure to a plain network, or from a plain to a secure network is not 3 octets in length, the request shall be dropped. In such cases the router shall generate and return a Reject-Message-To-Network message to the source device with a reject reason of 6 indicating the addressing information is erroneous. If the SADR is expected to be 3 octets in length when removing a security wrapper and it is not, a router shall drop the message and return a Reject-Message-To-Network message to the source device with a reject reason of 6 indicating the addressing information is erroneous.

24.12.2 Securing Response Messages

When a device responds to a secure request, whether it is a network layer request, an application layer request, or a BVLL request, the Security-Payload or Security-Response message(s) that contains the response shall be secured using the same key and to the same level (signed or encrypted and the same settings for do-not-unwrap flag and do-not-decrypt flag) as the request. This requires that security information associated with a request be passed along with the request as it moves between layers of the protocol so that it can be used to create a response that matches the request. The only exceptions to this requirement are when the responder is unable to provide the same level of security such as occurs when reporting certain security errors back to the requestor or when end-to-end security is required in which case the Do-not-unwrap flag is allowed to be set to 1.

There are some special cases where requests are to be secured as if they were a response to a previously received request. Unconfirmed requests sent in response to other unconfirmed requests such as I-Am-Router in response to a Who-Is-Router, I-Am in response to a Who-Is.

COV notifications shall be secured using the same key and to the same level as the SubscribeCOV or SubscribeCOVProperty request that created the subscription.

The level to which event notifications are secured shall be a local matter and not dependant on the security of the services used to configure the Notification Class object that directs the events.

All requests that make up a Backup or Restore procedure shall be secured using the same key and to the same level as the initial ReinitializeDevice request. No operation within a Backup or Restore procedure shall require a different key or higher level of security if the initial ReinitializeDevice request succeeds.

24.13 Authenticating Messages

Whenever security is removed from a message, either at the destination device or en route through a router where the destination network policy differs from the source network, the message should be authenticated. The authentication process consists of validating, in order:

(a) validating the Security Header Protocol Control Information
(b) message signature.
(c) source MAC address.
(d) uniqueness of the Message Id.
(e) timestamp.
(f) destination device instance

24.13.1 Validating the Security Header Protocol Control Information

All devices that receive and process or routers that modify the security on a secure message that is to be forwarded onto another network shall validate the Security Header Protocol Control Information.

If the check fails, the message shall be silently dropped. The check is described in Figure 24-6.

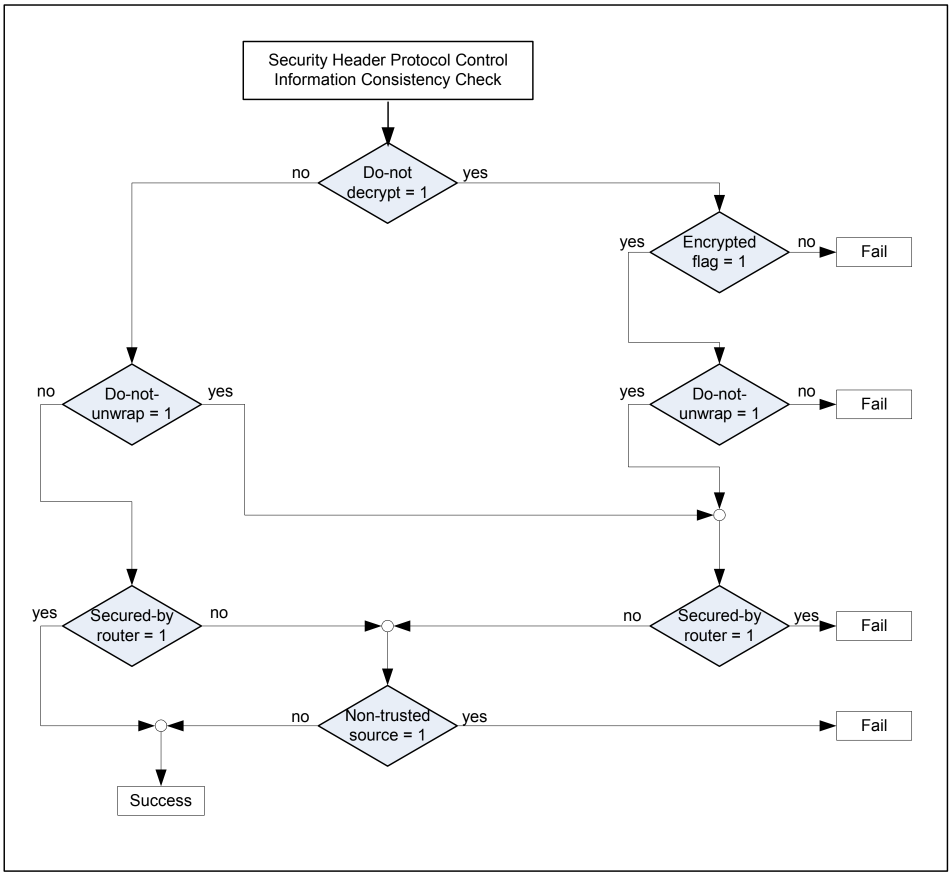


Figure 24-6: Security Header Protocol Control Information Consistency Check

24.13.2 Validating the Signature

All secure devices that receive and process a secure message shall validate the signature of the message. Routers are not required to validate the signature of a message unless the router changes or removes the security wrapper of the message, or the router is final destination for the message.

The signature is validated by calculating the signature as described in Clause 24.7.4. The validation succeeds if the calculated signature is the same as the signature contained in the message.

If the received message is encrypted, before validating the signature, the padding has to be removed. Implementations shall verify that the padding length is less than the size of the security wrapper and that the remaining security wrapper after removal of the padding is at least as large as the smallest legal security wrapper.

If the message is a Security-Response conveying the error securityNotConfigured, the signature shall not be validated.

24.13.3 Validating the Source MAC Address

All devices that receive and process or forward a secure message have the option to validate the MAC address that the message was received from. Validation of the source MAC address is useful when network address translation (NAT) is in use within the BACnet inter-network.

When validating the source MAC address of a message, there are 3 cases to consider:

(a) The message originated from a device that resides on the network from which that the message was received.
(b) The message originated from a device that resides on a remote network.
(c) The message does not contain source MAC address information in the security header.

When a message is received from a local device, the MAC address should be compared to the signed SADR in the security header. If the values match, then the source MAC address validation succeeded. If the values do not match, then the check failed. If the check fails the receiving device has the option to challenge the source of the message by sending a Challenge-Request to the MAC address (not the SADR from the security header). This check would most likely be employed when the network is of a type where address translation is an expected behavior, such as BACnet/IP. If the challenge succeeds, then the source MAC address validation will be considered to have succeeded.

When a message is received from a remote device, the MAC address should be compared against any locally cached value for a router to the source network. If the MAC address matches the cached value, then the source MAC address validation succeeded. If the values do not match or the device does not have a locally cached router address for the network, then the check failed and the receiving device has the option to challenge the router of the message. If the challenge succeeds, then the source MAC address validation will be considered to have succeeded.

To challenge a router to a network, a device may send a secure unicast Who-Is-Router-To-Network to the MAC address from the message. If a valid secure I-Am-Router-To-Network is received in response, then the challenge will have succeeded.

An alternative method to challenging the router is to challenge the originator of the message, ensuring that the MAC address sent to is the address the original message was received from. If the challenge succeeds, then the source device is reachable via the address the message was received from. This is the only method that is available for validating the route to the Key Server when the device has not yet been provided with a General-Network-Access key.

If the SLEN field of the security header is 0, then the source address information is not present, and thus not signed. The only option for validating the source address is to challenge the source device. If the challenge succeeds, then the source MAC address validation will be considered to have succeeded.

24.13.4 Validating the Destination Device Instance

Secure messages include a Destination Device Instance field that indicates the device that the message is intended for. A secure device shall only accept a message that has either a Destination Device Instance field of 4194303 or its own device instance.

A secure device is allowed to refuse unicast requests that have a Destination Device Instance of 4194303. In such cases, the device shall respond with an error code of destinationDeviceIdRequired. While a device is allowed to refuse unicast requests with a Destination Device Instance of 4194303, it is strongly recommended that this not be a device's default approach. Doing so will result in the device being unable to interoperate with incapable devices from plain-trusted networks.

24.13.5 Validating the Message Id

All secure devices that receive and process a secure message shall validate the Message Id of the message. Routers are not required to validate the Message Id of a message unless the router modifies or removes the security wrapper of the message, or the router is the final destination for the message.

The Message Id cache is based on:

> Packet Reorder Time: a period of time across which out of order packets can be received without causing Message Id validation problems.
>
> Last Message Id: the largest Message Id received from a device more than Packet Reorder Time seconds ago. This value is stored for each device from which secure messages are received.

All secure devices shall cache data records about recently received valid messages. The cached data records shall consist of the Message Id, the source device instance, and the time at which the message was received. In addition, a Last Message Id value is stored for each secure device being communicated with.

When a message is received, the cache shall be checked and if the Message Id is already present in the cache, or if the Message Id is less than the Last Message Id (taking into account wrapping), then the Message Id validation shall fail. Otherwise the Message Id Validation shall succeed.

If the Message Id validation, destination device instance validation, timestamp validation, and signature validation all pass, then a new record shall be added to the cache and the Message Id validation shall succeed.

The cache holds Message Id, Timestamp, Source Device tuples that are received over the last Packet Reorder Time window. When an entry in the cache becomes older than Packet Reorder Time, the Message Id is compared to Last Message Id for the source device (taking into account wrapping) and if it is larger, Last Message Id is set the Message Id of the entry removed from the cache,

If no messages are received from a device within Security Time Window seconds, then Last Message Id is cleared for that device. This is to ensure that devices can wait Security Time Window seconds after power up and then safely re-use any Message Id without it being rejected as a duplicate.

The size of the cache is a local matter. Determining which entries to remove from the cache when it overflows is a local matter.

24.13.6 Validating the Timestamp

All secure devices that receive and process a secure message shall validate the Timestamp of the message. Routers shall not validate the Timestamp of a message unless the router changes or removes the security wrapper of the message, or the router is the final destination for the message.

The timestamp shall be checked against the clock of the receiving device. If the Timestamp is within +/- Security_Time_Window seconds of the local time, then the validation shall succeed, otherwise the validation shall fail.

24.14 User Authentication

The BACnet standard provides one standard user authentication mechanism.

24.14.1 Proxied User Authentication

The Proxied Authentication mechanism is identified by an Authentication Mechanism field value of 0. When using this mechanism, the Authentication Data field only contains the User Id and User Role. The User Id in the message is assumed to have been authenticated by the source device when the key is anything other than the General-Network-Access key. See also Clause 24.2.1.11.

The User Id identifies the user requesting the operation and the User Role identifies the role under which the user is performing the operation. Secure devices shall not expect to use a fixed set of User Ids or user Roles. User Ids and User Roles will be assigned during site installation and setup and as such shall be configurable in both client and server devices. Devices that are capable of providing User Id and User Role values in secure messages shall be capable of being configured to provide any User Id, User Role value pair for each of the users that it supports. Devices that do not ignore

User Id and User Role values in received messages shall be capable of being configured to accept and process any User Id or User Role value. The method used to configure User Ids, User Roles and authorization rules is a local matter.

User Roles of 0 and 1 indicate that a request is initiated by "the system itself", that no user authentication was performed, and no user authentication was required. A User Id of 0 indicates that the actual user identity is unknown and is commonly used in conjunction with a User Role of 0 or 1.

A server device that receives a request with a User Id that is secured with the User-Authenticated key is allowed to apply local user authorization to determine whether the user is authorized to perform the requested service. If the user does not have sufficient rights for the operation, the server device shall return an error. For Confirmed-Request PDUs, a Complex-ACK-PDU or an Error-PDU conveying a class of SECURITY and an error code of WRITE_ACCESS_DENIED, READ_ACCESS_DENIED, or ACCESS_DENIED shall be returned. The error code returned depends on the service requested.. For BVLL and network layer messages that expect a response, a Security-Response message shall be returned that conveys the error code accessDenied. All other requests shall not result in errors being returned.

If a server is not configured to apply authorization rules, then the user authentication information may be ignored. If a server is configured to apply authorization rules and the message is secured with the General-Network-Access key, it is a local matter as to whether or not the request is granted. Exceptions to this rule are the Who-Is and Who-Has services which shall be executed.

The authorization scheme details, and the method of configuring and maintaining the scheme, are a local matter.

24.15 Time Synchronization Requirements

Because message timestamps are used to prevent replay attacks, secure devices are required to maintain the current time. Also, because message timestamps are in UTC, secure devices are required to support the UTC_Offset property in their Device object.

24.15.1 BACnet Time Synchronization Messages

Secure BACnet devices should not accept non-secured TimeSynchronization or UTCTimeSynchronization requests, unless the local network security policy is plain-trusted. Doing so will leave the secure device vulnerable to replay attacks.

24.15.2 Overcoming Non-synchronized Clocks

The network security architecture relies on devices having loosely synchronized clocks. If there are secure devices within the system whose clocks are not within the time window of each other, then those devices will be unable to communicate with each other and the system will be susceptible to replay attacks..

Devices that cannot keep accurate time are open to replay attacks as their clocks drift away from the clocks of the other secure BACnet devices. It is essential that Time-Synchronization is performed frequently enough to keep all device clocks well within Security_Time_Window seconds of each other.

It is worth noting that when a secure device is powered off for extremely long periods of time (years), the clock in the device may drift sufficiently far that it is no longer synchronized with the rest of the devices on the network.

24.15.2.1 Devices Without Real Time Clock Chips

Devices that do not keep time over a reset or while turned off will need to acquire the time before they can communicate safely.

The preferred method for determining the current time in such cases is for the device to use a special non-repeating Message Id that is remembered across resets. The device waits 2 * Security_Time_Window and then generates a Challenge request with the next special Message Id (note that there is no relationship between the Message Id used for this purpose and the last Message Id used by the device before it reset). The Challenge request shall be sent with an all 0 timestamp.

To generate the next special Message Id to use, a constant value is added to the previously used special Message Id. The constant, modulo 2^{32}, shall be prime. This ensures that when the value wraps, it wraps past 2^{32} to a previously unused value. Note that larger constants are generally better than smaller ones as more "entropy" is created making the signature, and any encryption, more secure.

If the secure device does not have the current key set, the Challenge request will have to be aimed at the Key Server and use the device's Distribution or Device Master key.

Other methods for determining time are described below. Each of the methods outlined requires that the device has the ability to generate a random value that will be used in the Message Id and/or Timestamp fields of the security header. Use of standard programming language random functions will not meet this requirement. The device must use its interaction with the outside world to generate a random value (inter-message delay, total network traffic, values found in DRAM if it is not cleared on startup, etc). If the same Message Id value is used each time on startup, then an attacker can record a sequence of messages when a device resets and then replay them at a later date when the device resets again.

24.15.2.1.1 Monitoring Broadcast Traffic

The device can determine the time by monitoring the network for broadcasts. If a broadcast is seen that is secure and validates, then the time can be taken from the message and used in a Challenge-Request containing a random Message Id and aimed at the source of the message. If the challenge succeeds, then the time can be accepted.

24.15.2.1.2 Monitoring For Any Traffic

The device can determine the time by monitoring the network for any traffic. Once a message is received, the source MAC address can be removed from the message and a Challenge-Request formed and sent to the source of the message. If the challenge succeeds, then the time can be accepted.

24.15.2.1.3 Wait For TimeSynchronization

The first two solutions may result in incorrect times being used by the device. If the device chooses a message from a device that has an invalid time, then the device will end up with the same invalid time.

A third option is for the device to be configured to challenge a time master on restart, or be programmed to remember the source address of the last successfully processed TimeSynchronization or UTCTimeSynchronization request and challenge this address on start up. The device should update its time only if the Challenge succeeds.

Alternatively, the device could wait for a TimeSynchronization or UTCTimeSynchronization request. When one is received, the device should challenge the message source and only accept the time if the Challenge succeeds.

24.16 Integrating the Security Layer into the BACnet Stack

24.16.1 Secure PDU Sizes

The addition of the BACnet security wrapper into the BACnet PDU reduces the amount of space for BACnet NSDUs and APDUs.

For secure devices, the value of the Max_APDU_Length_Accepted property of the Device object shall be calculated assuming the largest size of valid Authentication Data the device is configured to accept, and the largest encryption block size of any BACnet security encryption algorithm the device is configured to use.

Secure routers that are capable of routing between 2 BACnet/IP networks shall be capable of routing BACnet/IP messages that contain an NSDU of 2000 octets. This allows for 523 octets of authentication data to be included in the security header when transferring packets between BACnet/IP networks. All other BACnet routers are only required to support routing of message sizes indicated in Table 24-28.

In order to reduce the chance of inverted networks being used when security tunnels are in place, the maximum APDU for secure BACnet/IP segments is kept at 1476. To allow for this, the maximum length BACnet/IP NPDU is increased to 1562 for secure BACnet/IP messages. This allows full size non-secured APDUs to be secured and passed through BACnet/IP networks.

The following table of maximum NSDU and APDU lengths is calculated assuming an Authentication Mechanism of 1 with 2 octets of Authentication Data and an encryption block size of 16 octets.

Table 24-28. Maximum APDU Lengths for Secure BACnet Data Link Layers

Data Link Technology	Maximum APDU Length
ISO 8802-3 ("Ethernet"), as defined in Clause 7	1420 octets
ARCNET, as defined in Clause 8	412 octets
MS/TP, as defined in Clause 9	412 octets
Point-To-Point, as defined in Clause 10	412 octets
LonTalk, as defined in Clause 11	140 octets
BACnet/IP, as defined in Annex J	1476 octets

When reporting the Max_APDU_Length_Accepted in the APCI of a ConfirmedRequest-PDU, a secure device might be forced to indicate a value smaller than indicated by its Device object. While this will result in successful communications, the network bandwidth usage of large segmented messages will be sub-optimal.

