Application Note 159/13 v04

Title:	**KNXnet/IP Secure**	
Status:		**Date:**
	Draft Proposal	2013.09.19
Transitional period:	Immediate effect after Final Voting.	
Date:	2013.09.19	
Subject:	KNXnet/IP Secure Protocol Specification	

Documents:

Modified

None.

Referred

[1] **Bellare, Mihir. 2006.** *New Proofs for NMAC and HMAC: Security without Collision-Resistance.* 2006.

[2] **Dierks, T. und Rescorla, E. 2008.** *The Transport Layer Security (TLS) Protocol Version 1.2.* 2008.

[3] **Dr. Kinne, Andreas. 2004.** *Security in EIBnet/IP.* s.l. : Siemens AG, 2004.

[4] **Eronen, P. und Tschofenig, H. 2005.** *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS).* 2005.

[5] **Harkins, D. und Carrel, D. 1998.** *The Internet Key Exchange (IKE).* s.l. : IETF, 1998.

[6] **Kent, S. und Seo, K. 2005.** *Security Architecture for the Internet Protocol.* s.l. : IETF, 2005.

[7] **Lechner, Daniel, Granzer, Wolfgang und Kastner, Wolfgang. 2009.** *Security for KNXnet/IP.* s.l. : TU Wien, 2009.

[8] **Orman, H. 1998.** *The OAKLEY Key Determination Protocol.* 1998.

[9] **Rescorla, E. 2009.** *Diffie-Hellman Key Agreement Method.* s.l. : IETF, 2009.

[10] **Rescorla, E. 2000.** *HTTP Over TLS.* s.l. : Rescorla, E., 2000.

[11] **Rescorla, E. und Modadugu, N. 2006.** *Datagram Transport Layer Security.* 2006.

[12] **Rijmen, Vincent und Daemen, Joan. 2002.** *The Design of Rijndael: AES. The Advanced Encryption Standard.* 2002.

[13] **KNX.** Communication Security.

[14] **NIST.** Recommended Elliptic Curves for Government Use.

[15] **NIST.** Federal Information Processing Standards Publication 180-2, Secure Hash Signature Standard. 2002

Document updates

Version	Date	Modifications
TFIP008-01	2011.02.21	Initial proposal to TF IP
	2011.07.25	Revised version, incorporating feedback from TF IP
TFIP008-06	2011.10.11	Revised version, further feedback from TF IP
TFIP008-07	2011.11.11	Revised version, according to feedback from TF IP
TFIP008-08	2012-06-26	Revised version, further feedback from TF IP
TFIP008-09	2012-08-20	Revised version, further feedback from TF IP
TFIP008-10	2012-08-22	Changed unicast key exchange and authentication
TFIP008-11	2012-08-24	Final update with feedback from TF IP
AN159 v01	2013.02.04	Preparation of the Draft Proposal.
AN159 v02	2013.05.13	Resolution of RfV comments.
AN159 v03	2013.07.05	Addition of unencrypted message examples.
AN159 v04	2013.09.19	Editorial review in view of publication.

Contents

1 Purpose, motivation and scope 4
1.1 Introduction 4
1.2 KNXnet/IP security requirements 4
1.3 Authentication requirements 5
1.3.1 General 5
1.3.2 KNXnet/IP device authentication 6
1.3.3 Client authentication 6
2 Specification 6
2.1 General information 6
2.1.1 Securing KNXnet/IP services 6
2.1.2 Algorithms and key sizes 7
2.2 Unicast connections 7
2.2.1 Providing confidentiality 7
2.2.2 Providing data integrity 7
2.2.3 Key management 7
2.2.4 Authentication 7
2.2.5 Defending against replay attacks 9
2.2.6 Associating security keys and secure connections 10
2.3 Multicast communication 10
2.3.1 Providing confidentiality 10
2.3.2 Providing Data Integrity 10
2.3.3 Security domains 11
2.3.4 Key Management 12
2.3.5 Authentication 12
2.3.6 Defending against Replay Attacks 12
2.4 KNXnet/IP secure details 13
2.4.1 Integrating with KNXnet/IP 13
2.4.2 KNXnet/IP secure header 15
2.4.3 SECURE_WRAPPER frame 16

2.4.4 SECURE_CHANNEL_REQUEST frame 17
2.4.5 SECURE_CHANNEL_RESPONSE frame 17
2.4.6 SECURE_CHANNEL_AUTHORIZE frame 18
2.4.7 SECURE_CHANNEL_STATUS frame 19
2.4.8 SECURE_GROUP_SYNC_REQUEST frame 20
2.4.9 SECURE_GROUP_SYNC_RESPONSE frame 21
2.5 Communication patterns 22
2.5.1 Secure connection setup 22
2.5.2 Disconnecting a secure connection 23
2.5.3 Retransmission of lost frames 24
2.5.4 Synchronizing timers 24
2.6 Announcing KNXnet/IP secure capability 26
2.7 KNXnet/IP secure Parameter Object 26
2.7.1 PID_GROUP_KEY (PID=51) 26
2.7.2 PID_DEVICE_AUTHENTICATION_CODE (PID=52) 26
2.7.3 PID_PASSWORD_HASHES (PID=53) 27
2.7.4 PID_SECURED_SERVICES (PID=54) 27
2.7.5 PID_MULTICAST_LATENCY_TOLERANCE (PID=55) 28
2.7.6 PID_DOMAIN_SEND_MASK (PID=56) 29
2.7.7 PID_DOMAIN_ACCESS_MASK (PID=57) 29
2.7.8 Manufacturing requirements for secure devices 29
2.7.9 Implementation considerations 30
A.1 Example of SECURE_CHANNEL_REQUEST frame 31
A.1.1 Unencrypted Binary Example 31
A.1.2 Encrypted Binary Example 31
A.2 Example of SECURE_CHANNEL_RESPONSE frame 32
A.2.1 Unencrypted Binary Example (Success) 32
A.2.2 Unencrypted Binary Example (Error) 32
A.2.3 Encrypted Binary Example (Success) 32
A.3 Examples of SECURE_WRAPPER frames 33
A.3.1 Unencrypted SECURE_CHANNEL_AUTHORIZE frame 33
A.3.2 Encrypted SECURE_CHANNEL_AUTHORIZE frame 34
A.3.3 Unencrypted SECURE_CHANNEL_STATUS frame 34
A.3.4 Encrypted SECURE_CHANNEL_STATUS frame 34
A.3.5 Unencrypted ROUTING_INDICATION frame 35
A.3.6 Encrypted ROUTING_INDICATION frame 35
A.4 Example of SECURE_GROUP_SYNC_REQUEST frame 36
A.4.1 Unencrypted Binary Example 36
A.4.2 Encrypted Binary Example 36
A.5 Example of SECURE_GROUP_SYNC_RESPONSE frame 37
A.5.1 Unencrypted Binary Example 37
A.5.2 Encrypted Binary Example 37

1 Purpose, motivation and scope

1.1 Introduction

KNXnet/IP is a protocol designed to transport KNX building automation frames over an IP network. It is used as an infrastructure backbone for connecting KNX sub-networks, as a communication medium for KNX-IP devices and to provide IP based services for clients (e.g. connecting a tool software to a KNX installation). The main advantages of using IP for these purposes are that IP network infrastructure is inexpensive, available almost everywhere and that the distance of two communication parties on an IP network is virtually unlimited.

KNXnet/IP differentiates between unicast and multicast services. KNXnet/IP unicast services are used to connect a single client to a single KNXnet/IP device (e.g. for KNXnet/IP Tunnelling). KNXnet/IP multicast services are mainly used to connect KNX sub-networks using IP networks. The KNXnet/IP Routing services are defined for this purpose. KNXnet/IP multicast builds on top of IP multicast.