Therefore devices responding to ConfirmedRequest-PDUs may ignore the value indicated in the APCI and use the value indicated in the client's Device object instead, if it is known.

24.16.2 Selecting Error Codes

The security errors can be broken down into two groups: general security layer errors and authorization errors. General security layer errors are those that are required to be generated and returned by the security layer and indicate a problem with the formulation of the message itself. Authorization errors may be generated by the security layer, or they may be generated by another layer of the BACnet stack or by the device's application program.

The order in which the error conditions are checked is important. The general security layer errors are checked in the order they occur in the service's error table and are checked before any authorization errors. The order in which authorization errors are checked is not important except that the generic accessDenied error code shall not be generated if one of the other authorization error conditions exists. This ensures that knowledge imparted by the other authorization error codes is not lost through the over use of the accessDenied error code. It is recommended that the encryptionRequired error condition be checked before the incorrectKey error condition because this may help to reduce the amount of failure traffic produced.

If multiple error conditions are present, the condition that is checked first shall be the error that is returned.

Some of the error conditions are ignorable. When an ignorable error occurs, it is a local matter as to whether the device returns the error or does not respond at all. Note that if multiple errors are present and the one that is checked first is ignorable, then the device has the choice to either return that error or not respond at all; the device does not have the option to return an error indicating one of the other error conditions that exist.

24.16.3 Communicating Security Parameters

While the BACnet Security Layer is responsible for securing BACnet PDUs, the other layers of the BACnet stack are usually the source of the security parameter information. For example, the determination of whether or not any of the data in an APDU requires a higher level of security than the device's default policy is made by the application program. The application, network and BVLL layers of the BACnet stack provide security information to the security layer via the security_parameters ICI parameter.

24.16.4 Detecting and Processing Security Errors

Security errors can be detected both by the security layer and also by the layer of the stack executing a request. For example, the security layer will detect problems with signatures whereas the application program will detect most authorization errors. The result is that some security errors are returned in Security-Responses and others will be returned in Error-PDUs or ComplexACK-PDUs.

When a security error is received by a device, the error information may need to be indicated to the other layers of the BACnet stack. The security layer indicates the error information via the N-REPORT.indication primitive. The application layer indicates security errors to the local application program via the SEC_ERR.indication primitive.

Table 24-29 provides a discussion of the different security errors, the conditions that the errors indicate, and possible failover actions that devices can take.

Table 24-29. Security Errors

Description	**Suggested Failover Actions**
securityNotConfigured	
The recipient is not configured for security on this port. A Security-Response containing this error will not be signed nor encrypted by the source device.	If the source device is allowed to re-generate the request with no security (a 0 signature and not encrypted), do so otherwise fail the request. This error should be reported to a management entity and then silently dropped.
encryptionNotConfigured	
The Encrypted field is set to 1 and the receiving device is not configured to accept encrypted messages. A Security-Response that contains this error will not contain a valid Original Message Id as the responding device would be unable to decrypt the source message to obtain the MessageId. A Security-Response containing this error will never be encrypted by the originator although it may be encrypted by an intervening router.	Report to the stack layer that created the original request to allow it to decide if the request should be retried with different security parameters. Since the device cannot match it to a specific request, all outstanding requests sent with encryption to the specific device should be cancelled and retried.
unknownKey	
The Key Identifier field indicates a security key that the receiving device does not know.	Report the error to the stack layer that created the original request to allow it to decide if the request should be retried with different security parameters. If the original request was encrypted, the device that generated this error would not have been able to match it to a specific request, all outstanding requests sent with encryption using the specific key to the specific device should be cancelled and retried.
duplicateMessage	
A message with the provided Message Id has already been received from the source device within the security time window.	If the source device has restarted (within 2 * Security_Time_Window), this may be due to issues surrounding the initial MessageId used. In such cases, the device should stop all network traffic for a complete Security Time window, or retry with a larger Message Id. If traffic is to be halted, then indicate the error to the stack layer that created the original request so it has the option to retry later. If the source device has not recently restarted, then this error should be reported to a management entity.

unknownKeyRevision	
If the Key Revision field indicates a revision that the receiving device does not know. A Security-Response containing this error code should never be encrypted as the intended target will most likely not be able to decrypt the message.	If the source device has not recently checked its Key Revisions with the Key Server (definition of recently is a local matter), the device should check them using the Update-Key-Set message. This failover action should be taken by the security layer, not the application entity. The source device should either, resend the packet if the keys are found to be out of date, or indicate the error to the stack layer that created the original request resulting in the request failing completely. If the source device has recently checked its Key Revisions, its local policy is to not check, or if its keys are found to be current, then this error should be reported to a management entity. It is advisable to report this problem to the management entity regardless because frequent occurrences of this error indicate a problem with key distribution or network robustness. The device that generates this error should also check its key revision with the Key Server if it has not done so recently and its key table does not indicate that the unknown key revision recently expired.
malformedMessage	
If the length of the data received is too short to contain the security header, service parameters and the signature or the size of padding is larger than possible for the data received, etc.	The error should be reported to a management entity and dropped silently.
badSignature	
If the signature is not correct. This error may also be indicated if a decryption error occurs. This error indicates that either the packet was tampered with, a network error occurred, the key values differ, or some other kind of attack is underway.	If the signature of the response is correct, then the key values are correct. In this case, the error should be reported to a management entity, and the packet retried. If the signature is not correct, then the error should be reported to a management entity and dropped silently.
badDestinationAddress	
If the destination address information is missing or invalid. This is an indication that the packet was modified on route, replayed to the wrong device, the physical structure of the network has been tampered with or the address-binding table in the source device is stale.	If the local binding table is stale in the source device, the source device should clear the entry and re-bind to the destination device. The source device should be able to determine if the local binding table was stale based on the information received in the Security-Response message. If the original packet is still available, resend it, otherwise indicate the problem to the stack layer that created the original request and let that layer regenerate the packet. If the binding table is not stale, report the problem to a management entity.
badDestinationDeviceId	
If the Destination Device Instance does not	The source and destination devices should

match the local device instance. This is either an indication that a redirection attack is being attempted, that the local device has recently changed its device instance, or that the source device has stale addressing information.	report the error to a management entity. If the destination device provides an error response to the source device, or if the source device times out waiting for a response, the source device should clear the entry in its local binding table and re-bind to the destination device.
badSourceAddress	
If the source address information is invalid. This is an indication that the general network configuration is incorrect (there is disagreement on network numbers between routers and devices), or that the physical structure of the network has been tampered with.	The source and destination devices should report the error to a management entity.
badTimestamp	
The Timestamp in the security header of the message is not within the allowable timestamp window of the receiver. This indicates a problem with non-synchronized clocks, or an attempt to replay a message. The receiver of the Security-Response should be able to tell if it is a clock issue.	If it is a clock issue, the request could be re-tried with a timestamp adjusted to suit the other device. If it is not retried, the error should be indicated to the stack layer that created the original request and reported to a management entity.
destinationDeviceIdRequired	
If the Destination Device Instance in the security header of a unicast message has the value 4194303 and the destination device requires this value to be set correctly for the operation requested. This error may be generated by the application program and returned in an Error-PDU or ComplexAck-PDU.	The error should be reported to the stack layer that created the original request and the request should not be retried with the information because we cannot be sure that this is the device the application really should be communicating with and would thus thwart the protection the device was trying to achieve by returning the error.
unknownSourceMessage	
This error occurs when a Challenge fails.	The device should report the error to a management entity. The message that was being Challenged should be dropped and no response sent to its source device.
cannotVerifyMessageId	
This error occurs when a device has no record of sending the specified message. This may be sent if a device generates enough traffic that its local cache of message history has overflowed and it is unable to ascertain for sure that it did not send the message.	The device should report the error to a management entity. The message that was being Challenged should be dropped and no response sent to its source device.
encryptionRequired	
This error occurs when an operation requires encryption but the request was sent in the clear.	If the source device supports encryption for non-key exchange messages, it should retry the request with encryption. If the device does not support encryption, it should report the problem to a management entity.
sourceSecurityRequired	
If the secured-by-router flag is 1 and end-to-end security is required, or the Do-not-decrypt flag is 0 and end-to-end encryption is required for the operation requested. If the device's minimum security level requires	If the error is in a Security-Response, the error should be indicated to the stack layer that created the original request. The device should retry the request, with different security parameters.

end-to-end security this error is generated by the security layer. The equivalent application program error class and error code can also be returned by the application program if the requirement for end-to-end security is due to data specific rules. In such a case, the error will be returned in an Error-PDU or ComplexAck-PDU.	
incorrectKey	
The key provided to secure the message does not indicate sufficient authority to perform the requested operation. This error is usually returned by the application program in an Error-PDU or ComplexACK-PDU.	If the error is in a Security-Response, the error should be indicated to the stack layer that created the original request. The device should retry the request, with different security parameters.
notKeyServer	
The device that received the Request-Key-Update message is not configured as a Key Server or is not configured as the Key Server for the requesting device.	This error usually indicates that the Key Server configured to provide keys for the source device has been moved. The device should retry the request using a broadcast address.
keyUpdateInProgress	
The destination device is performing a key update and is unable to perform the request action.	The error indicates that a different Key Server is updating the device. The problem should be reported to a management entity, and the Key Server should retry the operation at a later time.
cannotUseKey	
The destination device is unable to use a key provided by the Key Server.	This indicates that the Key Server has been configured to issue keys to a device that it is unable to accept. The Key Server should report the problem to a management entity and attempt to provide all other security keys to the device.
tooManyKeys	
The destination device is unable to fit all of the keys that the Key Server has provided.	This indicates that the Key Server has been configured to issue too many keys to a device. The Key Server should report the problem to a management entity.
invalidKeyData	
The desitnation device determined that the key data provided is invalid.	This indicates an implementation error in the Key Server or in the desitnation device. Both devices should report the problem to a management entity.
unknownAuthenticationType	
If the user authentication method in the message is unknown to the device. This error is usually returned by the security layer, but might be returned by the application program depending on which layer is responsible for authentication checks. It might also be that the device accepts the whole security request and passes it to the application program with authentication mechanism and data marked as "unknown", letting the device decide whether the operation can be allowed	If other authentication methods are known, the request can be retried with one of those methods.

without knowing the authentication mechanism and the user requesting the action. Thus it lets the decision be that of the application program.	
accessDenied	
The network layer or BVLL request was denied due to insufficient authorization. See Clause 24.14.1 for more details. While the security layer may return this error code, for application layer requests, the application program should use other related error codes in Error-PDUs and ComplexACK-PDUs.	There is no suggested failover action for this error condition.

24.16.5 Security Errors in Network Layer Initiated Packets

The network layer does not have a mechanism for returning errors in response to a message and thus the information that can be conveyed in response to an error is limited.

If the network layer detects a security error condition, such as insufficient authorization, the network layer shall convey this information to the local security layer so that a Security-Response indicating the error condition can be generated. Non-security errors shall be handled as according to Clause 6.

Network layer messages that embody operations, in contrast to route discovery and traffic management, may require security stronger than the base security policies. The network layer messages that may require a higher level of security are: Initialize-Routing-Table Establish-Connection-To-Network, and Disconnect-Connection-To-Network. All other network layer messages shall be accepted if they are secured according to the network's security policy.

Reject-Message-To-Network messages shall be secured with the same level of security as the original request. This may result in intermediate routers being unable to process the Reject-Message-To-Network message. In such cases, the intermediate routers will not benefit from the information but the source device will.

24.16.6 Security Errors in BACnet/IP BVLL Initiated Packets

The BACnet/IP BACnet Virtual Link Layer does not have a mechanism for providing detailed error information. All it is able to indicate is that a request failed, it cannot provide details.

If the BACnet/IP BACnet Virtual Link Layer detects a security error condition, such as insufficient authorization, and the request was received in a Secure-BVLL, the layer shall convey this information to the local security layer so that a Security-Response indicating the error condition can be generated. Otherwise error conditions shall be handled according to Annex J.

Forwarded-NPDU, Distribute-Broadcast-To-Network, Original-Unicast-NPDU, Original-Broadcast-NPDU are just payload messages transferring packets from higher levels. They shall not require security higher than the network's local security policy (although any contained PDU may be rejected due to insufficient security by a higher layer of the BACnet stack).

BVLL messages that embody operations may require stronger security than the local network security policy. The BVLL messages that may require a higher level of security are: Delete-Foreign-Device-Table-Entry, Read-Broadcast-Distribution-Table, Read-Foreign-Device-Table, Register-Foreign-Device, Write-Broadcast-Distribution-Table. As with all responses BVLC-Result and all of the ACK BVLL PDUs shall be secured to the same level as the original request.

24.16.7 Data Hiding

In secure devices, some data within a device will not be available to all clients. Where a subset of data within an object, property, or service is restricted, a secure device shall hide the portion of the data for which access is restricted.

24.16.7.1 Hiding Properties

If an object in a secure device contains optional properties for which access to requires a higher level of security, those properties may be hidden by the secure device. The secure device has the option to return a security error (ACCESS_DENIED, READ_ACCESS_DENIED, or WRITE_ACCESS_DENIED), or to return a general error (UNKNOWN_PROPERTY). This method of data hiding is not available for required properties; restricted access to required properties has to be reported via a security error.

When a special property identifier (ALL or OPTIONAL) is used to access restricted optional properties, the device has the option to exclude the optional property completely from the result, or to include a security error in its place.

24.16.7.2 Hiding Array Elements

When elements in an array or list have different security requirements, the secure device shall hide elements of the property by making it appear that the elements do not even exist. The array or list will appear to be shorter, and entries that occur later in the property will be shifted down to fill the entries which are hidden. This approach to data hiding is not applicable to arrays or lists for which the length is mandated by the standard.

For example, when reading the Object_List property of the Device object with different security levels may result in a different value being returned. Where objects have to be completely hidden, the length of the Object_List should be adjusted so that entries do not show up.

24.16.7.3 Hiding Service Results

For services that report collections of data, such as the alarm summary services, the data returned by the service may change based on security of the request. For example, if a secure device contains alarm generating objects, for which access to requires a higher level of security, the alarm summarization services should not return entries for those objects regardless of whether they meet the summarization criteria or not.

A secure device does not have the option to leave out a required service response parameter, or in any other way modify a response such that it is inconsistent with the rules for that service response.

24.16.8 Device Identity

24.16.8.1 Security of Device Identity

The device instance is the only secure form of device identity. Network number and MAC address values should not be considered to be secure as these can be changed due to a network configuration change or can be manipulated through complicated attacks on a BACnet network.

When referring to another device such as is done in Notification_Class objects, it is always better to refer to the device via the device instance and not via the device's network number and MAC address. This is also better for handling cases where device MAC addresses may change, such as when DHCP is in use.

Devices should never use network number or MAC address information to make authorization decisions. The only device identity information that should be relied upon for making authorization decisions is the device instance.

24.16.8.2 Modifying A Device's Identity

Modifying identity information that is used by the security layer can result in problems within the layers of the stack. For example, changing a device's instance number or network number via a WriteProperty request may result in problems associating the response with the request.

When critical identity values (device instance, network number, MAC address) are modified by a service, the actual change of identity shall occur after the response has been sent so that the response is secured with the same identity that the request was secured with.

A device may, at the implementor's discretion, delay the application of identity value changes until the device is reset.

24.17 BACnet Security In A NAT Environment

A secure foreign device that resides behind a NAT cannot include a correct SADR in secure messages because the device does not know it. While BBMDs are given static IP addresses, a foreign device's address is not usually static. To overcome this issue, secure foreign devices that communicate through a NAT to reach the BACnet internetwork may first send a Challenge-Request message to the BBMD that they intend to register with as a foreign device. The resulting Security-Response will let them know their SADR as seen by other BACnet devices.

24.18 BACnet Security Proxy

In order to provide security for devices without requiring the devices themselves to be secure, a new type of device called a Security Proxy is defined.

A BACnet security proxy device is a secure router that strips and adds security on behalf of the devices "behind" it that are incapable of, or not configured for, secure communications. The proxy will remove security from messages destined for the protected devices, even if the do-not-unwrap or do-not-decrypt flags are set. The proxy will also determine when security should be applied to messages that are originated by protected devices and will apply it accordingly. The methods used by a security proxy device to determine when to apply security, what level of security, and what user information to use are local matters.

In its simplest form, a security proxy device protects a complete BACnet network or collection of BACnet networks but it is not restricted to operate in such a manner. A security proxy device could be implemented such that it protects a subset of the devices on the protected networks.

The BACnet security proxy functionality is optional and is not required to be supported by secure BACnet routers.

24.19 Deploying Secure Device on Non-Security Aware Networks

Where a network is served by a BACnet router that does not forward globally broadcast unknown network messages, global broadcasts of security messages will not be routed. This limitation will restrict the methods used to deploy secure devices in existing networks.

In particular, a secure device will be unable to request security keys using broadcast Request-Master-Key or Request-Key-Update requests. There are three options for deployment in this situation:

- Connect the device to the Key Server's local network to receive the Device-Master key and Key Server address information.
- Provide the device's network number, MAC address and device instance manually to the Key Server and manually interact with the Key Server to get it to provide a Device-Master key to the device.
- Enter the Key Server's information into the Last_Key_Server property of the device's Network Security Object, and then cause the device to request a Device-Master key using a unicast Request_Master_Key directed at the device indicated by Last_Key_Server.

In order to ensure that other secured messages are routed by the legacy router, secure devices should refrain from relying on globally broadcast security messages for proper operation.

24.20 Deploying Secure Single Network Installations

BACnet security requires that all devices know their local network numbers. To make support for this requirement for simpler installers, secure devices can leverage the What-Is-Network-Number and Network-Number-Is services. While this decreases the installation work required, it results in problems when no routers are deployed in the installation.