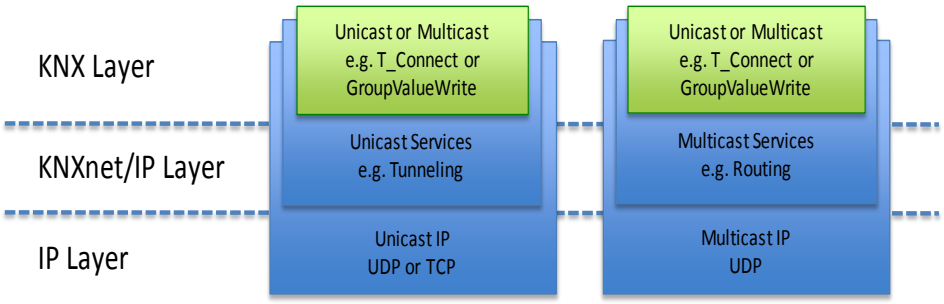


Figure 1 - Unicast and Multicast in the sense of KNX, KNXnet/IP and IP

The KNXnet/IP services like Routing or Tunnelling (as well as the transported KNX protocol itself) do not provide any dedicated security features today. Even with KNX secure communication used in the KNX Telegram, this does only provide security on application layer and still reveals potentially sensitive information for eavesdroppers. This increasingly becomes a problem because with IP as the underlying protocol networks today get connected worldwide and highly sophisticated methods of attacking IP networks exist. Thus, the use of KNX and KNXnet/IP in open IP networks today should be limited to applications where security isn't critical.

As part of the KNX secure communication strategy, this document specifies the optional security wrapper for securing KNXnet/IP traffic that puts an additional layer of security, that is transparent to all existing KNXnet/IP services, around the complete KNXnet/IP traffic.

1.2 KNXnet/IP security requirements

The most important goal of securing KNXnet/IP traffic is keeping outside attackers from gaining control over a KNX building automation system while connecting remotely over the Internet. A slightly different and also important scenario is an attacker gaining access over the local (W)LAN at the automation system's site. From the view of a KNXnet/IP device these two attacks are equivalent. Defending against this kind of attack leads to two security objectives:

- Data Integrity
  Ensuring data integrity means keeping an attacker from gaining control by injecting manipulated KNXnet/IP frames

- Freshness
  Ensuring freshness means keeping an attacker from recording packets and playing them back at a later time without manipulating the contents. This is called a replay-attack: An attacker records KNXnet/IP traffic and at the same time observes what is happening in the automated building to relate recorded packets to actions. At a later time he can maliciously trigger an action by replaying the appropriate previously recorded packet

A different goal is keeping attackers from deriving knowledge about what is going on inside a building by looking at the KNXnet/IP traffic. This leads to the security objective of ensuring

- Confidentiality
  Confidentiality is normally achieved by encrypting network traffic to grant an attacker the lowest possible insight into the data actually transferred.

The three security objectives mentioned above can collectively be referred to as providing a Secured Communication Channel. Aside from communicating over a secured channel it is important that the communicating peers can trust each other's claimed identity. The security objective for this is called

- Mutual Authentication.

Mutual authentication (if done correctly) also prevents an attack known as man-in-the-middle attack. When conducting a man-in-the-middle attack, an attacker is logically located in between two communicating parties. He is able to intercept and change the traffic flowing in both directions. He masquerades as legal communication partner to both peers and sets up secured communication channels to both sides. He decrypts traffic coming from one side and re-encrypts it before sending it to the other side. So he is able to intercept and change the clear text communication. See clause 1.3 for a discussion of KNXnet/IP authentication requirements.

A special requirement of securing KNXnet/IP is the support for

- Secure Multicast Communication

with more than one sender. A special difficulty arises in this case when trying to ensure freshness. See 2.3 for details on how to guarantee freshness for KNXnet/IP multicast services.

Because KNXnet/IP is implemented in small embedded devices, a security solution should require

- Reasonable Processing Power and Memory Consumption.

1.3 Authentication requirements

1.3.1 General

For KNXnet/IP we have to differentiate between three different kinds of authentication:

- Authenticating the server (the KNXnet/IP device) to the client (tool software) for a KNXnet/IP unicast connection (e.g. Tunnelling or Device Management).
- Authenticating the client (tool software) to the server (the KNXnet/IP device) for a KNXnet/IP unicast connection (e.g. Tunnelling or Device Management).
- Mutual authentication of the nodes involved in KNXnet/IP multicast (Routing) communication.

1.3.2 KNXnet/IP device authentication

The goal for unicast server authentication is to ensure that a client talking to a KNXnet/IP device (e.g. in order to configure the device) is really talking to the device it thinks it is talking to. Without this authentication an attacker could mimic or fake a specific KNXnet/IP device in order to gain knowledge about sensible configuration data sent to the device.

To achieve this, each KNXnet/IP device identifies itself with an individual authentication code that is configured in the device during commissioning. The factory-default setup key (KNX Communication Security) is used by the device as authentication code in factory-default state. Knowledge of the device authentication code during connection setup (requiring physical access to the device at least once during commissioning) establishes trust between the server device and the client software. For details on KNXnet/IP device authentication see 2.2.4.1.

1.3.3 Client authentication

The goal for unicast client authentication is to ensure that a client talking to a KNXnet/IP device (e.g. in order to configure the device or ta access the KNX bus) is really authorized to do so. Access to the KNXnet/IP device's services is secured with 2 passwords stored in the devices: One mandatory password (management level) for secure device configuration using the KNXnet/IP Device Management connection and one optional password (user level) for all other KNXnet/IP connections (e.g. Tunnelling). See clause 2.2.4.1 for details on the password based client authentication.

2 Specification

This chapter contains the details of securing the KNXnet/IP traffic regarding confidentiality, data integrity, freshness and mutual authentication of the communicating parties. It is based on the same cryptographic algorithms and procedures as KNX secure communication on application layer, but adopted to match the specific requirements of the KNXnet/IP domain.

2.1 General information

2.1.1 Securing KNXnet/IP services

At the time of this writing, 7 KNXnet/IP services are defined:

- Discovery
- Device Management
- Tunnelling
- Routing
- Remote Logging
- Remote Configuration and Diagnosis
- Object Server

The discovery service is intentionally not secured to allow any client to search for devices on the network and discover their security requirements. For the remaining services security can be turned on or off per service.

2.1.2 Algorithms and key sizes

As a trade-off between desired security and available processing power as measured in terms of throughput and latency, the key sizes and algorithms of the KNXnet/IP secure standard represent the minimum requirement for supporting secure communication. While this standard may be extended with additional key sizes and algorithms in the future, the current set will remain the default. This means that any KNXnet/IP device shall at least support AES-128 CCM as a fall back if negotiation of other algorithms between communication parties fails.

2.2 Unicast connections

2.2.1 Providing confidentiality

Confidentiality shall be provided using symmetric encryption with the AES algorithm (Rijmen & Daemen, 2002) and a key length of 128 bit. For encapsulating data of the higher protocol layers inside a secure container, AES shall be used in counter mode (CTR) with T_0 being composed of the sequence identifier in the upper 6 octets, 9 fixed zero octets and the counter (starting with zero) in the lowest octet.

2.2.2 Providing data integrity

A message authentication code (MAC) shall be appended to every message for ensuring data integrity. Message authentication is based on the assumption that only legal communication parties possess the session communication key. A unicast sender shall authenticate its sent messages by proving knowledge of the respective key. For calculating the MAC, the CBC-MAC method with AES as block cipher and the session key shall be used. B_0 shall be composed of the sequence identifier in the upper 6 octets, 8 fixed zero octets and the length Q in the lowest 2 octets.

2.2.3 Key management

A fresh session key shall be used for every session. This session key shall be used for calculating the CBC-MAC and for AES encryption of the payload (CCM).

The session key shall be negotiated between the two communicating parties during connection setup using the Elliptic-Curve Diffie-Hellman (ECDH) key agreement algorithm (Rescorla, Diffie-Hellman Key Agreement Method, 2009). As public ECDH domain parameters the NIST K-283 curve shall be used. During connection setup both parties exchange additional randomly generated ECDH parameters. Knowledge of each other's public parameter in combination with the own private parameter allows both peers to calculate the same random number. The lowest 16 octets of this number shall be used as session key for the symmetric session runtime encryption.

2.2.4 Authentication

During unicast communication setup, a KNXnet/IP secure device shall authenticate itself to the connecting client by proving knowledge of its device authentication code. The client shall authenticate itself to the device by proving knowledge of a secret password, which has previously been stored in the device.

2.2.4.1 Authenticating the device to the client

The KNX device shall authenticate itself against the client with its device authentication code. This authentication is integrated into the Elliptic-Curve Diffie-Hellman sequence when the KNXnet/IP server sends its ECDH public value to the client (compare STS station-to-station protocol) using the following steps:

- The received client public value X is XORed with the server public value Y.
- The resulting data is authenticated with CCM authentication using the device authentication code as key and a B0 of Q.
- All together (data and MAC) is CCM encrypted with the session key and T0 of zero.
- The authentication data generated in the above steps is sent together with the server public value Y in the ECDH response from the server to the client.