To ensure that this does not cause problems during the installation process, it is recommended that all secure devices support an alternate method for configuring the local network number.

24.21 Security Keys

In BACnet security there are six types of keys: General-Network-Access, User-Authenticated, Installation, Application-Specific, Device-Master, and Distribution. Each key actually consists of a pair of key values, one used for signatures and one used for encryption.

The General-Network-Access key is used for device and object binding, for encryption tunnels, and by user interface devices that cannot authenticate or are not trusted to authenticate a user. All secure BACnet devices shall support use of the General-Network-Access key.

The User-Authenticated key is used by devices that are allowed to authenticate the user's identity that is included in BACnet messages. It is also given to devices that do not contain a user interface. All secure BACnet devices shall support use of the User-Authenticated key.

Installation keys are distributed to pairs of devices, usually the configuration tool of a technician and a set of BACnet devices that require configuration. These keys are provided to allow temporary access to a specific set of controllers through a configuration tool that should not normally have access to the BACnet network. All secure BACnet devices shall support use of the Installation key.

In order to provide security boundaries between application areas, such as access control and HVAC, the BACnet security framework provides Application-Specific keys. All secure devices shall support at least 1 Application Specific key; support for multiple Application-Specific keys is optional. The semantics of the Application-Specific keys are site-specific, and as such all devices shall not restrict which Application Specific key identifiers may be accepted into their key sets.

The Device-Master key is a unique key per device that is either given to the device before the device is installed, or provided to the device by the Key Server during installation. In theory, the Device-Master key is never changed for a device other than during initial site setup. In practice, a device's master key may have to be changed if a Key Server's key database is lost and cannot be recovered. The Device-Master key is only used to provide Distribution keys to devices.

Distribution keys are unique per device and are used only to distribute key sets.

24.21.1 Key Identifiers

Keys are identified by their 2-octet key identifier. The identifier indicates which key is to be used and the signature and encryption algorithms that the key is used with.

The first octet of the Key Identifier field indicates the encryption and signature algorithms to be used. The values are:

Table 24-30. Key Identifier Algorithm Enumeration

Value	Algorithm (Encryption/ Signature)
0	AES / MD5
1	AES / SHA-256
2..255	Reserved

The second octet of the Key Identifier indicates the key being used. The values are:

Table 24-31. Key Identifier Key Number Enumeration

Value	Key number
0	(not used)
1	Device-Master
2	Distribution
3	Installation
4	General-Network-Access
5	User-Authenticated
6..127	Application-Specific Keys
128..255	Reserved

While the key identifier format would allow the specification of multiple keys of each type (such as the General-Network-Access key) differing only in the algorithms used, the intent is that only one key of each type would be used in practice.

24.21.2 Key Sets

A key set consists of all of the keys that a BACnet device is configured to have excluding the Device-Master key and the Distribution key, a time period during which the keys are valid and a key revision number. The key revision number applies to all of the keys in a key set.

A key shall not be used outside of the timeframe defined for the key set. To allow for variations in clocks between devices, a key's acceptability shall be determined by using the timestamp placed in the message by the sending device rather than the local time in the receiving device.

Each BACnet device has two key sets with possibly overlapping valid time periods. This allows a device to have a current and a new key set for transitioning between key sets. When a device contains 2 valid key sets, the device should secure messages with the newer of the two key sets as soon as its valid time period will allow.

If the key set's Expiration Date is X'FFFFFFFF', then the key set shall not expire. To allow for this deployment scenario, devices shall maintain key data in a non-volatile manner and shall not be dependent on a Key Server after a power up or reset.

A key revision of 0 indicates that the key set has not been configured. Therefore, when the key revision number wraps after being incremented past 255, it shall wrap back to 1, not 0. If the key set's revision number is 0, then any keys it contains shall be ignored.

24.21.3 Key Distribution

In general, the keys used by the BACnet security framework are not unique to each device, or device pair. The General-Network-Access key is shared by all of the BACnet devices and the User-Authenticated key is shared by most devices. In order to increase the security of the keys, they should be changed periodically.

Device-Master keys must be configured in a secure device and shared with the Key Server before the Key Server can provide keys to a device. Initial key distribution is discussed in more detail in Clause 24.22.3.

Keys are distributed as a set, excluding the Distribution key which is distributed on its own. The key revision number applies to all of the keys in the key set.

To protect the keys, the distribution messages shall be encrypted. The encryption algorithm used for distribution of keys shall be the same encryption algorithm as used with the most secure key in the distribution.

24.22 Key Server

BACnet Key Servers are responsible for the generation and distribution of BACnet security keys. If automatic generation and distribution of security key updates is required, the BACnet installation will require a permanent Key Server that is responsible for generating and distributing keys to all of the BACnet devices.

The Key Server is configured with a list of the devices to update, the keys that each should be given, and the period at which the keys should be distributed. In addition, the Key Server shall allow the operator to initiate a key update at any time.

24.22.1 Key Generation

The Key Server will be responsible for generating all keys, except for factory-fixed Device-Master keys. The algorithm used to generate the keys shall be a local matter.

The Key Server shall take into account local security policy and device requirements when choosing the algorithms to assign to generated keys. In order for the General-Network-Access key to work with all devices, it has to use algorithms supported by all secure devices in the BACnet internetwork. The Key Server shall report the existence of devices that do not support the minimum security requirements of the local security policy to the local user or a management entity. When determining the algorithms for Device-Master and Distribution keys, the Key Server should select algorithms that are at least as strong as the algorithms used by keys that are to be distributed and protected by these keys.

Note that if a Key Server is configured to provide key sets frequently and if the key sets are given long expiration times, it is possible for the Key Revision to wrap within the expiration time and create conflicts between active Key Revision numbers. Key Server implementations should take care to avoid such situations.

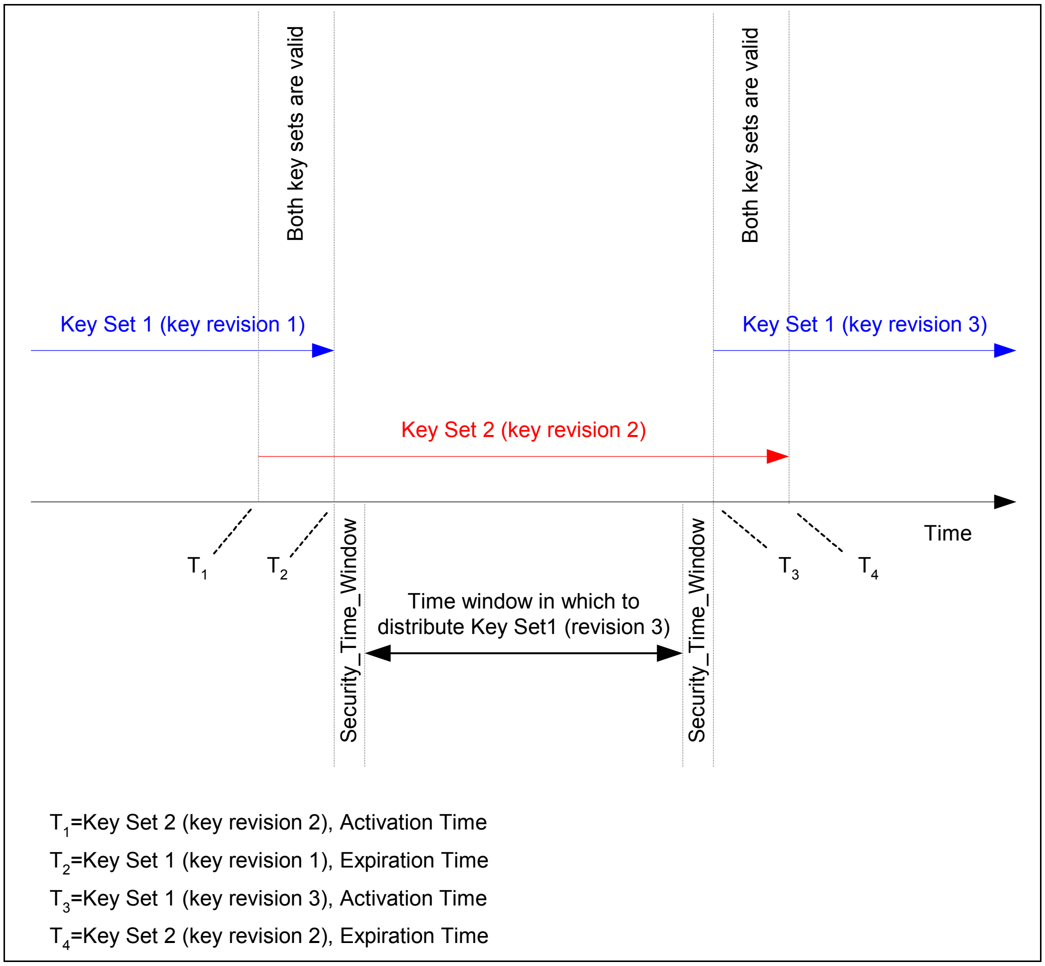

Both key sets are valid
Both key sets are valid
Key Set 1 (key revision 1)
Key Set 1 (key revision 3)
Key Set 2 (key revision 2)
Time
T_1
T_2
T_3
T_4
Security_Time_Window
Time window in which to distribute Key Set1 (revision 3)
Security_Time_Window
T_1=Key Set 2 (key revision 2), Activation Time
T_2=Key Set 1 (key revision 1), Expiration Time
T_3=Key Set 1 (key revision 3), Activation Time
T_4=Key Set 2 (key revision 2), Expiration Time

Figure 24-7: Key Set lifetimes.

The time window in which the Key Server has to distribute a new key set to all devices starts Security_Time_Window seconds after the key set expires and ends Security_Time_Window seconds before the replacement key set becomes active. If the Key Server does not complete the generation and distribution within this time window, some devices will be unable to communicate with each other. See Figure 24-7.

24.22.2 Distribution Method

When the Key Server has generated a new set of keys, the Key Server will increment the Key Revision and distribute the keys to each device. Each device receives the General-Network-Access key plus any other keys it has been authorized to receive. The set of keys are sent encrypted to the device using the Distribution key for that device. Distribution keys are delivered separately using the Device-Master key and need not be distributed as frequently as the key sets.

The Key Server packages the newly generated key set into an Update-Key-Set message, signs and encrypts the message with the device's Distribution key, and sends it to the device. Upon receipt of a valid sequence Update-Key-Set messages, the device shall update its key sets as directed. The device shall start using the new key set as soon as its valid time period will allow.

If the sending of the set of keys to the device is successful, the Key Server shall record the key revisions for both sets for the device. If the Key Server is unable to update the device, it shall re-try the update periodically until successful.

The Key Server shall update each device. If a device cannot be contacted, or the update fails, then the Key Server shall continue with the next device. The order in which the Key Server updates devices is a local matter.

If both of a device's key sets become invalid due to the current time being outside each key set's valid time period, then the device shall cease generating requests, other than Request-Key-Update messages signed with the device's Distribution key, or Device-Master key. The device shall periodically request a new key set from the Last_Key_Server. If the Key Server is on a remote network and the device does not know the MAC address of the router to use, then the device shall use a local MAC broadcast to transmit the Request-Key-Update message. If the device does not have a value for Last_Key_Server, or does not receive an update from the Last_Key_Server, the device shall globally broadcast, or send a sequence of directed broadcasts of, the Request-Key-Update message. The device shall not request a key set more than once every 5 minutes. If, upon power up, a device detects that its key sets have expired, the device shall wait at least 1 minute after power up before requesting a key set in order to allow the Key Server to distribute key sets after power outages.

When the Key Server receives a Request-Key-Update from a device where the Key Revision of the message's Distribution key does not match the recorded Distribution key for the device, the Key Server shall respond with an error code of unknownKeyRevision and then update the device's Distribution key set with an Update-Distribution-Key message.

For devices that the Key Server is unable to provide a key set to, the Key Server shall continue to periodically attempt to update the device using the Update-Key-Set message. The period at which the Key Server retries the key update is a local matter.

The Key Server shall periodically update each device's Distribution key. To update a device's Distribution Key, the Key Server sends the device an Update-Distribution-Key message that is signed and encrypted with the device's Device-Master key.

If a device loses its keys, it can request a new set by sending the Key Server a Request-Key-Update message signed with the device's Device-Master key. To such a request, the Key Server shall respond with an Update-Distribution-Key message.

Under normal operating conditions, after a device has received 2 sets of keys, the Key Server need only update 1 key set by replacing an expired key set with one that will activate at a future time.

24.22.3 Initial Key Distribution

A Key Server shall provide a mechanism for accepting manually input Device-Master keys. For example, the Key Server might have a user interface through which the Device-Master key values can be entered for each device, or the Key Server may be provided with a software tool that will provide keys to the Key Server through a secure channel. This mechanism allows for devices that have a fixed, factory configured, Device-Master key. All such devices shall provide a method of reporting the Device-Master key. For example, a device could have the Device-Master key printed on a tear-off label.

As an alternative to having predefined Device-Master keys, secure devices may optionally support initiation of the Request-Master-Key and execution of the Set-Master-Key messages. It is left up to site policy as to whether or not the use of these inherently insecure services is allowed during site setup.

For installations that do not use the Request-Master-Key and for devices that have factory-fixed Device-Master keys, Key Servers shall support interrogation of devices' Network Security objects in order to determine the encryption and signature algorithms supported before providing key sets.

24.22.4 Key Revision

The Key Revision is only incremented when the data for an existing key is changed (excluding Device Master, Distribution and Installation keys). The creation of new application keys (keys that here-to-for did not exist at all) will also not result in the Key Revision being incremented.

If the Key Server is reconfigured such that the keys that it supplies to a given device are changed (the device is supposed to be given a key it was not previously given, or is no longer supposed to receive a key it already has), the Key Server modifies the device's existing key sets without incrementing the Key Revision. The creation of, or expiration of, Installation Keys shall not result in the incrementing of the Key Revision for the current Key Sets.

24.22.5 Sites Without Key Servers

Secure BACnet sites are able to be deployed without a permanent Key Server. In such cases, a temporary Key Server is used to generate and distribute keys sets that do not expire. To do so, key set 1 shall be provided with an Expiration Date of X'FFFFFFFF' and key set 2 shall be uninitialized and provided with a Key Revision of 0.

Any Key Server that supports such deployment scenarios shall provide a mechanism for documenting all Keys in order to allow the same keys to be used when devices are added to the system at a later date. It is up to the building owner / operator to ensure that the Keys are in the possession of a building representative to allow for future upgrades.

Key Servers that support such deployments shall provide a mechanism for accepting manually input keys. This will allow any such BACnet Key Server to be used to add new devices to an existing installation without the requirement that the original Key Server be used.

24.22.6 Multiple Key Servers

When there are multiple Key Servers in a BACnet installation, the method by which the Key Servers generate new key sets, distribute keys between themselves and determine which Key Server distributes keys to which devices is a local matter. In addition, the method for selecting which Key Server responds to any particular Request-Master-Key service, and the subsequent sharing of generated Device-Master keys between the Key Servers is a local matter. Multiple Key Server systems need to ensure that conflicting keys are not provided to secure devices by different Key Servers. The method for ensuring this is a local matter.

[Add the following definitions to **Clause 3**, p. 1, inserting them in appropriate alphabetical positions and renumbering subsequent clauses.]

. . .

3.1.X1 authentication *- the act of verifying identity*

. . .

3.1.X2 authorization (network security) *- the control of access to network resources based on known identity and access rules*

. . .

3.1.X3 encrypted message *- a message that is wrapped in a security header, signed, and encrypted.*

. . .

3.1.X4 incapable device *- a device that is inherently incapable, or has been configured to be appear to be incapable, of producing or consuming secure BACnet messages. All incapable devices are plain devices.*

. . .

3.1.X5 - inverted network *- a BACnet internetwork where two or more networks are connected by a network with an NPDU size smaller than the networks it joins.*

. . .

3.1.X6 physically insecure *- not physically secure.*

. . .

***3.1.X7 plain device** - a device that does not normally produce or consume secure BACnet messages. All incapable devices are plain devices. However, a plain device that is not an incapable device is capable of producing or consuming secure BACnet messages when communicating with another device that requires it.*

. . .

***3.1.X8 plain network** - a network that does not require signed or encrypted traffic.*

. . .

***3.1.X9 plain message** - a message that is not secured by a BACnet security wrapper.*

. . .

***3.1.X10 physically secure** - a device or network that is protected from physical access by unauthorized individuals.*

. . .

***3.1.X11 signed message** - a message that is wrapped in a security header, signed, and not encrypted.*

. . .

***3.1.X12 secure network** - a network on which all traffic is required to be signed or encrypted.*

. . .

***3.1.X13 trusted** – a term used to refer to devices or networks from which messages are believed to be authentic, either through the use of secure messages or based on the physical security of that device or network.*

. . .