The client shall reverse the encryption and authentication steps and this way check the device authentication code. Only if the device authentication succeeded, the client shall continue authenticating itself to the device.

2.2.4.2 Authenticating the client to the device

For authenticating the client (typically the tool software or visualizations) to the server (KNXnet/IP secure device), each device shall be secured by two passwords for two different levels of access.

- Management-Level Access
  On the management level all services including Device Management shall be supported. To change KNXnet/IP secure parameters, a management level connection shall be required.
- User-Level Access
  On the user level, all services except Device Management shall be supported. This means, on the user level everything shall be allowed that would also be possible using a physical connection to the KNX bus.

The passwords are specific to devices and therefore may be individual for each device. For practical reasons, a user may choose to opt for the same password set in all devices of his installation project. The client decides which of the two passwords to use for authentication. After successful authentication, the degree of access granted by the device to the client shall depend on the password used during authentication.

After the Elliptic-Curve Diffie-Hellman key agreement the client shall authenticate itself to the server as part of the secure handshake sequence using the following steps:

- The received server public value Y is XORed with the client public value X.
- The resulting data is authenticated with CCM authentication using the password hash and a B0 of Q.
- All together (data and MAC) is CCM encrypted with the session key and T0 of zero.
- The authentication data generated in the above steps is sent in the authorization request from the client to the server.

The server shall reverse the encryption and shall check for correct password used. If this does not match, the client has failed its authentication to the server. The server shall answer to the authentication request with an authentication response indicating to the client if the authentication passed successfully.

If authentication fails, the client has to restart with new ECDH handshake; it is not allowed to send more than one authentication request with the same ECDH session key. This dramatically delays every attempt to on-line attack one of the installation passwords.

2.2.4.3 Securing against dictionary attacks on the installation passwords

Password based authentication mechanisms always bear the risk of a dictionary attack being conducted. In a dictionary attack an attacker tries passwords from a dictionary in order of decreasing probability. For real-world passwords regularly a surprisingly small number of attempts is necessary to find out the correct password.

Dictionary attacks can be separated into on-line attacks and off-line attacks. In an on-line attack, the attacker repeatedly connects to the system and on each connection tries a different password. In an off-line attack, the attacker needs to be able to gain enough information by passively listening to the traffic of a connection to be able to check the passwords off-line on his own (maybe parallelized) hardware.

Off-line attacks are for example possible when the unkeyed hash value for the password is transmitted as plain text. Password based challenge-response authentication schemes may also be prone to off-line dictionary attacks if the attacker is able to get the plain text challenge and the plain text response.

KNXnet/IP secure cannot be compromised by an off-line dictionary attack because the authentication data that an attacker would need to verify the validity of each tested password is protected with the previously ECDH-generated session key which is unknown to any listening third person/attacker.

Conducting an on-line dictionary attack is always possible. However, compared to an off-line attack the number of passwords that can be tried on-line in a given time-span is by orders of magnitude smaller. Since an on-line attack takes considerable time and requires the attacker being actively connected to the installation for that time, his risk of being discovered is much higher compared to an off-line attack.

To further decrease the feasibility of an on-line dictionary attack the Software used to set the installation password should check when the password is changed that the new password has a minimal strength.

2.2.5 Defending against replay attacks

To prevent replay attacks, every packet sent via a secure connection shall bear a counter as sequence identifier. Every direction shall have its own independent counter. The counter shall be initialized to zero on connection setup and incremented by one for each packet sent.

For every connection both peers shall store the sequence identifier of the last received frame. An incoming frame shall be discarded if the sequence identifier is less than or equal to the stored number.

An incoming frame shall be accepted by the receiver if the sequence identifier is greater than the sequence identifier of the previously received frame on the same connection. There is no requirement that the sequence identifiers of two consecutive frames differ by exactly one. Because of this, the sequence identifier mechanism is inherently tolerant to frames getting lost.

The counter shall have a width of 48bit. Assuming one million (10^6) packets being sent per second for each direction, an overflow of the counter would theoretically occur after 9 years. Furthermore the counters start at zero for every new connection further minimizing the risk of an overflow.

2.2.5.1 Securing the initial configuration

A freshly installed device shall have an empty management password set. Therefore during initial configuration of the device (until the password has been changed), no reliable authentication of the client to the device is possible. However, all other security objectives mentioned above are met. The device will authenticate itself to the management client by using the initial secret code printed on the device. As knowledge of this code by the client requires at least some form of physical access to the device, some sort of client authentication is still guaranteed. Confidentiality, freshness and data integrity are assured using the standard measures described above.

2.2.6 Associating security keys and secure connections

During connection setup, the server and the client shall each assign a connection index to the connection being established. When later transferring data over the new connection, the peer's connection index shall be included in every packet sent. This connection index is not to be confused with the KNXnet/IP connection identifier. The connection identifier shall be used for looking up the key and the current sequence counter for every incoming encrypted packet on the receiving side. The existing KNX/IP communication channel id does not serve this purpose because it is buried inside the encrypted data block.

2.3 Multicast communication

2.3.1 Providing confidentiality

Confidentiality shall be provided using symmetric encryption.

For symmetric encryption the AES algorithm (Rijmen & Daemen, 2002) with a key length of 128 bit shall be used. For encapsulating data of the higher protocol layers inside a secure container, AES shall be used in counter mode (CTR) with T_0 being composed of the group counter in the upper 6 octets, 9 fixed zero octets and the counter (starting with zero) in the lowest octet.

Using the CTR mode, no padding to the cipher block size is needed because the cipher text has exactly the same length as the plain text. Due to using an initialization vector containing the group counter, every sequence of cipher text is encrypted to a different sequence of plain text on every encryption. This makes it harder for an attacker to gain information from a traffic analysis.

A single encryption key is used for every IP multicast group. An IP multicast group contains the devices using the same IP multicast address and port number. The multicast group key shall be stored in every device belonging to the IP multicast group. It is transferred once during device commissioning using a secure unicast Device Management connection. The key shall be transferred via the property PID_SECURE_IP_GROUP_KEY. The key shall have an unlimited lifetime.

2.3.2 Providing Data Integrity

A message authentication code (MAC) shall be appended to every message for ensuring data integrity. For calculating the MAC, the CBC-MAC method with AES as cipher algorithm and the multicast group key shall be used. Message authentication is based on the assumption that only legal communication parties possess the multicast group key to correctly generate a CBC-MAC based on this key. Thus, a multicast sender shall authenticate its sent messages by proving knowledge of the respective key. B_0 shall be composed of the group counter in the upper 6 octets, 8 fixed zero octets and the length Q in the lowest 2 octets.

2.3.3 Security domains

KNXnet/IP secure only secures the KNX data transmitted over the IP backbone. In the case of using a KNXnet/IP secure router to connect TP or PL devices to an IP backbone, the traffic originating from TP or PL devices is trusted by the directly connected KNXnet/IP routers. The formerly insecure traffic will be routed via KNXnet/IP secure to other KNXnet/IP routers and eventually be forwarded into their KNX segments.

This would allow an attacker to inject KNX traffic physically into a low-security KNX segment (e.g. used for lighting) to have it forwarded to a high-security KNX segment (e.g. used for door locking).

To prevent this kind of attack, the concept of security domains is introduced. KNX lines that are used for similar functions from a security point of view can be grouped together into a security domain. E.g. lighting devices should be grouped into one security domain. Door locking devices should be grouped into a second security domain.

Up to 16 security domains can be specified. A KNXnet/IP secure device shall have two new configuration parameters for security domain management. Both are 16 bit wide. The first one shall specify the domain mask of all frames sent via KNXnet/IP secure by the device. The second one shall specify the mask for acceptable domains on reception.

If a device sends a KNXnet/IP secure frame, the sending mask shall be included into the frame being sent. If it receives a KNXnet/IP secure frame, it shall perform a bitwise AND between the received domain mask and the configured mask for acceptable domains. If the result is zero, the received frame shall be dropped. Otherwise it shall be processed. In the case of a router, processing means forwarding the frame to the KNX segment.

When processing incoming frames, security domains shall be checked at the KNXnet/IP secure network layer. For the existing layers security domains are fully transparent.