[Change **Clause 5.1,** p. 14]

Information exchanged between two peer application processes is represented in BACnet as an exchange of abstract service primitives, following the ISO conventions contained in the OSI technical report on service conventions, ISO TR 8509. These primitives are used to convey service-specific parameters that are defined in Clauses 13 through 17 and Clause 24. Four service primitives are defined: request, indication, response, and confirm. The information contained in the primitives is conveyed using a variety of protocol data units (PDUs) defined in this standard. In order to make clear which BACnet PDU is being used, the notation will be as follows:

CONF_SERV.request	CONF_SERV.indication	CONF_SERV.response	CONF_SERV.confirm
UNCONF_SERV.request	UNCONF_SERV.indication		
SEGMENT_ACK.request	SEGMENT_ACK.indication		
REJECT.request	REJECT.indication		
ABORT.request	ABORT.indication		
	SEC_ERR.indication		

The designation CONF_SERV indicates that BACnet confirmed service PDUs are being used. Similarly, the designations UNCONF_SERV, SEGMENT_ACK, ERROR, REJECT, and ABORT indicate that unconfirmed service PDUs, segment acknowledge PDUs, error PDUs, reject PDUs, and abort PDUs, respectively, are being used. *The designation SEC_ERR indicates that an error occurred in the BACnet security layer and is being indicated up to the BACnet application program. The format of a SEC_ERR.indication is a local matter.*

An application program that needs to communicate with a remote application process accesses the local BACnet User Element through the API. Some of the API parameters, such as the identity (address) of the device to which the service request is to be sent and protocol control information, is passed directly down to the network or data link layers. The remainder of the parameters make up an application service primitive that is passed from the BACnet User Element to the BACnet ASE. Conceptually, the application service primitive results in the generation of an APDU that becomes the data portion of a network service primitive, which is passed to the network layer through the Network Service Access Point (NSAP). Similarly this request passes down through the lower layers of the protocol stack in the local device. This process is illustrated in Figure 5-2. The message is then transmitted to the remote device, where it is passed up through the protocol stack in the remote device, eventually appearing as an indication primitive passed from the remote BACnet ASE to the remote BACnet User Element. The response from the remote device, if any, returns to the initiator of the service in a similar fashion (see *Clause* 5.5).

In addition to the service primitives and the service specific parameters, the application entity exchanges interface control information (ICI) parameters with the application program via the API. The content of the ICI is dependent upon the service primitive type. The ICI parameters received by the application entity provide the information that is passed on to the lower layers (as ICI across layer interfaces) to help them construct their PDUs. The ICI parameters that are provided by the application entity to the application programs contain information recovered by the lower layers from their respective PDUs.

The following ICI parameters are exchanged with the various service primitives across an API:

'destination_address' (DA): the address of the device(s) intended to receive the service primitive. Its format (device name, network address, etc.) is a local matter. This address may also be a multicast, local broadcast or global broadcast type.

'source_address' (SA): the address of the device from which the service primitive was received. Its format (device name, network address, etc.) is a local matter.

'network_priority' (NP): a four-level network priority parameter described in *Clause* 6.2.2.

'data_expecting_reply' (DER): a Boolean parameter that indicates whether (TRUE) or not (FALSE) a reply service primitive is expected for the service being issued.

'security_parameters' (SEC): The optional security parameters for the request to send, or from the received request. It indicates the level of security (Key Id, Plain/Signed/Encrypted, User Authentication data, End-To-End, etc) and its format is a local matter.

Table 5-1 describes the applicability of the ICI parameters to the service primitives.

Table 5-1. Applicability of ICI Parameters for Abstract Service Primitives

Service Primitive	DA	SA	NP	DER	*SEC*
CONF_SERV.request	Yes	No	Yes	Yes	*Yes*
CONF_SERV.indication	Yes	Yes	Yes	Yes	*Yes*
CONF_SERV.response	Yes	No	Yes	Yes	*Yes*
CONF_SERV.confirm	Yes	Yes	Yes	No	*Yes*
UNCONF_SERV.request	Yes	No	Yes	No	*Yes*
UNCONF_SERV.indication	Yes	Yes	Yes	No	*Yes*
REJECT.request	Yes	No	Yes	No	*Yes*
REJECT.indication	Yes	Yes	Yes	No	*Yes*
SEGMENT_ACK.request	Yes	No	Yes	No	*Yes*
SEGMENT_ACK.indication	Yes	Yes	Yes	No	*Yes*
ABORT.request	Yes	No	Yes	No	*Yes*
ABORT.indication	Yes	Yes	Yes	No	*Yes*
SEC_ERR.indication	*Yes*	*Yes*	*No*	*No*	*Yes*

[Change **Clause 5.1.2**, p. 18]

5.1.2 Unconfirmed Application Services

The client and server model, and the terms "requesting BACnet-user" and "responding BACnet-user," do not apply to unconfirmed services. The terms "sending BACnet-user" and "receiving BACnet-user" do apply, however, and they are used to define the service procedure for unconfirmed services.

In some cases, errors will be encountered when sending unconfirmed requests. While the standard does not provide a mechanism for reporting such errors to the application program, it is acceptable for an implementation to provide such a mechanism, if desired.

[Change **Clause 5.4.4**, p. 26]

5.4.4 State Machine for Requesting BACnet User (client)

5.4.4.1 IDLE

…

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer for which there is no active TSM associated with it,

then enter the IDLE state.

5.4.4.2 SEGMENTED_REQUEST

…

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then stop SegmentTimer; send SEC_ERR.indication with the appropriate security error information to the local application program; and enter the IDLE state.

InsufficientSecurity_Received

If a PDU is received from the network layer and the security parameters do not match the initial request,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE and an 'abort-reason' = INSUFFICIENT_SECURITY describing the security error; send SEC_ERR.indication indicating insufficient security to the local application program; and enter the IDLE state.

DuplicateACK_Received

If a BACnet-SegmentACK-PDU *that has sufficient security parameters and* whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of FALSE,

then restart SegmentTimer and enter the SEGMENTED_REQUEST state to await an acknowledgment.

NewACK_Received

If a BACnet-SegmentACK-PDU *that has sufficient security parameters and* whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there is at least one segment remaining to send,

then set InitialSequenceNumber equal to the 'sequence-number' parameter of the BACnet-SegmentACK-PDU plus one, modulo 256; set ActualWindowSize equal to the 'actual-window-size' parameter of the BACnet-SegmentACK-PDU; restart SegmentTimer; set SegmentRetryCount to zero; call FillWindow

(InitialSequenceNumber) to transmit one or more BACnet-Confirmed-Request-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalACK_Received

If a BACnet-SegmentACK-PDU *that has sufficient security parameters and* whose 'server' parameter is TRUE is received from the network layer and InWindow ('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there are no more segments to send,

then stop SegmentTimer; start RequestTimer; and enter the AWAIT_CONFIRMATION state to await a reply.

Timeout

If SegmentTimer becomes greater than T_{seg} and SegmentRetryCount is less than N_{retry},

then increment SegmentRetryCount; restart SegmentTimer; call FillWindow(InitialSequenceNumber) to retransmit one or more BACnet-Confirmed-Request-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalTimeout

If SegmentTimer becomes greater than T_{seg} and SegmentRetryCount is greater than or equal to N_{retry},

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU *that has sufficient security parameters and* whose 'server' parameter is TRUE is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

SimpleACK_Received

If a BACnet-SimpleACK-PDU *that has sufficient security parameters* is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

UnsegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is FALSE and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

SegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is zero and this device supports segmentation and SentAllSegments is TRUE,

then stop SegmentTimer; compute ActualWindowSize based on the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and on local conditions; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = FALSE, and 'actual-window-size' = ActualWindowSize; start SegmentTimer; set LastSequenceNumber to zero; set InitialSequenceNumber to zero; and enter the SEGMENTED_CONF state to receive the remaining segments. (The method used to determine ActualWindowSize is a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and shall be in the range 1 to 127, inclusive.)

ErrorPDU_Received

If a BACnet-Error-PDU *that has sufficient security parameters* is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

RejectPDU_Received

If a BACnet-Reject-PDU *that has sufficient security parameters* is received from the network layer and SentAllSegments is TRUE,

then stop SegmentTimer; send REJECT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If a BACnet-SimpleACK-PDU, BACnet-ComplexACK-PDU, BACnet-Error-PDU, or BACnet-Reject-PDU *that has sufficient security parameters* is received from the network layer and SentAllSegments is FALSE,

or if a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and this device does not support segmentation,

or if a BACnet-ComplexACK-PDU is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not zero,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

5.4.4.3 AWAIT_CONFIRMATION

In the AWAIT_CONFIRMATION state, the device waits for a response to a BACnet-Confirmed-Request-PDU.

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then stop RequestTimer; send SEC_ERR.indication with the appropriate security error information to the local application program; and enter the IDLE state.

InsufficientSecurity_Received

If a PDU is received from the network layer and the security parameters do not match the initial request,

then stop RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE and an 'abort-reason' = INSUFFICIENT_SECURITY; send

SEC_ERR.indication indicating insufficient security to the local application program; and enter the IDLE state.

SimpleACK_Received

If a BACnet-SimpleACK-PDU *that has sufficient security parameters* is received from the network layer,

then stop RequestTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

UnsegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is FALSE,

then stop RequestTimer; send CONF_SERV.confirm(+) to the local application program; and enter the IDLE state.

SegmentedComplexACK_Received

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is zero and this device supports segmentation,

then stop RequestTimer; compute ActualWindowSize based on the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and on local conditions; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = FALSE, and 'actual-window-size' = ActualWindowSize; start SegmentTimer; set LastSequenceNumber to zero; set InitialSequenceNumber to zero; and enter the SEGMENTED_CONF state to receive the remaining segments. (The method used to determine ActualWindowSize is a local matter, except that the value shall be less than or equal to the 'proposed-window-size' parameter of the received BACnet-ComplexACK-PDU and shall be in the range 1 to 127, inclusive.)

ErrorPDU_Received

If a BACnet-Error-PDU *that has sufficient security parameters* is received from the network layer,

then stop RequestTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

RejectPDU_Received

If a BACnet-Reject-PDU *that has sufficient security parameters* is received from the network layer,

then stop RequestTimer; send REJECT.indication to the local application program; and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is TRUE *that has sufficient security parameters* is received from the network layer,

then stop RequestTimer; send ABORT.indication to the local application program; and enter the IDLE state.

SegmentACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is TRUE *that has sufficient security parameters* is received from the network layer,

then discard the PDU as a duplicate, and re-enter the current state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-ComplexACK-PDU with 'segmented-message' = TRUE and 'sequence-number' not equal to zero or 'segmented-message' = TRUE and this device does not support segmentation) *that has sufficient security parameters* is received from the network layer,

then stop RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

TimeoutUnsegmented

If RequestTimer becomes greater than $\mathbf{T_{out}}$ and RetryCount is less than Number_Of_APDU_Retries and the length of the Confirmed Request APDU is less than or equal to maximum-transmittable-length as determined according to *Clause* 5.2.1,

then stop RequestTimer; increment RetryCount; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU with 'segmented-message' = FALSE; start RequestTimer; and enter the AWAIT_CONFIRMATION state to await a reply.

TimeoutSegmented

If RequestTimer becomes greater than $\mathbf{T_{out}}$ and RetryCount is less than Number_Of_APDU_Retries and the length of the Confirmed-Request APDU is greater than maximum-transmittable-length as determined according to *Clause* 5.2.1,

then stop RequestTimer; increment RetryCount; set SegmentRetryCount to zero; set SentAllSegments to FALSE; start SegmentTimer; set InitialSequenceNumber to zero; set ActualWindowSize to 1; issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit a BACnet-Confirmed-Request-PDU containing the first segment of the message, with 'segmented-message' = TRUE, 'more-follows' = TRUE, and 'sequence-number' = zero; and enter the SEGMENTED_REQUEST state to await an acknowledgment.

FinalTimeout

If RequestTimer becomes greater than $\mathbf{T_{out}}$ and RetryCount is greater than or equal to Number_Of_APDU_Retries,

then stop RequestTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop RequestTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

5.4.4.4 SEGMENTED_CONF

In the SEGMENTED_CONF state, the device waits for one or more segments in response to a BACnet-SegmentACK-PDU.

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then stop SegmentTimer; send SEC_ERR.indication with the appropriate security error information to the local application program; and enter the IDLE state.

InsufficientSecurity_Received

If a PDU is received from the network layer and the security parameters that do not match the initial request, *then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE and an 'abort-reason' = INSUFFICIENT_SECURITY; send SEC_ERR.indication indicating insufficient security to the local application program; and enter the IDLE state.*

NewSegmentReceived_NoSpace

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and the segment cannot be saved due to local conditions,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

NewSegmentReceived

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'more-follows' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'sequence-number' parameter is not equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-ComplexACK-PDU segment; increment LastSequenceNumber, modulo 256; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive additional segments.

LastSegmentOfGroupReceived

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; whose 'more-follows' parameter is TRUE; and whose 'sequence-number' parameter is equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-ComplexACK-PDU segment; increment LastSequenceNumber, modulo 256; set InitialSequenceNumber to LastSequenceNumber; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, server = FALSE, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive additional segments.

LastSegmentOfComplexACK_Received

If a ComplexACK PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'more-follows' parameter is FALSE (i.e., the final segment),

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server = FALSE', and 'actual-window-size' = ActualWindowSize; send CONF_SERV.confirm(+) containing all of the received segments to the local application program; and enter the IDLE state.

SegmentReceivedOutOfOrder

If a BACnet-ComplexACK-PDU *that has sufficient security parameters* is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not equal to LastSequenceNumber plus 1, modulo 256,

then discard the BACnet-ComplexACK-PDU segment; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = TRUE, 'server' = FALSE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; and enter the SEGMENTED_CONF state to receive the remaining segments.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is TRUE *that has sufficient security parameters* is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-SimpleACK-PDU, BACnet-ComplexACK-PDU with 'segmented-message' = FALSE, BACnet-Error-PDU, BACnet-Reject-PDU, or BACnet-SegmentACK-PDU with 'server' = TRUE) *that has sufficient security parameters* is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than T_{seg} times four,

then stop SegmentTimer; send CONF_SERV.confirm(-) to the local application program; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = FALSE; and enter the IDLE state.

[Change **Clause 5.4.5**, p. 31]

5.4.5 State Machine for Responding BACnet User (server)

5.4.5.1 IDLE

…

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then enter the IDLE state.

5.4.5.2 SEGMENTED_REQUEST

…

IncorrectSecurityPdu_Received

If a PDU is received that is not secured with the same settings as the original PDU,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE and 'abort-reason' = INSUFFICIENT_SECURITY; and enter the IDLE state.

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then stop SegmentTimer and enter the IDLE state.

NewSegmentReceived

If a BACnet-Confirmed-Request-PDU *that is secured with the same settings as the original PDU* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'more-follows' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'sequence-number' parameter is not equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; restart SegmentTimer; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

LastSegmentOfGroupReceived

If a BACnet-Confirmed-Request-PDU *that is secured with the same settings as the original PDU* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; whose 'more-follows' parameter is TRUE; and whose 'sequence-number' parameter is equal to InitialSequenceNumber plus ActualWindowSize, modulo 256,

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; set InitialSequenceNumber = LastSequenceNumber; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

LastSegmentOfMessageReceived

If a BACnet-Confirmed-Request-PDU *that is secured with the same settings as the original PDU* is received from the network layer whose 'segmented-message' parameter is TRUE; whose 'sequence-number' parameter is equal to LastSequenceNumber plus 1, modulo 256; and whose 'more-follows' parameter is FALSE (i.e., the final segment),

then save the BACnet-Confirmed-Request-PDU segment; increment LastSequenceNumber, modulo 256; stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; set InitialSequenceNumber = LastSequenceNumber; send CONF_SERV.indication(+) containing all of the received segments to the local application program; start RequestTimer; and enter the AWAIT_RESPONSE state.

SegmentReceivedOutOfOrder

If a BACnet-Confirmed-Request-PDU *that is secured with the same settings as the original PDU* is received from the network layer whose 'segmented-message' parameter is TRUE and whose 'sequence-number' parameter is not equal to LastSequenceNumber plus 1, modulo 256,

then discard the PDU; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = TRUE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; restart SegmentTimer; set InitialSequenceNumber = LastSequenceNumber; and enter the SEGMENTED_REQUEST state to receive the remaining segments.

AbortPDU_Received

If a BACnet-Abort-PDU whose server parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer,

then stop SegmentTimer and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-Confirmed-Request-PDU with 'segmented-message' = FALSE or BACnet-SegmentACK-PDU with 'server' = FALSE) *that is secured with the same settings as the original PDU* is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than $\mathbf{T_{seg}}$ times four,

then stop SegmentTimer and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer, issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE, and enter the IDLE state.

5.4.5.3 AWAIT_RESPONSE

In the AWAIT_RESPONSE state, the device waits for the local application program to respond to a BACnet-Confirmed-Request-PDU. See *Clause* 9.8 for specific considerations in MS/TP networks.

…

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then send SEC_ERR.indication with the appropriate security error information to the local application program; and enter the IDLE state.

IncorrectSecurityPdu_Received

If a PDU is received that is not secured with the same settings as the original PDU,

then discard the PDU, and re-enter the current state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer,

then send ABORT.indication to the local application program; and enter the IDLE state.

DuplicateRequestReceived

If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer,

then discard the PDU as a duplicate request, and re-enter the current state.

DuplicateSegmentReceived

If a BACnet-Confirmed-Request-PDU whose 'segmented-message' parameter is TRUE *that is secured with the same settings as the original PDU* is received from the network layer,

then discard the PDU as a duplicate segment; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-SegmentACK-PDU with 'negative-ACK' = FALSE, 'server' = TRUE, 'sequence-number' = LastSequenceNumber, and 'actual-window-size' = ActualWindowSize; and re-enter the current state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-SegmentACK-PDU whose 'server' parameter is FALSE) *that is secured with the same settings as the original PDU* is received from the network layer,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; send ABORT.indication to the local application program; and enter the IDLE state.

Timeout

If RequestTimer becomes greater than $\mathbf{T_{out}}$**,**

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; send ABORT.indication to the local application program; and enter the IDLE state.

5.4.5.4 SEGMENTED_RESPONSE

In the SEGMENTED_RESPONSE state, the device waits for a BACnet-SegmentACK-PDU for a segment or segments of a BACnet-ComplexACK-PDU.

SecurityError_Received

If a security error is received via an N-REPORT.indication from the network layer,

then stop SegmentTimer; send SEC_ERR.indication with the appropriate security error information to the local application program; and enter the IDLE state.

IncorrectSecurityPdu_Received

If a PDU is received that is not secured with the same settings as the original PDU,

then issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

DuplicateACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer and InWindow('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of FALSE,

then restart SegmentTimer and enter the SEGMENTED_RESPONSE state to await an acknowledgment or timeout.

NewACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer and InWindow('sequence-number' parameter of the BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there is at least one segment remaining to send,

then set InitialSequenceNumber equal to the 'sequence-number' parameter of the BACnet-SegmentACK-PDU plus one, modulo 256; set ActualWindowSize equal to the 'actual-window-size' parameter of the BACnet-SegmentACK-PDU; restart SegmentTimer; set SegmentRetryCount to zero; call FillWindow(InitialSequenceNumber) to issue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit one or more BACnet-ComplexACK-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_RESPONSE state to await an acknowledgment.

FinalACK_Received

If a BACnet-SegmentACK-PDU whose 'server' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer and InWindow('sequence-number' parameter of the

BACnet-SegmentACK-PDU, InitialSequenceNumber) returns a value of TRUE and there are no more segments to send,

then stop SegmentTimer and enter the IDLE state.

Timeout

If SegmentTimer becomes greater than $\mathbf{T_{seg}}$ and SegmentRetryCount is less than Number_Of_APDU_Retries,

then increment SegmentRetryCount; restart SegmentTimer; call FillWindow(InitialSequenceNumber) to reissue an N-UNITDATA.request with 'data_expecting_reply' = TRUE to transmit one or more BACnet-ComplexACK-PDUs containing the next ActualWindowSize segments of the message; and enter the SEGMENTED_RESPONSE state to await an acknowledgment.

FinalTimeout

If SegmentTimer becomes greater than $\mathbf{T_{seg}}$ and SegmentRetryCount is greater than or equal to Number_Of_APDU_Retries,

then stop the SegmentTimer, and enter the IDLE state.

AbortPDU_Received

If a BACnet-Abort-PDU whose 'server' parameter is FALSE *that is secured with the same settings as the original PDU* is received from the network layer,

then stop SegmentTimer; send ABORT.indication to the local application program; and enter the IDLE state.

UnexpectedPDU_Received

If an unexpected PDU (BACnet-Confirmed-Request-PDU) *that is secured with the same settings as the original PDU* is received from the network layer,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

SendAbort

If ABORT.request is received from the local application program,

then stop SegmentTimer; issue an N-UNITDATA.request with 'data_expecting_reply' = FALSE to transmit a BACnet-Abort-PDU with 'server' = TRUE; and enter the IDLE state.

[Change **Clause 6.1**, p. 47]

6.1 Network Layer Service Specification

Conceptually, the BACnet network layer provides an unacknowledged connectionless form of data unit transfer service to the application layer. The primitives associated with the interaction are the N-UNITDATA request and indication, *and the N-REPORT indication*. These primitives provide parameters as follows:

N-UNITDATA.request (
destination_address,
data,
network_priority,
data_expecting_reply,
security_parameters
)

N-UNITDATA.indication (

source_address,
destination_address,
data,
network_priority,
data_expecting_reply,
security_parameters
)

N-REPORT.indication (
peer_address,
error_condition,
error_parameters,
security_parameters
)

The 'destination_address' and 'source_address' parameters provide the logical concatenation of 1) an optional network number, 2) the MAC address appropriate to the underlying LAN technology, and the 3) the link service access point. A network number of X'FFFF' indicates that the message is to be broadcast "globally" to all devices on all currently reachable networks. Currently reachable networks are those networks to which an active connection is already established within the BACnet internet. In particular, a global broadcast shall not trigger any attempts to establish PTP connections. The 'data' parameter is the network service data unit (NSDU) passed down from the application layer and is composed of a fully encoded BACnet APDU. The 'network_priority' is a numeric value used by the network layer in BACnet routers to determine any possible deviations from a first-in-first-out approach to managing the queue of messages awaiting relay. The data_expecting_reply parameter indicates whether (TRUE) or not (FALSE) a reply data unit is expected for the data unit being transferred. *The optional parameter 'security_parameters' contains the security information used to secure the request and context information required for security related N-REPORT.indication primitives to be related to application TSMs .*

Upon receipt of an N-UNITDATA.request primitive from the application layer, the network layer shall attempt to send an NSDU using the procedures described in this clause. Upon receipt of an NSDU from a peer network entity, a network entity shall either 1) send the NSDU to its destination on a directly connected network, 2) send the NSDU to the next BACnet router en route to its destination, and/or 3) if the destination address matches that of one of its own application entities, issue an N-UNITDATA.indication primitive to the appropriate entity in its own application layer to signal the arrival of the NSDU.

The N-REPORT.indication primitive is used by the local network layer to indicate failures to transmit N-UNITDATA.requests to peer devices. The errors may be locally detected error conditions, or error conditions reported by a peer device via a network layer message. This primitive is used extensively by the network security wrapper to indicate security errors up the stack. The 'peer_address' parameter is of the same form as the 'destination_address' or 'source_address' parameters of the N-UNITDATA primitives and indicates the peer with which the error condition arose. The optional parameter 'security_parameters' conveys information describing the security failure and context required to relate the error to a previous N-UNITDATA.request or N-UNITDATA.indication primitive.

[Change **Clause 6.2.4**, p. 52]

6.2.4 Network Layer Message Type

If Bit 7 of the control octet described in *Clause* 6.2.2 is 1, a message type octet shall be present as shown in Figure 6-1. The following message types are indicated:

X'00': Who-Is-Router-To-Network
X'01': I-Am-Router-To-Network
X'02': I-Could-Be-Router-To-Network
X'03': Reject-Message-To-Network
X'04': Router-Busy-To-Network
X'05': Router-Available-To-Network

X'06': Initialize-Routing-Table
X'07': Initialize-Routing-Table-Ack
X'08': Establish-Connection-To-Network
X'09': Disconnect-Connection-To-Network
X'0A': Challenge-Request
X'0B': Security-Payload
X'0C': Security-Response
X'0D': Request-Key-Update
X'0E': Update-Key-Set
X'0F': Update-Distribution-Key
X'10': Request-Master-Key
X'11': Set-Master-Key
X'12': What-Is-Network-Number
X'13': Network-Number-Is
X'~~0A~~*14*' to X'7F': Reserved for use by ASHRAE
X'80' to X'FF': Available for vendor proprietary messages

Figures 6-5 through 6-10 provide examples of NPDUs containing network layer messages.

[Change **Clause 6.4.4**, p. 54]

6.4.4 Reject-Message-To-Network

This message is indicated by a Message Type of X'03' followed by an octet indicating the reason for the rejection and a 2-octet network number (see Figure 6-7). It is directed to the node that originated the message being rejected, as indicated by the source address information in that message. The rejection reason octet shall contain an unsigned integer with one of the following values:

0: Other error.

1: The router is not directly connected to DNET and cannot find a router to DNET on any directly connected network using Who-Is-Router-To-Network messages.

2: The router is busy and unable to accept messages for the specified DNET at the present time.

3: It is an unknown network layer message type.

4: The message is too long to be routed to this DNET.

5. The source message was rejected due to a BACnet security error and that error cannot be forwarded to the source device. See Clause 24.12.1.1 for more details on the generation of Reject-Message-To-Network messages indicating this reason.

6. The source message was rejected due to errors in the addressing. The length of the DADR or SADR was determined to be invalid.

[Insert new **Clauses 6.4.11 through 6.4.20**, p. 56]

6.4.11 Challenge-Request

This message is indicated by a Message Type of X'0A'. It is described in Clause 24.

6.4.12 Security-Payload

This message is indicated by a Message Type of X'0B'. It is described in Clause 24.

6.4.13 Security-Response

This message is indicated by a Message Type of X'0C'. It is described in Clause 24.

6.4.14 Request-Key-Update

This message is indicated by a Message Type of X'0D'. It is described in Clause 24.

6.4.15 Update-Key-Set

This message is indicated by a Message Type of X'0E'. It is described in Clause 24.

6.4.16 Update-Distribution-Key

This message is indicated by a Message Type of X'0F'. It is described in Clause 24.

6.4.17 Request-Master-Key

This message is indicated by a Message Type of X'10'. It is described in Clause 24.

6.4.18 Set-Master-Key

This message is indicated by a Message Type of X'11'. It is described in Clause 24.

6.4.19 What-Is-Network-Number

This message is indicated by a Message Type of X'12'. It is used to request the local network number from other devices on the local network. This message may be transmitted with a local broadcast or a local unicast address. This message shall never be routed. Devices shall ignore What-Is-Network-Number messages that contain SNET/SADR or DNET/DADR information in the NPCI.

Upon receipt of a What-Is-Network-Number message, a device that knows the local network number shall transmit a local broadcast Network-Number-Is message back to the source device. If the What-Is-Network-Number message was broadcast, then a non-routing device may optionally wait for up to 10 seconds before sending the Network-Number-Is message. If during that time, a different device broadcasts the Network-Number-Is message, the non-routing device may choose to not send a Network-Number-Is message..

Upon receipt of a What-Is-Network-Number message, a device that does not know the local network number shall discard the message.

A device shall cache its local network number, and not repeatedly issue this service.

6.4.20 Network-Number-Is

This message is indicated by a Message Type of X'13' followed by a 2-octet network number (most significant octet first), followed by a 1-octet flag, where a value of 1 indicates that the network number was configured, and a value of 0 indicates that the network number was learned by receipt of a previous Network-Number-Is message. This message is used to indicate the local network number to other devices on the local network. It shall be transmitted with a local broadcast address, and shall never be routed. Devices shall ignore Network-Number-Is messages that contain SNET/SADR or DNET/DADR information in the NPCI or that are sent with a local unicast address.

For a device that has not been configured to know its network number, when it receives this message indicating a configured network number, it shall set its network number from this message. If it receives this message indicating a learned network number, then it shall set its network number only if it has not previously received a message indicating a configured network number. If a device resets, it shall reduce the quality of its last received network number to "learned".

For a device that has been configured to know its network number, when it receives this message indicating a configured network number that is in conflict with its configuration, it should report the conflict to a local or remote management entity.

Upon startup, routers that have been configured to know their network numbers shall broadcast out each port a Network-Number-Is message containing the network number for the port and indicate that this number is configured, not learned.

[Change Clause **6.5.2.2**, p. 57]

6.5.2.2 Receipt of Local Network Layer Messages

If the control NPCI octet indicates the absence of a DNET field or a DNET field is present and contains the global broadcast address X'FFFF', the NE shall attempt to interpret the network layer message. ~~If the DNET field is present and the NE resides in a routing node and the network layer message can be interpreted, then the NE shall take the actions specified in 6.5.4.~~ If the *DNET field is absent and the* message cannot be interpreted, a Reject-Message-To-Network shall be returned to the device that sent the message.

If the DNET is present and not equal to the global broadcast address X'FFFF' and the NE resides in a non-routing node, the data link data shall be discarded and no further action taken. If the DNET is present and the NE resides in a BACnet router, the NE shall take the actions specified in *Clause* 6.5.4.

[Change **Clause 12.11.18**, p 182]

12.11.18 Max_APDU_Length_Accepted

This property, of type Unsigned, is the maximum number of octets that may be contained in a single, indivisible application layer protocol data unit. The value of this property shall be greater than or equal to 50. The value of this property is also constrained by the underlying data link technology. See Clauses 6 through 11.

If the value of this property is not encodable in the 'Max APDU Length Accepted' parameter of a ConfirmedRequest-PDU, then the value encoded shall be the highest encodable value less than the value of this property. In such cases, a responding device may ignore the encoded value in favor of the value of this property, if it is known.

[Insert new **Clause 12.X**, p. 288]

12.X Network Security Object Type

The Network Security object type defines a standardized object whose properties represent the externally visible network security settings and status of a BACnet Device. Secure BACnet devices shall contain exactly one Network Security object and they shall have an instance of 1. A detailed description of BACnet security and secure BACnet devices can be found in Clause 24.

Network Security object type and its properties are summarized in Table 12-Y and described in detail in this subclause.

Operations on the Network Security object shall always be deemed to have sufficient authorization if the request is secured with an Installation key.

Table 12-Y. Properties of the Network Security Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Base_Device_Security_Policy	BACnetSecurityLevel	W
Network_Access_Security_Policies	BACnetARRAY[N] of BACnetNetworkSecurityPolicy	W
Security_Time_Window	Unsigned	W
Packet_Reorder_Time	Unsigned	W
Distribution_Key_Revision	Unsigned8	R
Key_Sets	BACnetARRAY[2] of BACnetSecurityKeySet	R
Last_Key_Server	BACnetAddressBinding	W
Security_PDU_Timeout	Unsigned16	W
Update_Key_Set_Timeout	Unsigned16	R
Supported_Security_Algorithms	List of Unsigned8	R
Do_Not_Hide	BOOLEAN	W
Profile_Name	CharacterString	O

12.X.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

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 property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be NETWORK_SECURITY.

12.X.4 Description

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

12.X.5 Base_Device_Security_Policy

This writable property, of type BACnetSecurityLevel, specifies the minimum level of security that the device requires allowing client devices to know the level of security to use when communicating with the device.

While devices may require higher security levels for some operations, this property shall be readable using the security level defined by this property.

12.X.6 Network_Access_Security_Policies

This writable property, of type BACnetArray of BACnetNetworkSecurityPolicy, specifies the security policy for each network directly connected to the device. It specifies the level of security that the device should use for network infrastructure services, such as Who-Is, I-Am, Who-Is-Router, etc. This array shall have 1 entry for each network port.

The Port ID field shall correspond to the Port ID of the associated network as described in Clause 6. For non-routing nodes, this value shall be 0.

This property shall be readable via the base security level for the device.

12.X.7 Security_Time_Window

This writable property, of type Unsigned, specifies the security time window for the device in seconds. The recommended default value for this property is 180 (3 minutes). The property shall be restricted to the range 1 through 600.

12.X.8 Packet_Reorder_Time

This writable property, of type Unsigned, specifies the packet reorder time, in milliseconds, used by the device for validating Message Ids. The recommended default value for this property is 500 (0.5 seconds). The property shall be restricted to the range 1 through 3000.

12.X.9 Distribution_Key_Revision

This read-only property, of type Unsigned8, identifies the device's Distribution key revision. This property shall be 0 if the device does not have a Distribution key.

12.X.10 Key_Sets

This read only property, of type BACnetArray of BACnetSecurityKeySet, describes the contents of the device's 2 key sets. The actual key values are not included in the contents of this property. When a key set has not been provided, the key-revision field shall be set to 0, the key-ids field shall be empty, and the activation-time and expiration-time fields shall contain all wildcard values.

12.X.11 Last_Key_Server

This writable property, of type BACnetAddressBinding, specifies the device identifier and address of the last Key Server that successfully updated a security key in the device. If no Key Server has updated the keys sets in the device, the deviceObjectIdentifier field shall contain 4194303 in the instance part, the network-number field shall be 0, and the mac-address field shall be empty.

This property is writable in order to allow a Key Server address to be provided to the secure device before it has received a Device-Master key. This allows the secure device to be directed to the Key Server in a legacy environment where globally broadcast Request-Key-Update messages will not be routed. A device may make this property read-only once a Device-Master key has been received.

12.X.12 Security_PDU_Timeout

This writable property, of type Unsigned16, specifies the length of time, in milliseconds, the device waits for a security response. For the application TSM to work correctly, this value should be configured to be less than the APDU_Segment_Timeout value in the Device object.

12.X.13 Update_Key_Set_Timeout

This property, of type Unsigned16, indicates the maximum amount of time, in milliseconds, that the device will take to respond to an Update-Key-Set message. This value added to the device Max_APDU_Timeout results in the amount of time that a Key Server shall wait for a Security-Response for an Update-Key-Set message. The use of Max_APDU_Timeout is to allow for network delay; whereas the Update_Key_Set_Timeout provides for the actual time the device will need to apply the keys to its key set.

12.X.14 Supported_Security_Algorithms

This read-only property, of type list of Unsigned8, identifies the encryption and signature algorithm pairs that the device supports. See Clause 24.21.1 for a list of defined values.

12.X.15 Do_Not_Hide

This writable property, of type BOOLEAN, indicates whether or not the device is allowed to ignore certain network security error conditions. When True, the device is required to return errors in all of the conditions described in Clause 24.3.

It is recommended that this property default to True.

12.X.16 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 document named by the remainder of the profile name. 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. The definition of the profiles themselves is outside the scope of this standard.

[Change **Table 12-13**, p. 180]

Table 12-13. Properties of the Device Object Type

Property Identifier	Property Datatype	Conformance Code
. . . ~~List_Of_Session_Keys~~ . . .	. . . ~~List of BACnetSessionKey~~ . . .	. . . ~~O~~ . . .

[Delete **Clause 12.11.30**, p. 184]

~~12.11.30 List_Of_Session_Keys~~

~~This property is a List of BACnetSessionKey each of which is one of the cryptographic keys used to communicate with other security-conscious BACnet Devices. This property shall not be readable or writable by any device except a device designated the "Key Server." A session key shall consist of a 56-bit encryption key and a BACnet Address of the peer with which secure communications is requested.~~

[Change **Clause 18**, p. 393]

. . .

18.5 Error Class – SECURITY

This Error Class pertains to problems related to the execution of ~~security.~~ *security services. Without exception, these errors signal the inability of the responding BACnet-user to carry out the desired service in its entirety and are thus "fatal."*

~~**18.5.1 AUTHENTICATION_FAILED** - The message being authenticated was not generated by the service provider.~~

~~**18.5.2 CHARACTER_SET_NOT_SUPPORTED** - A character string value was encountered that is not a supported character set.~~

~~**18.5.3 INCOMPATIBLE_SECURITY_LEVELS** - The two clients do not have the same security authorization.~~

~~**18.5.4 INVALID_OPERATOR_NAME** - The 'Operator Name' is not associated with any known operators.~~

~~**18.5.5 KEY_GENERATION_ERROR** - The key server was unable to generate a Session Key (SK).~~

18.5.6 PASSWORD_FAILURE – ~~The password was incorrect.~~ *The 'Operator Name' and 'Operator Password' did not associate correctly.*

~~**18.5.7 SECURITY_NOT_SUPPORTED** - The remote client does not support any BACnet security mechanisms.~~

~~**18.5.8 TIMEOUT** - The APDU with the expected Invoke ID was not received before the waiting time expired.~~

18.5.9 OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

18.5.X1 READ_ACCESS_DENIED *- The requesting device did not provide security credentials of sufficient authorization to allow the request.*

18.5.X2 WRITE_ACCESS_DENIED *- The requesting device did not provide security credentials of sufficient authorization to allow the request.*

18.5.X3 SUCCESS *- The security operation was successful.*

18.5.X4 ACCESS_DENIED *- The requesting device did not provide security credentials of sufficient authorization to allow the request. This error is used when READ_ACCESS_DENIED and WRITE_ACCESS_DENIED are not appropriate.*

18.5.X5 BAD_DESTINATION_ADDRESS *- The destination address in the request does not match that of the receiver.*

18.5.X6 BAD_DESTINATION_DEVICE_ID *- The Destination Device Instance in the security wrapper does not match the local device instance.*

18.5.X7 BAD_SIGNATURE *- The signature in a secure packet was incorrect.*

18.5.X8 BAD_SOURCE_ADDRESS *- The source address in a secure packet was incorrect or missing.*

18.5.X9 BAD_TIMESTAMP *- The timestamp in a secure packet was not within the allowable timestamp window of the receiver.*

18.5.X10 CANNOT_USE_KEY *- A key was provided to the device via an Update-Key-Set or Update-Distribution-Key service that is based on an algorithm that the device does not support.*

18.5.X11 CANNOT_VERIFY_MESSAGE_ID *- A device could not accurately ascertain whether it sent a challenged message or not.*

18.5.X12 CORRECT_KEY_REVISION *- The device's key sets are current.*

18.5.X13 DESTINATION_DEVICE_ID_REQUIRED *- The Destination Device Instance in the security header of a unicast message had the value 4194303 and the destination device requires this value to be set correctly for the operation requested.*

18.5.X14 DUPLICATE_MESSAGE *- A message with the provided Message Id has already been received from the source device within the security time window.*

18.5.X15 ENCRYPTION_NOT_CONFIGURED *- The device is not configured to accept encrypted messages.*

18.5.X16 ENCRYPTION_REQUIRED *- The device requires encryption for the requested operation.*

18.5.X17 INCORRECT_KEY *- The key provided to secure the message does not indicate sufficient authority to perform the requested operation.*

18.5.X18 INVALID_KEY_DATA *- A key was received that contained invalid data.*

18.5.X19 KEY_UPDATE_IN_PROGRESS *- A key update is already in progress.*

18.5.X20 MALFORMED_MESSAGE *- The message size is invalid, or security parameters are missing or malformed.*

18.5.X21 NOT_KEY_SERVER *- A device received a request that only a Key Server configured to service the message source could fulfill.*

18.5.X22 SECURITY_NOT_CONFIGURED *- The device is not configured for security on the receiving port.*

18.5.X23 SOURCE_SECURITY_REQUIRED *- The operation requested requires that the source secure or encrypt the request.*

18.5.X24 TOO_MANY_KEYS *- The device cannot be configured with the number of keys provided for the key set.*

18.5.X25 UNKNOWN_AUTHENTICATION_TYPE *- The authentication method in a secure message is unknown to the receiving device.*

18.5.X26 UNKNOWN_KEY *- The key used to secure message is unknown to the receiving device.*

18.5.X27 UNKNOWN_KEY_REVISION *- The key revision used to secure message is unknown to the receiving device.*

18.5.X28 UNKNOWN_SOURCE_MESSAGE *- The device did not send the challenged message.*

[Change **Clause 18.7,** p. 394**]**

18.7 Error Class – COMMUNICATIONS

This Error Class pertains to problems related to network communications. These codes indicate problems reported by a remote device in abort and reject PDUs, or they indicate problems detected internally. These error codes are stored in properties of objects whose operation involves the network communications, such as the Trend Log object's Log_Buffer property. This Error Class shall not be conveyed in error PDUs.

…

18.7.X1 NOT_ROUTER_TO_DNET *– A Reject-Message-To-Network with reason 1 was returned in response to a network request.*

18.7.X2 ROUTER_BUSY *– A network request failed due to a router en-route being busy.*

18.7.X3 UNKNOWN_NETWORK_MESSAGE *– A network request failed that relied on a network message unknown to the receiver.*

18.7.X4 MESSAGE_TOO_LONG *– A network request failed due a message that was too long to make it to its destination.*

18.7.X5 SECURITY_ERROR *– A network request failed due a security error en-route.*

18.7.X6 ADDRESSING_ERROR *- A network request failed due to an addressing error.*

18.7.X7 WRITE_BDT_FAILED *– A Write-Broadcast-Distribution-Table request failed.*

18.7.X8 READ_BDT_FAILED *– A Read-Broadcast-Distribution-Table request failed.*

18.7.X9 REGISTER_FOREIGN_DEVICE_FAILED *– A Register-Foreign-Device request failed.*

18.7.X10 READ_FDT_FAILED *– A Read-Foreign-Device-Table request failed.*

18.7.X11 DELETE_FDT_ENTRY_FAILED *– A Delete-Foreign-Device-Table-Entry request failed.*

18.7.X12 DISTRIBUTE_BROADCAST_FAILED *– A broadcast network request failed due to a failure of a Distribute-Broadcast-To-Network.*

18.7.X13 ABORT_SECURITY_ERROR *- The Transaction is aborted due to receipt of a security error.*

18.7.X14 ABORT_INSUFFICIENT_SECURITY - The transaction is aborted due to receipt of a PDU secured differently than the original PDU of the transaction.

[Change **Clause 18.10**, p. 396]

18.10 Abort Reason

…

18.10.X1 SECURITY_ERROR *- The Transaction is aborted due to receipt of a security error.*

18.10.X2 INSUFFICIENT_SECURITY *- The transaction is aborted due to receipt of a PDU secured differently than the original PDU of the transaction.*

[Change **Clause 21**, p. 431]

```
. . .
********************* Confirmed Service Productions ********************

BACnetConfirmedServiceChoice ::= ENUMERATED {
. . .
-- Virtual Terminal Services
    vtOpen                      (21),
    vtClose                     (22),
    vtData                      (23),

~~-- Security Services~~
--  authenticate                (24),     This value was deleted in version 1 revision 11
--  requestKey                  (25),     This value was deleted in version 1 revision 11
```

-- Services added after 1995
-- readRange (26) see Object Access Services
-- lifeSafetyOperation (27) see Alarm and Event Services
-- subscribeCOVProperty (28) see Alarm and Event Services
-- getEventInformation (29) see Alarm and Event Services
}
-- Other services to be added as they are defined. All enumeration values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer or UnconfirmedPrivateTransfer
-- services. See Clause 23.

. . .

BACnet-Confirmed-Service-Request ::= CHOICE {
. . .
-- Virtual Terminal Services
vtOpen [21] VT-Open-Request,
vtClose [22] VT-Close-Request,
vtData [23] VT-Data-Request,

-- Security Services
~~authenticate [24] Authenticate-Request,~~
~~requestKey [25] RequestKey-Request,~~
-- *Removed in Version 1, Revision 11* *[24],*
-- *Removed in Version 1, Revision 11* *[25],*

-- Services added after 1995
-- readRange [26] see Object Access Services
-- lifeSafetyOperation [27] see Alarm and Event Services
-- subscribeCOVProperty [28] see Alarm and Event Services
-- getEventInformation [29] see Alarm and Event Services
}
-- Context-specific tags 0..29 are NOT used in the encoding. The tag number is transferred as the service-choice parameter
-- in the BACnet-Confirmed-Request-PDU.
--
-- Other services will be added as they are defined. *Tag numbers 24 and 25 are not used.* All choice values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer service. See Clause 23.

. . .

BACnet-Confirmed-Service-ACK ::= CHOICE {
. . .
-- Virtual Terminal Services
vtOpen [21] VT-Open-ACK,
vtData [23] VT-Data-ACK,

~~-- Security Services~~
~~authenticate [24] Authenticate-ACK~~
-- *Removed in Version 1, Revision 11* *[24],*

-- Context-specific tags 3..29 are NOT used in the encoding. The tag number is transferred as the service-ACK-choice
-- parameter in the BACnet-ComplexACK-PDU.
--
-- Other services to be added as they are defined. *Tag number 24 is not used.* All choice values in this production are reserved for definition by
-- ASHRAE. Proprietary extensions are made by using the ConfirmedPrivateTransfer service.

-- See Clause 23.
    }

. . .

~~********************* Confirmed Security Services ********************~~

~~**Authenticate-Request** ::= SEQUENCE {~~
~~pseudoRandomNumber [0] Unsigned32,~~
~~expectedInvokeID [1] Unsigned8 OPTIONAL,~~
~~operatorName [2] CharacterString OPTIONAL,~~
~~operatorPassword [3] CharacterString (SIZE (1..20)) OPTIONAL,~~
~~startEncipheredSession [4] BOOLEAN OPTIONAL~~
~~}~~

~~**Authenticate-ACK** ::= SEQUENCE {~~
~~modifiedRandomNumber Unsigned32~~
~~}~~

~~**RequestKey-Request** ::= SEQUENCE {~~
~~requestingDeviceIdentifier BACnetObjectIdentifier,~~
~~requestingDeviceAddress BACnetAddress,~~
~~remoteDeviceIdentifier BACnetObjectIdentifier,~~
~~remoteDeviceAddress BACnetAddress~~
~~}~~

. . .

BACnet-Error ::= CHOICE {
    other [127] Error,
. . .
-- Virtual Terminal Services
    vtOpen [21] Error,
    vtClose [22] VTClose-Error,
    vtData [23] Error,

~~-- Security Services~~
~~authenticate [24] Error,~~
~~requestKey [25] Error~~
-- Removed in Version 1 Revision 11 *[24],*
-- Removed in Version1 Revision 11 *[25],*

-- Services added after 1995
    -- readRange [26] see Object Access Services
    -- lifeSafetyOperation [27] see Alarm and Event Services
    -- subscribeCOVProperty [28] see Alarm and Event Services
    -- getEventInformation [29] see Alarm and Event Services
    }
-- Context-specific tags 0..29 and 127 are NOT used in the encoding. The tag number is transferred as the error-choice
-- parameter in the BACnet-Error-PDU.
--
-- Other services to be added as they are defined. *Tag numbers 24 and 25 are not used.* All choice values in this production are reserved for definition by
-- ASHRAE. See Clause 23.

. . .

```
Error ::= SEQUENCE {
-- NOTE: The valid combinations of error-class and error-code are defined in Clause 18.
        error-class     ENUMERATED {
                        device          (0),
                        object          (1),
                        property        (2),
                        resources       (3),
                        security        (4),
                        services        (5),
                        vt              (6),
                        ...
                        },
        -- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
        -- 64-65535 may be used by others subject to the procedures and constraints described
        -- in Clause 23.

        error-code      ENUMERATED {
                        other                                   (0),
                        abort-buffer-overflow                   (51),
                        abort-insufficient-security             (135),
                        abort-invalid-apdu-in-this-state        (52),
                        abort-preempted-by-higher-priority-task (53),
                        abort-security-error                    (136),
                        abort-segmentation-not-supported        (54),
                        abort-proprietary                       (55),
                        abort-other                             (56),
                        access-denied                           (85),
                        addressing-error                        (115),
~~                      authentication-failed                   (1),~~
                        -- this enumeration was removed         (1),
                        bad-destination-address                 (86),
                        bad-destination-device-id               (87),
                        bad-signature                           (88),
                        bad-source-address                      (89),
                        bad-timestamp                           (90),
                        cannot-use-key                          (91),
                        cannot-verify-message-id                (92),
                        character-set-not-supported             (41),
                        configuration-in-progress               (2),
                        correct-key-revision                    (93),
                        datatype-not-supported                  (47),
                        delete-fdt-entry-failed                 (120),
                        device-busy                             (3),
                        destination-device-id-required          (94),
                        distribute-broadcast-failed             (121),
                        duplicate-message                       (95),
                        duplicate-name                          (48),
                        duplicate-object-id                     (49),
                        dynamic-creation-not-supported          (4),
                        encryption-not-configured               (96),
                        encryption-required                     (97),
                        file-access-denied                      (5),
~~                      incompatible-security-levels            (6),~~
                        -- this enumeration was removed         (6),
```

	inconsistent-parameters	(7),
	inconsistent-selection-criterion	(8),
	incorrect-key	*(98),*
	invalid-array-index	(42),
	…	
	invalid-file-start-position	(11),
	~~invalid-operator-name~~	~~(12),~~
	-- this enumeration was removed	*(12),*
	invalid-key-data	*(99),*
	invalid-parameter-data-type	(13),
	invalid-time-stamp	(14),
	~~key-generation-error~~	~~(15),~~
	-- this enumeration was removed	*(15),*
	key-update-in-progress	*(100),*
	log-buffer-full	(75),
	logged-value-purged	(76),
	malformed-message	*(101),*
	message-too-long	*(113),*
	missing-required-parameter	(16),
	…	
	not-configured-for-triggered-logging	(78),
	not-key-server	*(102),*
	not-router-to-dnet	*(110),*
	object-deletion-not-permitted	(23),
	…	
	read-access-denied	(27),
	read-bdt-failed	*(117),*
	read-fdt-failed	*(119),*
	register-foreign-device-failed	*(118),*
	reject-buffer-overflow	(59),
	…	
	reject-other	(69),
	router-busy	*(111),*
	security-error	*(114),*
	security-not-configured	*(103),*
	~~security-not-supported~~	~~(28),~~
	-- this enumeration removed	*(28),*
	service-request-denied	(29),
	source-security-required	*(104),*
	success	*(84),*
	timeout	(30),
	too-many-keys	*(105),*
	unknown-authentication-type	*(106),*
	unknown-key	*(107),*
	unknown-key-revision	*(108),*
	unknown-network-message	*(112),*
	unknown-object	(31),
	unknown-property	(32),
	-- this enumeration was removed	(33),
	unknown-source-message	*(109),*
	unknown-vt-class	(34),
	…	
	write-access-denied	(40),
	write-bdt-failed	*(116),*
-- see	character-set-not-supported	(41),

```
                    …
                    --see   communication-disabled              (83),
                    -- see  success                             (84),
                    -- see  access-denied                       (85),
                    -- see  bad-destination-address             (86),
                    -- see  bad-destination-device-id           (87),
                    -- see  bad-signature                       (88),
                    -- see  bad-source-address                  (89),
                    -- see  bad-timestamp                       (90),
                    -- see  cannot-use-key                      (91),
                    -- see  cannot-verify-message-id            (92),
                    -- see  correct-key-revision                (93),
                    -- see  destination-device-id-required      (94),
                    -- see  duplicate-message                   (95),
                    -- see  encryption-not-configured           (96),
                    -- see  encryption-required                 (97),
                    -- see  incorrect-key                       (98),
                    -- see  invalid-key-data                    (99),
                    -- see  key-update-in-progress              (100),
                    -- see  malformed-message                   (101),
                    -- see  not-key-server                      (102),
                    -- see  security-not-configured             (103),
                    -- see  source-security-required            (104),
                    -- see  too-many-keys                       (105),
                    -- see  unknown-authentication-type         (106),
                    -- see  unknown-key                         (107),
                    -- see  unknown-key-revision                (108),
                    -- see  unknown-source-message              (109),
                    -- see  not-router-to-dnet                  (110),
                    -- see  router-busy                         (111),
                    -- see  unknown-network-message             (112),
                    -- see  message-too-long                    (113),
                    -- see  security-error                      (114),
                    -- see  addressing-error                    (115),
                    -- see  write-bdt-failed                    (116),
                    -- see  read-bdt-failed                     (117),
                    -- see  register- foreign-device-failed     (118),
                    -- see  read-fdt-failed                     (119),
                    -- see  delete-fdt-entry-failed             (120),
                    -- see  distribute-broadcast-failed         (121),
                    ...
                    -- see  abort-insufficient-security         (135),
                    -- see  abort-security-error                (136)
                  }
                  -- Enumerated values 0-255 are reserved for definition by ASHRAE. Enumerated values
                  -- 256-65535 may be used by others subject to the procedures and constraints described
                  -- in Clause 23.
}

  …

  BACnetAbortReason ::= ENUMERATED {
      other                             (0),
      buffer-overflow                   (1),
      invalid-apdu-in-this-state        (2),
      preempted-by-higher-priority-task (3),
      segmentation-not-supported      (4),
      security-error                  (5),
      insufficient-security           (6),
      ...
      }
      -- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values 64-255
      -- may be used by others subject to the procedures and constraints described in Clause 23.

  …

  BACnetNetworkSecurityPolicy ::= SEQUENCE {
      port-id                     [0] Unsigned8,
      security-level              [1] BACnetSecurityPolicy
      }

  BACnetKeyIdentifier ::= SEQUENCE {
      algorithm                   [0] Unsigned8,
      key-id                      [1] Unsigned8
      }
  . . .

  BACnetSecurityKeySet ::= SEQUENCE {
      key-revision                [0] Unsigned8,          -- 0 if key set is not configured
      activation-time             [1] BACnetDateTime, -- UTC time, all wild if unknown
      expiration-time             [2] BACnetDateTime, -- UTC time, all wild if infinite
      key-ids                     [3] SEQUENCE OF BACnetKeyIdentifier
      }

  BACnetSecurityLevel ::= ENUMERATED {
      incapable                   (0),        -- indicates that the device is configured to not use security
      plain                       (1),
      signed                      (2),
      encrypted                   (3),
      signed-end-to-end           (4),
      encrypted-end-to-end        (5)
      }

  BACnetSecurityPolicy ::= ENUMERATED {
      plain-non-trusted           (0),
      plain-trusted               (1),
      signed-trusted              (2),
      encrypted-trusted           (3)
      }


  BACnetObjectType ::= ENUMERATED {
              …
              multi-state-value           (19),
              network-security            (38),
              notification-class          (15),
              …
              -- see access-door          (30),
              -- see network-security     (38),
              ...
              }
```