A factory-new device shall have the sending mask set to 0001h. This means it is sending in security domain 0. The accept mask shall be set to FFFFh. So frames sent on any security domain will be accepted by default.

2.3.3.1 Example

An existing installation used solely for lighting is extended by a door opener and a code input pad. For security reasons both the door opener and the code input pad are KNX IP devices.

The code input pad is intended to open the door via KNXnet/IP secure if the correct code was entered. At the same time the light is intended to be switched on. The security can be maintained by configuring the domain masks of the two new devices. The domain send mask of both the code input pad and the door opener will be set to 0002h. This means both devices will be sending in security domain 1. The domain accept mask of both devices will also be set to 0002h. This means that both devices will only accept frames sent in security domain 1. So the code pad is able to write a group value which controls the door opener.

The lighting devices in the installation have the default values set for both masks. They all send in security domain 0 and accept frames sent in any security domain. So a lighting device will receive group writes sent by the code pad.

But on the other hand, even by gaining physical access to the lighting installation it is impossible to control the door opener because frames from security domain 0 are dropped by the door opener.

2.3.4 Key Management

For a device, at any given time, a single key shall be used for secure IP multicast communication. The key shall be set as parameter in the device configuration using secure unicast device management. If one device gets compromised, the key can be changed for the remaining devices using unicast device management to restore security for the group communication. In some scenarios it might be desirable to separate the KNX network into different IP multicast groups, each group using a different key.

2.3.5 Authentication

The devices in an IP multicast group mutually authenticate themselves by the knowledge of the IP multicast group key. As the IP multicast group key may only be changed over a secure unicast device management connection, the combined password / public key authentication scheme used for unicast connections is in effect.

Everyone who is having knowledge of the secret IP multicast group key is considered a legal member of the IP multicast group.

2.3.6 Defending against Replay Attacks

2.3.6.1 Multicast sequence identifiers

For IP multicast communication it is not possible to use a simple counter like in the unicast case. In the multicast case there is an IP multicast group consisting of multiple nodes each of which can act as sender or as receiver. A sequence counter would have to be maintained and persisted by every possible receiver for every sender. Without persistence, a receiver would be vulnerable by a replay attack from power-up until every sender has received a valid frame.

Apart from the resources required on the receiving side, at least one kind of vulnerability would remain. If an attacker can capture a valid frame and make sure that the frame doesn't reach the legal receiver, he can replay the frame later at any time. To get around this issue, a sequence identifier is needed which changes even if no communication is taking place.

Therefore a free running timer that is synchronized between all KNXnet/IP multicast group members shall be used for providing sequence identifiers. The range of the timer values is large enough for the timers to be assumed to never overflow. Synchronization of timer values between devices are only allowed in the forward direction: The sender of a packet shall include its current timer value as time stamp in the sent packet. The receiver of a packet shall replace its own timer value with the received time stamp if the received time stamp is having a greater value than the internal timer.

The sequence identifier of an incoming frame shall be compared to the current local counter value. If the received frame is older than the time span given in PID_SECURE_MULTI-CAST_LATENCY_TOLERANCE (typically a few seconds), the frame shall be discarded. This means that frames with slightly past timer values shall be accepted to account for network latency.

The use of KNX communication security on application layer for critical applications includes an additional device specific counter that further minimizes the risk of replay attacks and reduces the need to tweak the latency tolerance on KNXnet/IP.

2.3.6.2 Device timer implementation

The device timer shall not be decreased in any case. This requirement especially includes the power-down case. See 2.7.9.2 for implementation details.

2.3.6.3 System clock accuracy considerations

The proposed timer-based replay attack protection depends on the device clocks being synchronized with an allowable tolerance of a few milliseconds. Given a standard quartz micro controller oscillator circuit, a clock accuracy better than 50ppm can easily be achieved. This implies that the clock deviation between any two devices is at most 100ppm. If our goal is to achieve synchronization to around 500 milliseconds, the clocks would need to be resynchronized every 5000 seconds. To get a better error tolerance margin, KNXnet/IP secure devices shall start a resynchronization cycle about once an hour.

2.3.6.4 Synchronizing timers

Two special packet types are specified for synchronizing the group timers.

A SECURE_GROUP_SYNC_REQUEST shall be sent by a device requiring timer synchronization (see below). The other devices shall answer by sending SECURE_GROUP_SYNC_RESPONSE frames in a way that

- every device shall send its response frame at a random time between 0 and 5s after reception of the SECURE_GROUP_SYNC_REQUEST,
- every device shall send at most one frame,
- a device shall not send a response if it has received a SECURE_GROUP_SYNC_RESPONSE frame with at least the own local timer value since the SECURE_GROUP_SYNC_REQUEST was received,

A group synchronization request shall sent by a device

- after power up and
- if no synchronization response has been received since start of the cyclic synchronization timer.

When the cyclic synchronization timer of a device is started, it shall be set to expire after a random time between 60 and 70 minutes. It shall be started after power up and shall be restarted with a new expiration time after reception of each SECURE_GROUP_SYNC_-RESPONSE frame.

The timer and the time stamp shall have a width of 48bit. With timer ticks every millisecond, an overflow of the timer would theoretically occur after 9 thousand years. This value should be sufficient to well exceed the lifetime of an average building. The timer shall automatically be reset to zero if the multicast key is changed.

2.4 KNXnet/IP secure details

2.4.1 Integrating with KNXnet/IP

Figure 2 shows the protocol architecture of the KNXnet/IP protocol. KNXnet/IP is implemented on top of the UDP/TCP network layer. Figure 3 shows how the KNXnet/IP security layer is integrated into the existing protocol architecture. KNXnet/IP is replaced by a security layer which wraps and encrypts the original KNXnet/IP layer.

Locating the security layer between the TCP/IP layer and the KNXnet/IP layer has the advantage that on one side the TCP/IP standard mechanisms like routing and multicast remain intact and also apply to the secured connection. On the other side by just wrapping KNXnet/IP frames, no change to existing frames is necessary. Furthermore, KNXnet/IP secure can be expected to work seamlessly with existing and future extensions to KNXnet/IP.

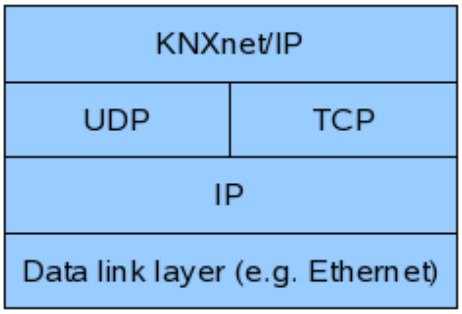


Figure 2 - KNXnet/IP protocol stack

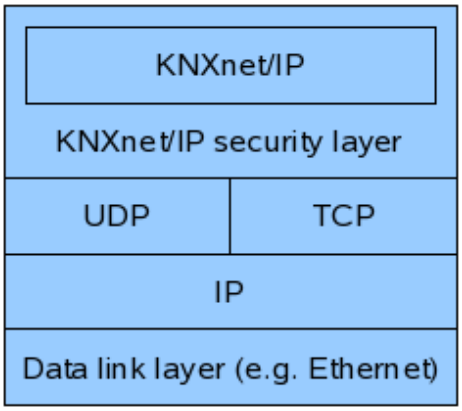


Figure 3 - KNXnet/IP secure protocol stack

For compatibility the KNXnet/IP security layer frames shall start with a standard KNXnet/IP header. Using the standard header it is possible to run secure communication in parallel to insecure communication in the same installation and even on the same endpoint. A security-enabled device can distinguish security layer frames from regular KNXnet/IP frames by looking at the Service Type Identifier located in the common KNXnet/IP header. Non-security-enabled devices will just ignore the secure frames because they claim to belong to an unsupported KNXnet/IP protocol version.

When KNXnet/IP frames are to be sent over a secured connection, each frame including the KNXnet/IP header shall be completely encapsulated as encrypted payload inside a SECURE_WRAPPER frame.

Inserting an additional security layer without changing the original KNXnet/IP frame format also eases the extension of existing KNX protocol stacks for secure communication. No significant changes to the internal logic should be required. KNXnet/IP can be implemented by intercepting incoming and outgoing frames and reaching decrypted incoming frames though to the regular processing as needed.

In KNXnet/IP Secure, the secure frames get assigned a service type and a protocol version just like any other KNXnet/IP frame. The encapsulated frame is carrying its own protocol version and service type. This has the advantage that the security extension is able to carry any existing and future KNXnet/IP service type without forcing a device supporting future protocol versions to also implement the security extension.