BACnetObjectTypesSupported ::= BIT STRING {
…
-- multi-state-value (19),
-- *network-security (38),*
notification-class (15),
…
access-door ~~(30)~~*(30),*
-- *Objects added after 2008*
network-security (38)
}

BACnetPropertyIdentifier ::= ENUMERATED {

backup-failure-timeout (153),
base-device-security-policy (327),
bias (14),
...
direct-reading (156),
distribution-key-revision (328),
do-not-hide (329),
effective-period (32),
...
-- issue-confirmed-notifications (51), This property was deleted in version 1 revision 4.
key-sets (330),
last-key-server (331),
last-notify-record (173),
...
list-of-object-property-references (54),
~~list-of-session-keys (55),~~
-- *list-of-session-keys (55), This property was deleted in version 1 revision 11*
local-date (56),
...
modification-date (71),
network-access-security-policies (332),
notification-class (17), -- renamed from previous version
...
-- see event-parameters (83),
packet-reorder-time (333),
polarity (84),
...
schedule-default (174),
security-pdu-timeout (334),
security-time-window (335),
segmentation-supported (107),
...
stop-when-full (144),
supported-security-algorithms (336),
system-status (112),
…
update-interval (118),
update-key-set-timeout (337),
update-time (189),
...
-- *see base-device-security-policy (327),*
-- *see distribution-key-revision (328),*

-- see do-not-hide (329),
-- see key-sets (330),
-- see last-key-server (331),
-- see network-access-security-polices (332),
-- see packet-reorder-time (333),
-- see security-pdu-timeout (334),
-- see security-time-window (335),
-- see supported-security-algorithms (336),
-- see update-key-set-timeout (337),
...
}

BACnetServicesSupported ::= BIT STRING {
. . .
-- Virtual Terminal Services
vtOpen (21),
vtClose (22),
vtData (23),

~~-- Security Services~~
~~authenticate (24),~~
~~requestKey (25),~~
-- Removed Service (24),
-- Removed Service (25),

-- Unconfirmed Services
i-Am (26),
i-Have (27),
unconfirmedCOVNotification (28),
unconfirmedEventNotification (29),
unconfirmedPrivateTransfer (30),
unconfirmedTextMessage (31),
timeSynchronization (32),
-- utcTimeSynchronization (36),
who-Has (33),
who-Is (34),

-- Services added after 1995
readRange (35), -- Object Access Service
utcTimeSynchronization (36), -- Remote Device Management Service
lifeSafetyOperation (37), -- Alarm and Event Service
subscribeCOVProperty (38), -- Alarm and Event Service
getEventInformation (39) -- Alarm and Event Service
}

~~**BACnetSessionKey** ::= SEQUENCE {~~
~~sessionKey OCTET STRING (SIZE(8)), -- 56 bits for key, 8 bits for checksum~~
~~peerAddress BACnetAddress~~
~~}~~
. . .

[Add to **Clause 25**, pp. 448-449]
[Note: the normal convention for the use of italics in this document does not apply in this section because the lines to be added will contain italics in the printed standard. Therefore, the following lines are to be added exactly as shown below]

FIPS 180-2 (2002) *Federal Information Processing Standards – Secure Hash Standard*

FIPS 197 (2002) *Federal Information Processing Standards – Advanced Encryption Standard*

[Change **Annex C**, p. 502]

```
DEVICE ::= SEQUENCE {
. . .
                number-of-APDU-retries          [73]    Unsigned,
                ~~list-of-session-keys          [55]    SEQUENCE OF BACnetSessionKey OPTIONAL,~~
                time-synchronization-recipients [116]   SEQUENCE OF BACnetRecipient OPTIONAL
. . .
}
```

[Insert into **Annex C**, p. 511]

```
NETWORK-SECURITY ::= SEQUENCE {
          object-identifier                 [75]  BACnetObjectIdentifier,
          object-name                       [77]  CharacterString,
          object-type                       [79]  BACnetObjectType,
          description                       [28]  CharacterString OPTIONAL,
          base-device-security-policy       [327] BACnetSecurityLevel,
          network-access-security-policies  [332] SEQUENCE OF BACnetNetworkSecurityPolicy,
                                                   -- accessed as a BACnetARRAY
          security-time-window              [335] Unsigned,
          packet-reorder-time               [333] Unsigned,
          distribution-key-revision         [328] Unsigned8,
          key-sets                          [330] SEQUENCE SIZE(2) OF BACnetSecurityKeySet,
                                                  -- accessed as a BACnetARRAY
          last-key-server                   [331] BACnetAddressBinding,
          security-pdu-timeout              [334] Unsigned16,
          update-key-set-timeout            [337] Unsigned16,
          supported-security-algorithms     [336] SEQUENCE OF Unsigned8,
          do-not-hide                       [329] BOOLEAN,
          profile-name                      [168] CharacterString OPTIONAL
}
```

[Change Clause **D.11**, p. 518]

D.11 Examples of a Device Object

Example 1: A "sophisticated" BACnet device.

```
. . .
Property:          Number_Of_APDU_Retries =        3
~~Property:          List_Of_Session_Keys =    ((X'3799246237984589', 1, X'03'),~~
~~                    (X'4446214686489744', 1, X'05'))~~
Property:          Time_Synchronization_Recipients =(Device, Instance 18)
 . . .
```

[Add new Clause **D.X**, p. 534]

D.X Example of a Network Security Object

The following is an example of a Network Security object that is found in a secure device that does not support routing.

Property:	*Object_Identifier =*	*(Network Security, Instance 1)*
Property:	*Object_Name =*	*"Network Security Settings"*
Property:	*Object_Type =*	*NETWORK_SECURITY*
Property:	*Description =*	*"Encrypted Device"*
Property:	*Base_Device_Security_Policy =*	*ENCRYPTED*
Property:	*Network_Access_Security_Policies*	*= ((40, PLAIN_NON_TRUSTED) , (50, ENCRYPTED_TRUSTED))*
Property:	*Security_Time_Window =*	*180*
Property:	*Packet_Reorder_Time =*	*1000*
Property:	*Distribution_Key_Revision =*	*32*
Property:	*Key_Sets =*	*((32, (1-OCT-2007, 00:00:00.0), (2-OCT-2007, 01:00:00.0), ((0,1),(0,2),(0,4),(0,5))), ((33, (2-OCT-2007, 00:00:00.0), (3-OCT-2007, 01:00:00.0), ((0,1),(0,2),(0,4),(0,5))))*
Property:	*Last_Key_Server =*	*((Device, Instance 40), 2, X'C0A80001BAC0')*
Property:	*Security_PDU_Timeout =*	*3000*
Property:	*Update_Key_Set_Timeout =*	*9000*
Property:	*Supported_Security_Algorithms =*	*(0, 1)*
Property:	*Do_Not_Hide =*	*TRUE*
Property:	*Profile_Name =*	*"0-Network Security Object Type"*

[Delete **Clause E.6**, "Security Services," pp. 502-503]

[Delete **Clause F.6**, "Encoding for Example E.6 - Security Services," pp. 536-537.]

[Insert new Clause **J.2.13**, p. 568]

J2.13 Secure-BVLL: Purpose

This message is used to secure BVLL messages that do not contain NPDUs. Its use is described in Clause 24.

J.2.13.1 Secure-BVLL: Format

The Secure-BVLL message consists of four fields:

BVLC Type:	1-octet	X'81'	BVLL for BACnet/IP
BVLC Function:	1-octet	X'0C'	Secure-BVLL
BVLC Length:	2-octets	L	Length L, in octets, of the BVLL message
Security Wrapper:	Variable length		

The BVLL to be secured is placed into the Service Data field of the Security Wrapper. For more details on securing BACnet message see Clause 24.

[Change **Annex A**]

…
Network Security Options:

□ Non-secure Device - is capable of operating without BACnet Network Security
□ Secure Device - is capable of using BACnet Network Security (NS-SD BIBB)
□ Multiple Application-Specific Keys
□ Supports encryption (NS-ED BIBB)
□ Key Server (NS-KS BIBB)

[Insert new **Clause K.X**, p 590]

K.X BIBB - Network Security BIBBs

These network security BIBBs prescribe the BACnet capabilities required to interoperably perform the network security functions described in Clause 24.

K.X.1 BIBB - Network Security - Secure Device (NS-SD)

The Secure Device BIBB describes the basic functionality that all secure BACnet devices shall support.

BACnet Service	Initiate	Execute
Challenge-Request		x
Security-Payload	x	x
Security-Response	x	x
Request-Key-Update	x	
Update-Key-Set		x
Update-Distribution-Key		x
What-Is-Network-Number	x	x
Network-Number-Is	x	x

Devices claiming support for this BIBB shall support a Device-Master key, Distribution key, General-Network-Access key, Installation key, User-Authenticated and at least 1 Application-Specific key. A secure device is allowed to limit the number of Application-Specific keys it can contain, but it shall not limit which Application-Specific Key Identifier (6 .. 127) values it accepts.

Secure devices shall support MD5 and SHA-256 for signing secure BACnet messages, and AES for encrypting BACnet messages. Secure devices are allowed to restrict their use of encryption to key exchange only.

K.X.2 BIBB - Network Security - Encrypted Device (NS-ED)

The Encrypted Device BIBB is claimed by devices that are capable of using encryption for all BACnet communications. Devices claiming this BIBB shall support NS-SD and shall be able to encrypt all BACnet messages except the Request-Master-Key and Set-Master-Key services which by definition cannot be encrypted.

K.X.4 BIBB - Network Security - Multi-Application Device (NS-MAD)

The Multi-Application Device BIBB is claimed by devices that are capable of using more than 1 Application-Specific security key. Devices claiming this BIBB shall support NS-ED, shall be able to be configured with at least 5 Application-Specific security keys (the key identifier key numbers to be selected by site policy, not by the implementation), and shall be able to use any of its configured Application-Specific keys to encrypt all BACnet messages except the Request-Master-Key and Set-Master-Key services which by definition cannot be encrypted.

K.X.5 BIBB - Network Security-Device Master Key-A (NS-DMK-A)

The A device is capable of providing Device-Master to secure devices.

BACnet Service	Initiate	Execute
Request-Master-Key		x
Set-Master-Key	x	

K.X.6 BIBB - Network Security-Device Master Key-B (NS-DMK-B)

The B device is capable of accepting Device-Master keys via the Requeste-Master-Key and Set-Master-Key services.

BACnet Service	Initiate	Execute
Request-Master-Key	x	
Set-Master-Key		x

K.X.7 BIBB - Network Security-Key Server (NS-KS)

The Key Server BIBB describes the functionality that all BACnet Key Servers shall support.

BACnet Service	Initiate	Execute
Request-Key-Update		x
Update-Key-Set	x	
Update-Distribution-Key	x	
Request-Master-Key		x
Set-Master-Key	x	

A device claiming the Key Server BIBB shall support the NS-SD BIBB, NS-DMK-A BIBB and all of the functionality described in Clause 24.22 with the exception of the optional temporary key server functionality described in Clause 24.22.4. The device shall support a configurable key distribution period with a range of at least 1 day to 1 year.

A device claiming the Key Server BIBB that provides any other BACnet functionality shall be capable of having the Key Server functionality disabled while allowing the other BACnet functionality to operate normally.

K.X.8 BIBB - Network Security-Temporary Key Server (NS-TKS)

The Temporary Key Server BIBB describes the functionality required to configure keys in installations that do not have a permanent Key Server installed.

BACnet Service	Initiate	Execute
Request-Key-Update		x
Update-Key-Set	x	
Update-Distribution-Key	x	
Request-Master-Key		x
Set-Master-Key	x	

A device claiming the Temporary Key Server BIBB shall support the NS-SD BIBB, NS-DMK-A BIBB and the functionality described in Clause 24.22. Temporary Key Servers need not be able to support periodic updating of Key Sets in secure devices.

A device claiming the Temporary Key Server BIBB that provides any other BACnet functionality shall be capable of having the Key Server functionality disabled while allowing the other BACnet functionality to operate normally.

K.X.9 BIBB - Network Security-Secure Router (NS-SR)

The Secure Router BIBB describes the basic functionality that all secure BACnet routers shall support.

A device claiming the Secure Router BIBB shall be a BACnet router or BACnet half-router and shall support the NS-SD and NS-ED BIBBs. Secure BACnet routers shall support individually configurable security levels for each port and shall support all security levels (plain-non-trusted, plain-trusted, signed-trusted, and encrypted-trusted) for each port. A secure router shall support the largest NPDU for each port that it supports based on the medium connected to the port.

K.X.10 BIBB - Network Security-Security Proxy (NS-SP)

The Secure Proxy BIBB describes the basic functionality that all secure BACnet Secure Proxy devices shall support.

A device claiming the Secure Proxy BIBB shall support the NS-SR BIBB and shall also support at least 1 port that can be configured to be plain-trusted for which it acts as a security proxy.

Security proxy devices shall provide the functionality to protect a complete network of non-secured BACnet devices as described in Clause 24.18 BACnet Security Proxy. The optional ability to protect a subset of the devices is not required by this BIBB.

[Insert new **ANNEX R,** p. 599**]**

ANNEX R - MAPPING NETWORK LAYER ERRORS (NORMATIVE)

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

This annex describes the mapping of network layer and BVLL layer errors to application errors to allow for reporting of errors up the BACnet stack to the application program. This allows recording of errors by the application entity in a singular format.

There is no requirement that all of these errors be passed to the application layer, but when errors are provided to the application layer, these mappings shall be used. There are cases, such as the receipt of Reject-Message-To-Network messages, where there is no simple method for associating the error with the original request.

Table R-1. Mapping Security-Response Response Codes to Error Class and Error Code Pairs

Security-Response Response Code	Error Class / Error Code
success	SECURITY / SUCCESS
accessDenied	SECURITY / ACCESS_DENIED
badDestinationAddress	SECURITY / BAD_DESTINATION_ADDRESS
badDestinationDeviceId	SECURITY / BAD_DESTINATION_DEVICE_ID
badSignature	SECURITY / BAD_SIGNATURE
badSourceAddress	SECURITY / BAD_SOURCE_ADDRESS
badTimestamp	SECURITY / BAD_TIMESTAMP
cannotUseKey	SECURITY / CANNOT_USE_KEY
cannotVerifyMessageId	SECURITY / CANNOT_VERIFY_MESSAGE_ID
correctKeyRevision	SECURITY / CORRECT_KEY_REVISION
destinationDeviceIdRequired	SECURITY / DESTINATION_DEVICE_ID_REQUIRED
duplicateMessage	SECURITY / DUPLICATE_MESSAGE
encryptionNotConfigured	SECURITY / ENCRYPTION_NOT_CONFIGURED
encryptionRequired	SECURITY / ENCRYPTION_REQUIRED
incorrectKey	SECURITY / INCORRECT_KEY
invalidKeyData	SECURITY / INVALID_KEY_DATA
keyUpdateInProgress	SECURITY / KEY_UPDATE_IN_PROGRESS
malformedMessage	SECURITY / MALFORMED_MESSAGE
notKeyServer	SECURITY / NOT_KEY_SERVER
securityNotConfigured	SECURITY / SECURITY_NOT_CONFIGURED
sourceSecurityRequired	SECURITY / SOURCE_SECURITY_REQUIRED
tooManyKeys	SECURITY / TOO_MANY_KEYS
unknownAuthenticationType	SECURITY / UNKNOWN_AUTHENTICATION_TYPE
unknownKey	SECURITY / UNKNOWN_KEY
unknownKeyRevision	SECURITY / UNKNOWN_KEY_REVISION
unknownSourceMessage	SECURITY / UNKNOWN_SOURCE_MESSAGE

Table R-2. Mapping Reject-Message-To-Network Reasons to Error Class and Error Code Pairs

Reject-Message-To-Network Reason	Error Class / Error Code
0	COMMUNICATION / OTHER
1	COMMUNICATION / NOT_ROUTER_TO_DNET
2	COMMUNICATION / ROUTER_BUSY
3	COMMUNICATION / UNKNOWN_NETWORK_MESSAGE
4	COMMUNICATION / MESSAGE_TOO_LONG
5	COMMUNICATION / SECURITY_ERROR
6	COMMUNICATION / ADDRESSING_ERROR

Table R-3. Mapping BVLL Errors to Error Class and Error Code Pairs

Error Condition	Error Class / Error Code

Write-Broadcast-Distribution-Table NAK	COMMUNICATION / WRITE_BDT_FAILED
Read-Broadcast-Distribution-Table NAK	COMMUNICATION / READ_BDT_FAILED
Register-Foreign-Device NAK	COMMUNICATION / REGISTER_FOREIGN_DEVICE_FAILED
Read-Foreign-Device-Table NAK	COMMUNICATION / READ_FDT_FAILED
Delete-Foreign-Device-Table-Entry NAK	COMMUNICATION / DELETE_FDT_ENTRY_FAILED
Distribute-Broadcast-To-Network NAK	COMMUNICATION / DISTRIBUTE_BROADCAST_FAILED

[Insert new **ANNEX S,** p. 599**]**

ANNEX S - EXAMPLES OF SECURE BACnet MESSAGES (INFORMATIVE)

(This annex is not part of this standard but is included for information only.)

This annex provides examples of the use of secure messages defined in Clause 24. All of the examples are written from the point of view of secure device 1 (SecDev1), whose messages are shown in the left-hand column. Messages from all other devices are shown in the right-hand column.