Figure 4 shows how a KNXnet/IP frame shall be embedded into a KNXnet/IP secure frame. It shall apply to secure multicast communication and to secure unicast runtime communication after the secure handshake has been conducted. The standard KNXnet/IP frame shall be prefixed with a secure header and followed by a message authentication code.

The network headers including the TCP/UDP headers shall be unencrypted to take advantage of standard mechanisms like routing, flow control, network address translation, etc. The network headers shall also not be included into the message authentication code to avoid problems with secured frames passing a network address translation.

<table>
<tr><td rowspan="2">Ethernet</td><td rowspan="2">IP</td><td rowspan="2">TCP/
UDP</td><td>KNXnet/IP
Secure
Header</td><td>KNXnet/IP
Standard
Header</td><td>KNXnet/IP
cEMI message</td><td>KNXnet/IP
Secure
MAC</td></tr>
<tr><td rowspan="2">Unencrypted</td><td>Authenticated</td><td colspan="2">Authenticated & Encrypted</td><td>Encrypted</td></tr>
<tr><td colspan="4">Replay protected</td></tr>
</table>

Figure 4 - KNXnet/IP secure frame structure detailing encrypted and authenticated parts

Figure 5 shows a more detailed view on the encrypted part of a multicast routing frame. Because a routing device receives the complete routing traffic within its multicast group, it is important to be able to filter out unwanted frames as fast as possible. The decision whether a received multicast routing frame has to be processed any further depends on the KNX destination address buried inside the cEMI part of the routing frame.

As the figure shows, the KNX destination address can be found at octet positions 13 and 14 of the encrypted part of the KNXnet/IP secure frame. This means that for a routing decision it is sufficient to decrypt only the first 16 octet block of a received frame. So decrypting the whole frame is only necessary for frames that are already known to be of interest for the receiving device. Most other frames can be discarded after just decrypting the first 16 octets of encrypted data.

<table>
<tr><td>KNXnet/IP
Routing Header
(6 Octet)</td><td>KNXnet/IP
cEMI message
(6 octets before KNX destination address)</td><td>KNXnet/IP
Secure
MAC</td></tr>
<tr><td colspan="2">Authenticated & Encrypted</td><td>Encrypted</td></tr>
</table>

Figure 5 - Encrypted part of a KNXnet/IP secure routing frame

2.4.2 KNXnet/IP secure header

All KNXnet/IP secure frames shall share a common header which is based on the KNXnet/IP header.

```
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|         Header Length         |        Protocol Version       |
|         (06h)                 |        (13h)                  |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|                     Service Type Identifier                   |
|                     (2 Octet)                                 |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|                           Total Length                        |
|                           (2 Octet)                           |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
```

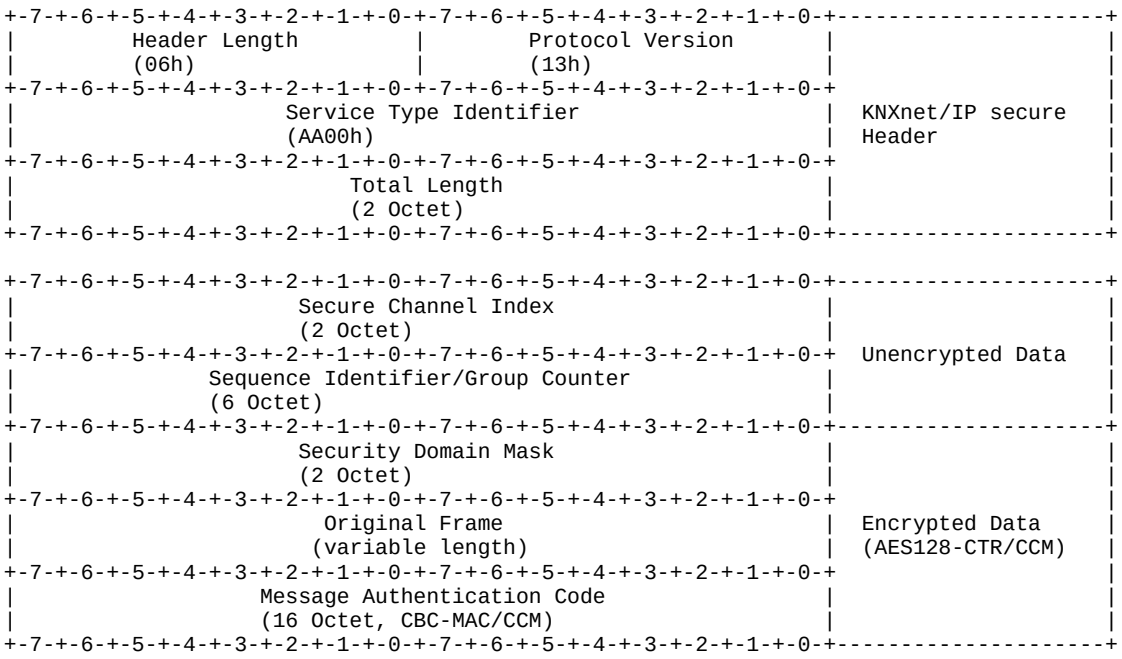
Header fields:

- Header Length
  Length of the header in Octets
- Protocol Version
  Always 13h for KNX/IP secure, meaning protocol version 1.3

- Service Type Identifier
  Identifies the type of the frame body following the header. The high octet shall always be AAh for KNXnet/IP secure frames
- Total Length
  The complete length of the KNXnet/IP secure frame including the frame header

2.4.3 SECURE_WRAPPER frame

The SECURE_WRAPPER frame shall contain an encrypted KNXnet/IP frame plus some extra information needed to decrypt the frame and for ensuring data integrity and freshness. It shall start with the common header specified above.



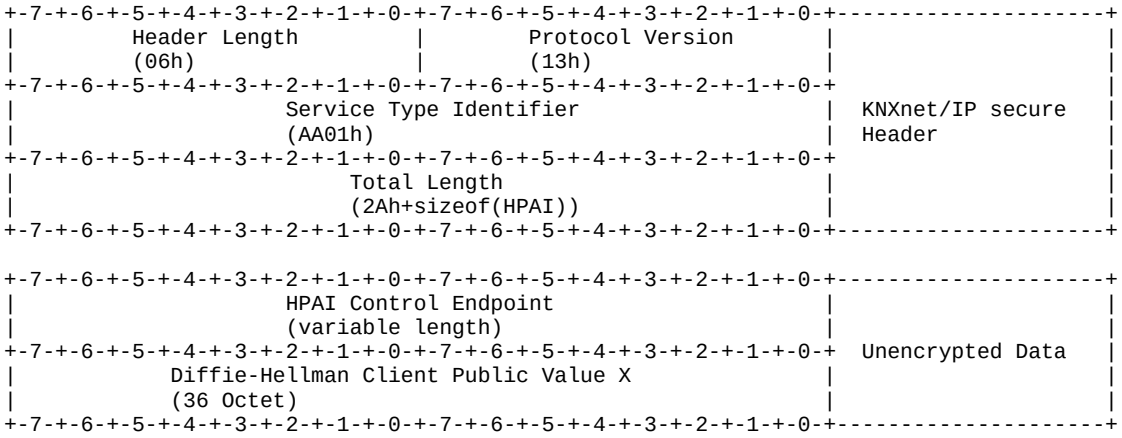
Fields:

- Secure Channel Index
  Index of the secure channel to decide which security key to use for decryption. For multicast connections the fixed channel index 0000h shall be used. For unicast connections the index was established during a previous successful secure channel handshake procedure.
- Sequence Identifier/Group Counter
  This field is used to defend against replay attacks.
  For unicast connections: Monotonically increasing counter value assigned by the sender. Incremented by the sender for each frame sent.
  For multicast connections: Time stamp value with millisecond resolution. Device timers shall be synchronized on reception of valid SECURE_WRAPPER and SECURE_GROUP_SYNC_RESPONSE frames
- Security domain mask
  To restrict access to high-security KNX TP segments by gaining physical access to low-security KNX TP segments, the concept of security domains is used. The bits set in the domain mask shall indicate which security domain this frame is directed to (see 2.3.3). For unicast connections this field shall be ignored and should be set to 0000h.

- Message Authentication Code
  CBC-MAC calculated using the session key (unicast connections) or group key (multicast connections) over all frame fields before encryption up to but not including the Message Authentication Code field itself.

2.4.4 SECURE_CHANNEL_REQUEST frame

The SECURE_CHANNEL_REQUEST frame shall be sent by the KNXnet/IP secure client to the control endpoint of the KNXnet/IP secure server to initiate the secure connection setup handshake for a new secure communication channel.

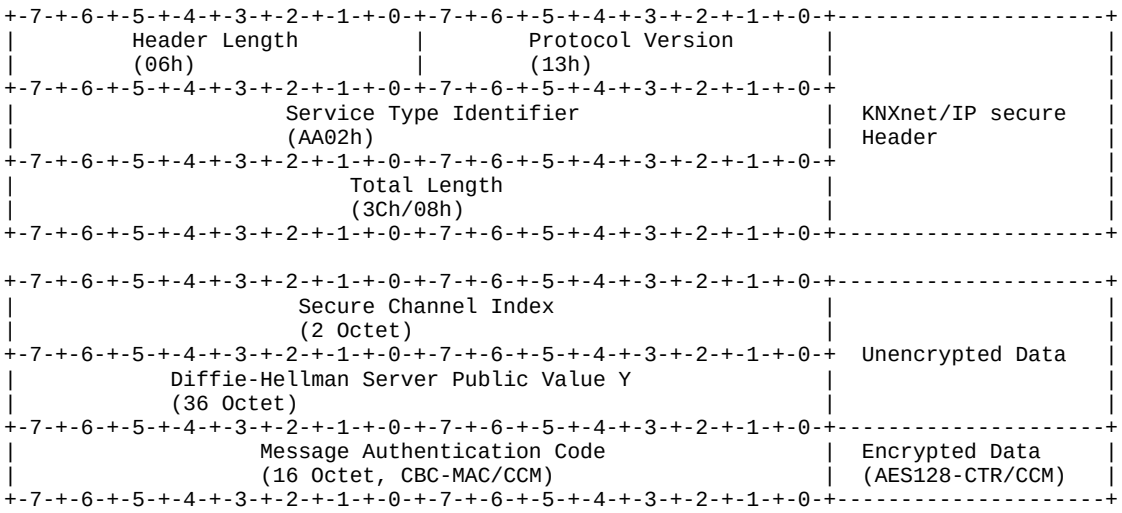


Fields:

- HPAI Control Endpoint
  This field shall contain the return address information of the KNXnet/IP secure client's control endpoint so the KNXnet/IP secure server knows where to direct possible secure channel responses and/or status information.
- Diffie-Hellman Client Public Value X
  ECDH public value X as calculated by the client from its chosen secret on the K-283 standard curve for ECDH key agreement.

2.4.5 SECURE_CHANNEL_RESPONSE frame

The SECURE_CHANNEL_RESPONSE frame shall be sent by the KNXnet/IP secure server to the KNXnet/IP secure client's control endpoint in response to a received secure channel request frame.



Fields:

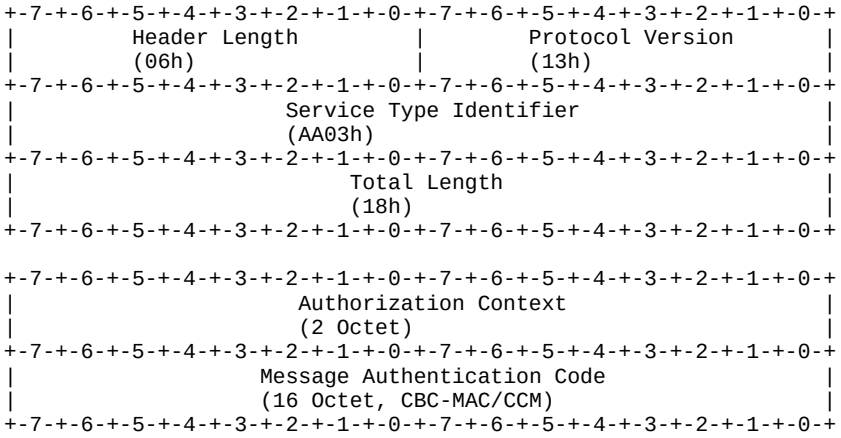
- Secure Channel Index
  Index assigned by the server to the to-be-created connection. The client shall later send this index with every frame related to the connection for the server to select the appropriate decryption key. At choice of the server this number may be the same as the KNXnet/IP connection identifier assigned to this connection later. The secure channel index 00h is reserved for multicast data and must not be used for unicast connections. This value therefore is used to indicate error conditions (see below).
- Diffie-Hellman Server Public Value Y
  ECDH public value Y as calculated by the server from its chosen secret on the K-283 standard curve for ECDH key agreement.
- Message Authentication Code
  To defend against man-in-the-middle-attacks, the server's public value Y shall be authenticated by the device. Therefore the public values X and Y are XORed together and a CBC-MAC with the device authentication code as key is generated for this data. To protect the device authentication code against offline dictionary attacks, the MAC is then AES128-CTR encrypted with the session key derived from the ECDH handshake. The session key shall be calculated on both sides of the connection as the first 16 octets of the shared Elliptic-Curve Diffie-Hellman secret.

If the KNXnet/IP server device cannot process the secure channel request (either because there are no more free secure channels or the device is too busy to calculate another ECDH parameter set) then the secure channel response contains zero as channel index and no Diffie-Hellman parameter nor MAC.

2.4.6 SECURE_CHANNEL_AUTHORIZE frame

The SECURE_CHANNEL_ AUTHORIZE frame shall be sent by the KNXnet/IP secure client to the control endpoint of the KNXnet/IP secure server after the Diffie-Hellman handshake to authorize the user against the server device.

To protect the communication from a denial-of-service attack, the authorization request shall be encrypted and authenticated. Therefore this frame shall be sent wrapped in a SECURE_WARPPER frame.



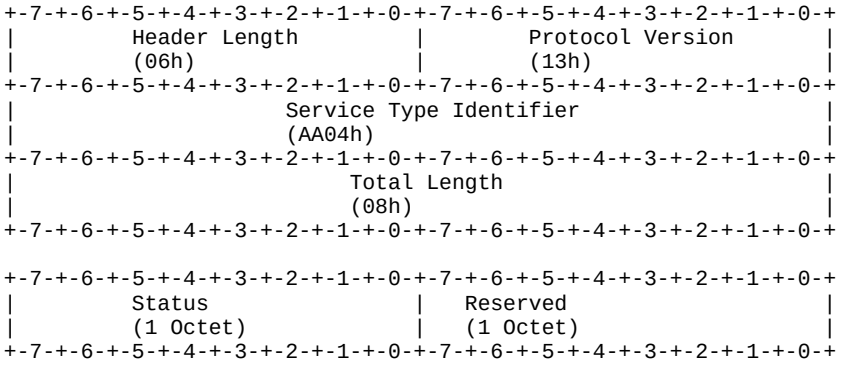
Fields:

- Authorization Context
  01h: Management access
  02h – 7Fh: User access
  80h – FFh: Reserverd, shall not be used
  This field shall indicate the user/role requested by the client for authorization. The server shall use the authorization context as index into the passwords array (see 2.7.3 PID_PASSWORD_HASHES (PID=53) for details). The access level (management access or user access with any device dependent role) shall also determine the set of services accepted by the server after successful authentication.
- Message Authentication Code
  For authorization of the requested access level, the client authenticates itself to the server by signing the ECDH public values X and Y. For this purpose both values are XORed together and a CBC-MAC with the password hash as key is generated for this data.

2.4.7 SECURE_CHANNEL_STATUS frame

The SECURE_HANDSHAKE_STATUS frame shall be sent by the server to the client. It may be sent in any stage of the secure channel handshake to indicate an error condition or status information. In particular the server uses this frame type to response to the authorization request of the client.

To protect the communication from a denial-of-service attack, the channel status frame shall be encrypted and authenticated. Therefore this frame shall be sent wrapped in a SECURE_WARPPER frame.



Fields:

- Status
  Status code
  00h → STATUS_AUTHORIZATION_SUCCESS
  01h → STATUS_AUTHORIZATION_FAILED
  02h → STATUS_ERROR_UNAUTHORIZED
  03h → STATUS_TIMEOUT

2.4.8 SECURE_GROUP_SYNC_REQUEST frame

This frame shall be sent during secure group communication to request synchronization of the group member's timer values. The frame shall be sent to the configured IP group multicast address and port.