S.1 Example of an Initial Key Distribution

In this example, SecDev1, has just been connected to the BACnet network and is manually manipulated to request a Device-Master key. SecDev1 is not connected to a temporary physically secure network in this example; not all sites would accept this form of initial key distribution as it is inherently insecure. It would be better to connect SecDev1 to a physically secured port on the KeyServer that is dedicated to providing initial key sets to devices.

; Send a request for a Device-Master key.
 	
Request-Master-Key(
Control = NPDU,
Key Revision = 0,
Key Id = 0/0,
Source Device Instance = SecDev1,
Message Id = any valid value,
Timestamp = any valid value (may be incorrect),
Destination Device Instance = 4194303,
DNET = 65535,
DADR = empty,
SNET = 0,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
Number_Of_Encryption_Algs = 1,
Supported_Encryption_Algs = (0),
Number_Of_Signature_Algs = 2,
Supported_Signature_Algs = (0, 1),
Padding = not present,
Signature = all 0s)	

	; The Key Server responds with a Device-Master key. Set-Master-Key(Control = NPDU, Key Revision = 0, Key Id = 1/1, Source Device Instance = KeyServer1, Message Id = MsgId1, Timestamp = current time (TimeStamp1), Destination Device Instance = SecDev1, DNET = 0, DADR = SecDev1MAC,

	SNET = KeyServer1Net,
	SADR = KeyServer1MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	Key = any valid key,
	Padding = not present,
	Signature = as generated

; Acknowledge receipt of the Device-Master key. Security-Response(Control = NPDU, Key Revision = current revision, Key Id = 1/1, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = KeyServer1, DNET = KeyServer1Net, DADR = KeyServer1MAC, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Response Code = 0 (success), Originating Message Id = MsgId1, Original Timestamp = TimeStamp1 Padding = not present, Signature = as generated)	
; Request the Key Server provide a full set of keys. Request-Key-Update(Control = NPDU, Encrypted Key Revision = 0, Key Id = 1/1, Source Device Instance = SecDev1, Message Id = anything, Timestamp = current time, Destination Device Instance = KeyServer1, DNET = KeyServer1Net, DADR = KeyServer1MAC, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Set 1 Key Revision = 0, Set 1 Key Expiration Time = any valid value, Set 2 Key Revision = 0, Set 2 Key Expiration Time = any valid value, Distribution Key Revision = 0, Padding = present, Signature = all 0s)	
	; Key Server provides a Distribution key first.

	Update-Distribution-Key(Control = NPDU, Encrypted Key Revision = 0, Key Id = 1/1, Source Device Instance = KeyServer1, Message Id = MsgId1, Timestamp = TS1, Destination Device Instance = SecDev1, DNET = 0, DADR = SecDev1MAC, SNET = KeyServer1Net, SADR = KeyServer1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Key Revision = any valid value, Key = any valid value, Padding = present, Signature = as generated)
; Acknowledge receipt of the Distribution key. Security-Response(Control = NPDU, Encrypted Key Revision = 0, Key Id = 1/1, Source Device Instance = SecDev1, Message Id = anything, Timestamp = current time, Destination Device Instance = KeyServer1, DNET = KeyServer1Net, DADR = KeyServer1MAC, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Response Code = 0, MsgId1, TS1, Response Specific Parameters = not present, Padding = present, Signature = as generated)	
	; Key Server then provides all other keys. Update-Key-Set(Control = NPDU, Encrypted Key Revision = distribution key revision, Key Id = 1/2, Source Device Instance = KeyServer1, Message Id = MsgId2, Timestamp = TS2, Destination Device Instance = SecDev1, DNET = 0, DADR = SecDev1MAC, SNET = KeyServer1Net, SADR = KeyServer1MAC,

	Authentication Mechanism = not present, Authentication Data = not present, Service Data = <parameters describing key sets appropriate for SecDev1>, Padding = present, Signature = as generated
; Acknowledge receipt of keys. Security-Response(Control = NPDU, Encrypted Key Revision = distribution key revision, Key Id = 1/2, Source Device Instance = SecDev1, Message Id = anything, Timestamp = current time, Destination Device Instance = KeyServer1, DNET = KeyServer1Net, DADR = KeyServer1MAC, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Response Code = 0, MsgId2, TS2, Response Specific Parameters = not present, Padding = present, Signature = as generated)	
; Determine local network number now that the ; device has a General-Network-Access key. Security-Payload(Control = NPDU Key Revision = current key revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = anything, Timestamp = current time, Destination Device Instance = 4194303, DNET = 0, DADR = empty, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Type = What-is-Network-Number, Padding = not present, Signature = as generated)	
	; A device on the local network provides the network ; number. Security-Payload(Control = NPDU

	Key Revision = current key revision, Key Id = 1/4, Source Device Instance = SecDev2, Message Id = anything, Timestamp = current time, Destination Device Instance = 4194303, DNET = SecDev2Net, DADR = empty, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Type = Network-Number-Is, Network Number = SecDev2Net, Configured Flag = any valid value, Padding = not present, Signature = as generated)

S.2 Example of Device Startup

In this example, SecDev1, has just been powered up. It has its security keys but does not have time nor a network number. In this example, the message that allows the device to determine the current time is a broadcast packet. The device might also wait for a unicast directed at it, or it might collect any unicast packet aimed at any device in order to retrieve the time.

	; First broadcast message sent on the network after the ; device startup (this could also be a unicast message ; sent to SecDev1). Security-Payload(Control = APDU Key Revision = current key revision, Key Id = 1/4, Source Device Instance = SecDev3, Message Id = anything, Timestamp = current time, Destination Device Instance = 4194303, DNET = 65535, DADR = empty, SNET = SecDev3Net, SADR = SecDev3MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = …, Padding = not present, Signature = as generated)
; Request the local network number. Security-Payload(Control = NPDU Key Revision = current key revision, Key Id = 1/4,	

Source Device Instance = SecDev1, Message Id = a random value, Timestamp = current time (derived from initial msg), Destination Device Instance = SecDev2, DNET = 0, DADR = empty, SNET = 0, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Type = What-is-Network-Number, Padding = not present, Signature = as generated)	
	; A device on the local network provides the network ; number. Security-Payload(Control = NPDU Key Revision = current key revision, Key Id = 1/4, Source Device Instance = SecDev2, Message Id = MsgId1, Timestamp = Timestamp1, Destination Device Instance = 4194303, DNET = SecDev2Net, DADR = empty, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Type = Network-Number-Is, Network Number = SecDev2Net, Configured Flag = any valid value, Padding = not present, Signature = as generated)
; Challenge the device to validate the learned ; timestamp and network number. Challenge-Request(Control = NPDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = MsgId2, Timestamp = TimeStamp2, Destination Device Instance = SecDev2, DNET = SecDev1Net, DADR = SecDev1MAC, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Challenge = 1,	

Original Message Id = MsgId1,
Original Timestamp = TimeStamp1,
Padding = not present,
Signature = as generated)	

	; The challenge response, allowing SecDev1 to trust the ; time and local network number. Security-Response(Control = NPDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev2, DNET = SecDev1Net, DADR = SecDev1MAC, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Response Code = 0 (success), Originating Message Id = MsgId2, Original Timestamp = TimeStamp2 Padding = not present, Signature = as generated)

S.3 Examples of Secured Confirmed Requests

S.3.1 ReadProperty Example

In this example, SecDev1, is reading a property from SecDev2.

; Send a ReadProperty request. Security-Payload(Control = APDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev2, DNET = SecDev2Net, DADR = SecDev2MAC, SNET = SecDev1Net, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Confirmed-Request-PDU(service = ReadProperty, objectIdentifier = SecDev2,	

propertyIdentifier = object-name),
Padding = not present,
Signature = as generated)	

	; A positive response to the ReadProperty request.
	
	Security-Payload(
	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	Complex-Ack(
	service-ACK-choice = ReadProperty,
	objectIdentifier = SecDev2,
	propertyIdentifier = object-name,
	propertyValue = "Lighting Controller 201"),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

S.3.2 ReadProperty Error Example

In this example, SecDev1, is reading a property from SecDev2 for which it does not have sufficient authorization.

; Send a ReadProperty request. Security-Payload(Control = APDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev2, DNET = SecDev2Net, DADR = SecDev2MAC, SNET = SecDev1Net, SADR = SecDev1MAC, Authentication Mechanism = 0, Authentication Data = 10, Service Data = Confirmed-Request-PDU(service = ReadProperty, objectIdentifier = SecDev2, propertyIdentifier = device-address-binding), Padding = not present,	

Signature = as generated)	

	; A negative response to the ReadProperty request.
	
	Security-Payload(
	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	Error (
	error-choice = ReadProperty,
	error-class = security,
	error-code = read-access-denied),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

S.3.3 Segmented ReadProperty Example

In this example, SecDev1, is reading a property from SecDev2, and the response requires segmentation.

; Send a ReadProperty request.
 	
Security-Payload(
Control = APDU,
Key Revision = current revision,
Key Id = 1/4,
Source Device Instance = SecDev1,
Message Id = any valid value,
Timestamp = current time,
Destination Device Instance = SecDev2,
DNET = SecDev2Net,
DADR = SecDev2MAC,
SNET = SecDev1Net,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
Confirmed-Request (
invokeID = 2,
service = ReadProperty,
objectIdentifier = SecDev2,
propertyIdentifier = object-list),
Padding = not present,
Signature = as generated)	

	; First segment of a segmented response.

	Security-Payload(
	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	ComplexACK(
	segmented-message = True,
	more-follows = True,
	invokeID = 2,
	sequence-number = 0,
	proposed-window-size = 2,
	service-ACK-choice = ReadProperty,
	objectIdentifier = SecDev2,
	propertyIdentifier = object-name,
	propertyValue = …),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

; SegmentAck to set the window size.
 	
Security-Payload(
Control = APDU,
Key Revision = current revision,
Key Id = 1/4,
Source Device Instance = SecDev1,
Message Id = any valid value,
Timestamp = current time,
Destination Device Instance = SecDev2,
DNET = SecDev2Net,
DADR = SecDev2MAC,
SNET = SecDev1Net,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
SegmentACK (
negative-ACK = False,
server = False,
original-invokeID = 2,
sequence-number = 0,
actual-window-size = 2),
Padding = not present,
Signature = as generated)	

	; First segment of the first window. Security-Payload(

	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	ComplexACK(
	segmented-message = True,
	more-follows = True,
	invokeID = 2,
	sequence-number = 1,
	proposed-window-size = 2,
	service-ACK-choice = ReadProperty,
	…),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

	; Second segment of the first window.
	
	Security-Payload(
	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	ComplexACK(
	segmented-message = True,
	more-follows = True,
	invokeID = 2,
	sequence-number = 2,
	proposed-window-size = 2,
	service-ACK-choice = ReadProperty,
	…),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

; Send a SegmentAck for the first window. Security-Payload(	

Control = APDU,
Key Revision = current revision,
Key Id = 1/4,
Source Device Instance = SecDev1,
Message Id = any valid value,
Timestamp = current time,
Destination Device Instance = SecDev2,
DNET = SecDev2Net,
DADR = SecDev2MAC,
SNET = SecDev1Net,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
SegmentACK (
negative-ACK = False,
server = False,
original-invokeID = 2,
sequence-number = 2,
actual-window-size = 2),
Padding = not present,
Signature = as generated)	

	; First segment of the second window and last segment.
	
	Security-Payload(
	Control = APDU,
	Key Revision = 0,
	Key Id = 1/4,
	Source Device Instance = SecDev2,
	Message Id = any valid value,
	Timestamp = current time,
	Destination Device Instance = SecDev1,
	DNET = SecDev1Net,
	DADR = SecDev1MAC,
	SNET = SecDev2Net,
	SADR = SecDev2MAC,
	Authentication Mechanism = not present,
	Authentication Data = not present,
	Service Data =
	ComplexACK(
	segmented-message = True,
	more-follows = False,
	invokeID = 2,
	sequence-number = 3,
	proposed-window-size = 2,
	service-ACK-choice = ReadProperty,
	…),
	Key = any valid key,
	Padding = not present,
	Signature = as generated)

; Send a SegmentAck for the final segment. Security-Payload(Control = APDU, Key Revision = current revision, Key Id = 1/4,	

Source Device Instance = SecDev1,
Message Id = any valid value,
Timestamp = current time,
Destination Device Instance = SecDev2,
DNET = SecDev2Net,
DADR = SecDev2MAC,
SNET = SecDev1Net,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
SegmentACK (
negative-ACK = False,
server = False,
original-invokeID = 2,
sequence-number = 3,
actual-window-size = 2),
Padding = not present,
Signature = as generated)	

S.4 Security Challenge Example

In this example, SecDev1, is reading a property from SecDev2, and SecDev2 challenges SecDev1 to ensure that it is the true source of the message.

; Send a ReadProperty request.
 	
Security-Payload(
Control = APDU,
Key Revision = current revision,
Key Id = 1/4,
Source Device Instance = SecDev1,
Message Id = MsgId1,
Timestamp = TimeStamp1,
Destination Device Instance = SecDev2,
DNET = SecDev2Net,
DADR = SecDev2MAC,
SNET = SecDev1Net,
SADR = SecDev1MAC,
Authentication Mechanism = not present,
Authentication Data = not present,
Service Data =
Confirmed-Request-PDU(
service = ReadProperty,
objectIdentifier = SecDev2,
propertyIdentifier = object-name),
Padding = not present,
Signature = as generated)	

	; Challenge SecDev1 to ensure it originated the message. Challenge-Request(Control = NPDU, Key Revision = current revision,

	Key Id = 1/4, Source Device Instance = SecDev2, Message Id = MsgId2, Timestamp = TimeStamp2, Destination Device Instance = SecDev1, DNET = SecDev1Net, DADR = SecDev1MAC, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Message Challenge = 1, Original Message Id = MsgId1, Original Timestamp = TimeStamp1, Padding = not present, Signature = as generated)
; Answer the Challenge. Security-Response(Control = NPDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev2, DNET = SecDev2Net, DADR = SecDev2MAC, SNET = SecDev1Net, SADR = SecDev1MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data = Response Code = 0 (success), Originating Message Id = MsgId2, Original Timestamp = TimeStamp2 Padding = not present, Signature = as generated)	
	; Send a response to the ReadProperty request. Security-Payload(Control = APDU, Key Revision = 0, Key Id = 1/4, Source Device Instance = SecDev2, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev1, DNET = SecDev1Net, DADR = SecDev1MAC, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present, Service Data =

	Complex-Ack(
	service-ACK-choice = ReadProperty,
	objectIdentifier = SecDev2,
	propertyIdentifier = object-name,
	propertyValue = "Lighting Controller 201"),
	Key = any valid key,
	Padding = not present,
	Signature = as generated

S.5 Secure-BVLL Example

In this example, SecDev1, reads the Broadcast Distribution Table from SecDev2.

; Send Read-Broadcast-Distribution-Table request. Secure-BVLL(Security Wrapper = Security-Payload(Control = NPDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev1, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev2, DNET = SecDev2Net, DADR = SecDev2MAC, SNET = SecDev1Net, SADR = SecDev1MAC, Authentication Mechanism = 0, Authentication Data = 10, Service Data = Read-Broadcast-Distribution-Table () Padding = not present, Signature = as generated))	
	; Send a response. Secure-BVLL(Security Wrapper = Security-Payload(Control = NPDU, Key Revision = current revision, Key Id = 1/4, Source Device Instance = SecDev2, Message Id = any valid value, Timestamp = current time, Destination Device Instance = SecDev1, DNET = SecDev1Net, DADR = SecDev1MAC, SNET = SecDev2Net, SADR = SecDev2MAC, Authentication Mechanism = not present, Authentication Data = not present,

	Service Data = Read-Broadcast-Distribution-Table - Ack(…) Padding = not present, Signature = as generated))

[Add a new entry to **History of Revisions**, p. 688]

(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.)

HISTORY OF REVISIONS

Protocol		***Summary of Changes to the Standard***
Version	***Revision***	
…	…	**…**
1	11	**Addendum *g* to ANSI/ASHRAE 135-2008** Approved by the ASHRAE Standards Committee June 26, 2010; by the ASHRAE Board of Directors June 30, 2010; and by the American National Standards Institute July 1, 2010. 1. Update BACnet Network Security

POLICY STATEMENT DEFINING ASHRAE'S CONCERN FOR THE ENVIRONMENTAL IMPACT OF ITS ACTIVITIES

ASHRAE is concerned with the impact of its members' activities on both the indoor and outdoor environment. ASHRAE's members will strive to minimize any possible deleterious effect on the indoor and outdoor environment of the systems and components in their responsibility while maximizing the beneficial effects these systems provide, consistent with accepted standards and the practical state of the art.

ASHRAE's short-range goal is to ensure that the systems and components within its scope do not impact the indoor and outdoor environment to a greater extent than specified by the standards and guidelines as established by itself and other responsible bodies.

As an ongoing goal, ASHRAE will, through its Standards Committee and extensive technical committee structure, continue to generate up-to-date standards and guidelines where appropriate and adopt, recommend, and promote those new and revised standards developed by other responsible organizations.

Through its *Handbook*, appropriate chapters will contain up-to-date standards and design considerations as the material is systematically revised.

ASHRAE will take the lead with respect to dissemination of environmental information of its primary interest and will seek out and disseminate information from other responsible organizations that is pertinent, as guides to updating standards and guidelines.

The effects of the design and selection of equipment and systems will be considered within the scope of the system's intended use and expected misuse. The disposal of hazardous materials, if any, will also be considered.

ASHRAE's primary concern for environmental impact will be at the site where equipment within ASHRAE's scope operates. However, energy source selection and the possible environmental impact due to the energy source and energy transportation will be considered where possible. Recommendations concerning energy source selection should be made by its members.