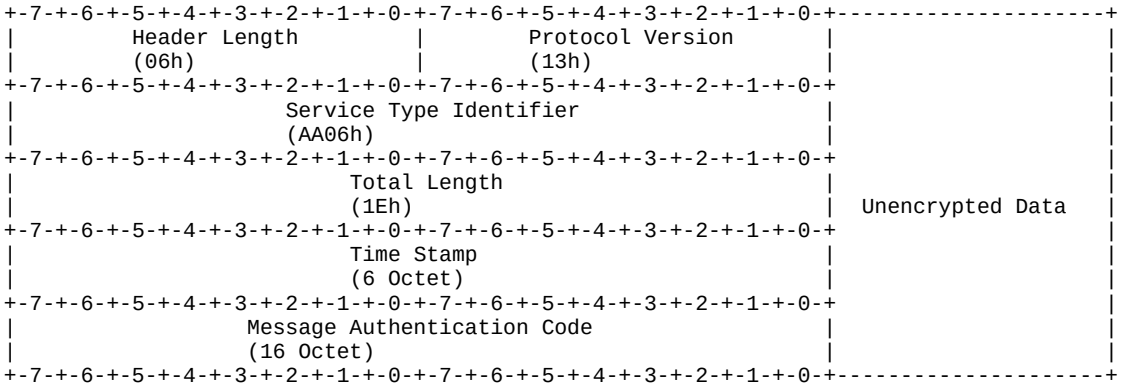
It shall be sent by a group member in two cases:

- To synchronize the device timer after power up, a SECURE_GROUP_SYNC_REQUEST shall be sent at a random time between 0 and 10s after power up if no SECURE_GROUP_SYNC_RESPONSE has been received within this period of time. If the device needs to send data within this first period of time it can immediately issue a SECURE_GROUP_SYNC_REQUEST frame.
- To re-synchronize the timers of all group members a SECURE_GROUP_SYNC_REQUEST shall be sent by any device if no SECURE_GROUP_SYNC_RESPONSE has been received for one hour plus a small random variation between 0 and 10 minutes. The variation is added to keep the group members from sending a SECURE_GROUP_SYNC_REQUEST at the same time.

A device receiving a SECURE_GROUP_SYNC_REQUEST frame shall compare the transmitted Time Stamp value to its local timer value and update its timer value if the received value is greater.

Independently of the received and local timer value any device receiving a SECURE_GROUP_SYNC_REQUEST shall schedule sending a SECURE_GROUP_SYNC_RESPONSE frame after a random delay of up to 5s.

If a SECURE_GROUP_SYNC_RESPONSE frame with at least the local timer value as Time Stamp field is received before the delay elapses, the schedule shall be cancelled.



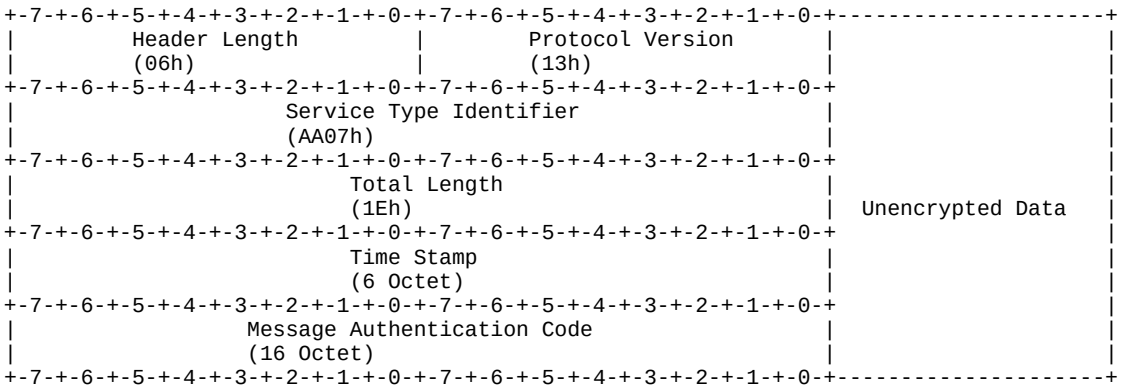
Fields:

- Time Stamp
  Current timer value of the sender. A receiving group member shall take this value as its own timer value if it is greater than its current timer value
- Message Authentication Code
  CBC-MAC calculated using the group key over all frame fields up to but not including the Message Authentication Code field itself

2.4.9 SECURE_GROUP_SYNC_RESPONSE frame

This frame shall be sent as response to a received SECURE_GROUP_SYNC_REQUEST by any device. It shall be sent with a random delay between 0 and 10s to avoid collisions. It shall only be sent if no SECURE_GROUP_SYNC_RESPONSE with at least the own timer value in the Time Stamp field is received during the delay time starting at reception of the SECURE_GROUP_SYNC_REQUEST frame.

The frame is sent to the configured IP group multicast address.



Fields:

- Time Stamp
  Current timer value of the sender. A receiving group member shall take this value as its own timer value if it is greater than its current timer value
- Message Authentication Code
  CBC-MAC calculated using the group key over all frame fields up to but not including the Message Authentication Code field itself

2.5 Communication patterns

2.5.1 Secure connection setup

See Figure 6 for a sequence diagram of the handshake taking place during secure connection setup (the handling of authentication failures is omitted for better readability).

During secure connection setup the Elliptic-Curve Diffie-Hellman (ECDH) key agreement algorithm is used to agree on a common secret key. Since the original ECDH algorithm is vulnerable to a man-in-the-middle attack, a modified form of the algorithm is used which integrates mutual authentication of client and server. For authentication the ECDH public values sent between client and server are authenticated and encrypted in a way that protects the symmetrical keys used for authentication against offline dictionary attacks.

Establishing a secure channel between a client and a server device is always initiated by the client sending a SECURE_CHANNEL_REQUEST frame to the server. This frame contains the client's ECDH public value X.

For every received SECURE_CHANNEL_REQUEST frame the server reserves a secure index, generates the ECDH parameters and calculates the session key for this secure channel. The server then answers with a SECURE_CHANNEL_RESPONSE frame containing its ECDH public value Y and an authentication part consisting of the CBC-MAC signature of "X XOR Y" calculated with the device authentication code as key and AES128-CTR encrypted with the session key for this secure channel.

As the calculation of the ECDH parameters on the server device may take considerable amount of time, the timeout for the server's SECURE_CHANNEL_RESPONSE frame shall be 30 seconds.

If the server runs out of free secure channels or is too busy to fulfil any additional secure channel requests, it shall send an empty SECURE_CHANNEL_RESPONSE frame with a connection index field of 00h.

With the server sends out a successful secure channel response, the returned channel index is marked as "connected" and must not be reused until it is disconnected again.

If no SECURE_WRAPPER frame is received by the server for a connected channel index within two minutes after the last valid frame for this channel, the server assumes a communication error, sends a final SECURE_CHANNEL_STATUS frame with a status field of STATUS_TIMEOUT and closes the connection.

Typically the KNXnet/IP secure client starts authorization of the user directly after the secure channel was established. Therefore the KNXnet/IP secure client sends a SECURE_CHANNEL_AUTORIZE request wrapped inside a SECURE_WRAPPER frame to the KNXnet/IP secure server. The client expects a SECURE_CHANNEL_STATUS from the server within a timeout period of 10 seconds to get the authorization results.

In case of the server detecting an authentication failure, a SECURE_CHANNEL_STATUS frame with a status field of STATUS_AUTHORIZATION_FAILED shall be sent to the client and the connection setup handshake shall be immediately aborted (the secure channel disconnected). Otherwise, if the authorization was successful, the status of the SECURE_CHANNEL_STATUS shall be STATUS_AUTHORIZATION_SUCCESS.

After the secure connection has been established and successfully authorized, a regular KNXnet/IP connection will typically be created on top of it by exchanging CONNECT_REQUEST and CONNECT_RESPONSE frames encapsulated as payload inside SECURE_WRAPPER frames.

The following traffic belonging to the connection will also be securely encapsulated inside SECURE_WRAPPER frames.

If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a service code that requires an authorized channel, but the secure channel has not successfully passed authentication, then the server shall send a SECURE_CHANNEL_STATUS frame with a status field of STATUS_UNAUTHORIZED and disconnect the secure channel.

If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a disconnected channel index, this frame shall be discarded.

If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a connected channel index, but the payload could not be successfully decrypted and/or authenticated, this frame shall be discarded.

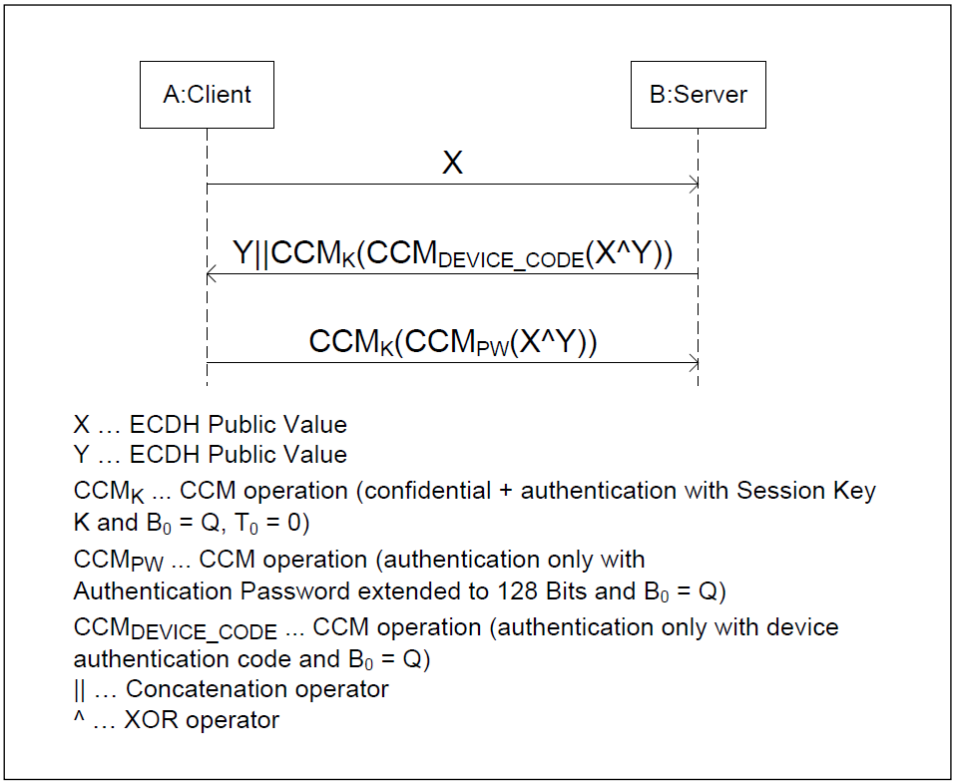


Figure 6 - Secure connection setup handshake

2.5.2 Disconnecting a secure connection

Disconnecting a secure connection works exactly the same way as disconnecting a regular connection. The only difference is the encapsulation of the DISCONNECT_REQUEST and DISCONNECT_RESPONSE frames as payload inside SECURE_WRAPPER frames.

If the secure layer implementation is to be kept separate from the regular KNXnet/IP implementation, it may examine the payload of a received SECURE_WRAPPER frame and the payload of a SECURE_WRAPPER frame to be sent and check for DISCONNECT-frames. It may then free the resources allocated to the secure connection on disconnection of the regular connection running on top.

For connections running over TCP, the secure connection must be disconnected when the underlying TCP connection closes.

2.5.3 Retransmission of lost frames

In the case of a frame getting lost during transmission, the frame will likely be retransmitted. For connections running on top of TCP, retransmissions are handled transparently by the network layer. From the perspective of KNXnet/IP retransmissions will never occur on such connections.

For connections running over UDP, retransmissions are handled by the KNXnet/IP application layer. To avoid interference with the replay-protection, retransmitted frames must be handled by the security layer exactly like new frames. This means, a retransmitted frame passed by the application layer to the security layer is by no means handled differently from a regular frame. It gets a new sequence identifier and is encrypted and authenticated on the sending side exactly like a brand-new frame. On the receiving side it is decrypted like any other frame and passed up to the application layer. The application layer is then responsible to detect the retransmission and act appropriately. The actions taken by the application layer will be exactly the same as in the unsecured case.

2.5.4 Synchronizing timers

Figure 7 shows how a device synchronizes its local timer to the devices in an existing installation after power up. It can be seen how the random delays work together to minimize the number of frames actually sent dependent on the transmitted timer values.

If no SECURE_GROUP_SYNC_RESPONSE was received by device 1, it would continue sending SECURE_GROUP_SYNC_REQUEST frames about every 10 seconds.

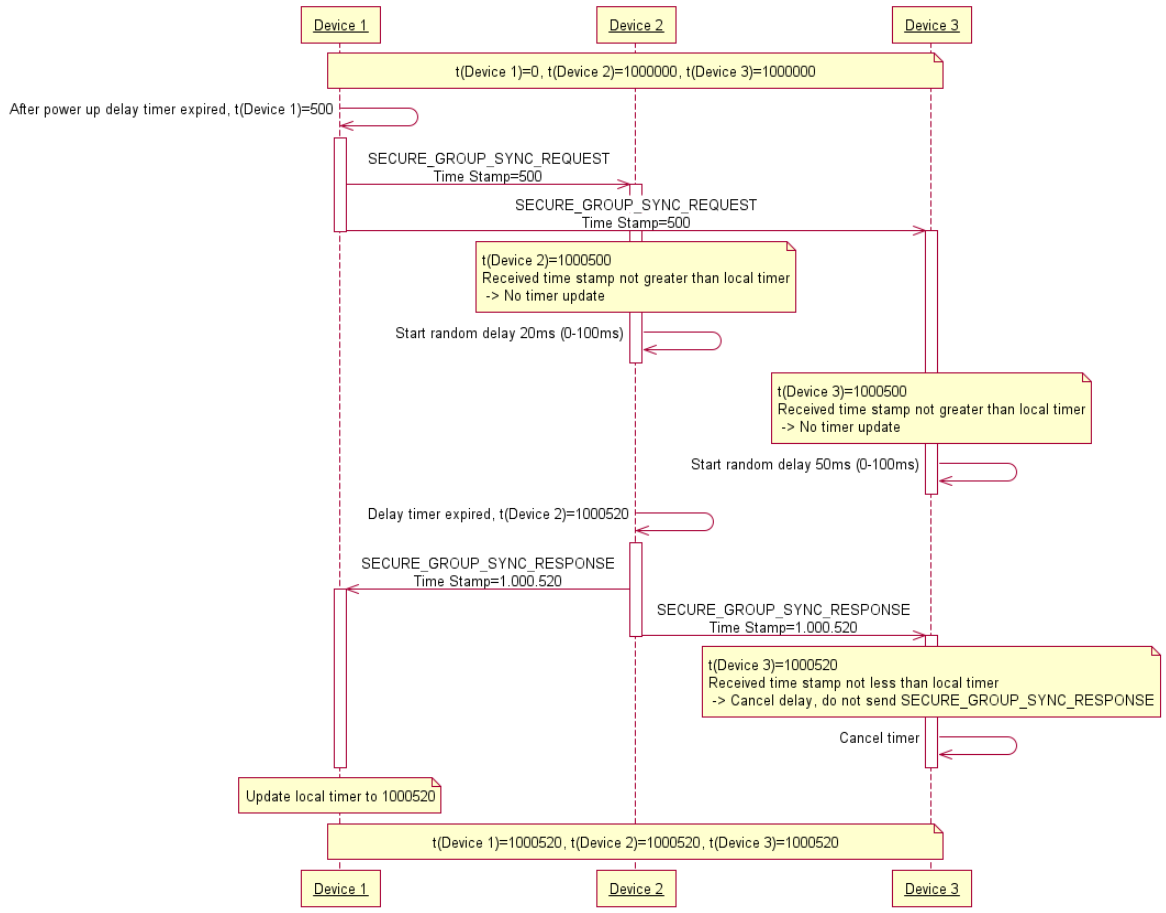


Figure 7 - Timer synchronization after power up

Figure 5 shows how devices are cyclically resynchronized to keep the lag between the individual device timers small. It also helps to achieve synchronization between former physically independent groups being connected together.

Device 1 initiates the synchronization process. Each device has a cyclic synchronization timer expiring at a fixed time with a small random variation. In this example the timer of Device 1 was the first to expire. With the same probability every other device could have been the device initiating the synchronization process.

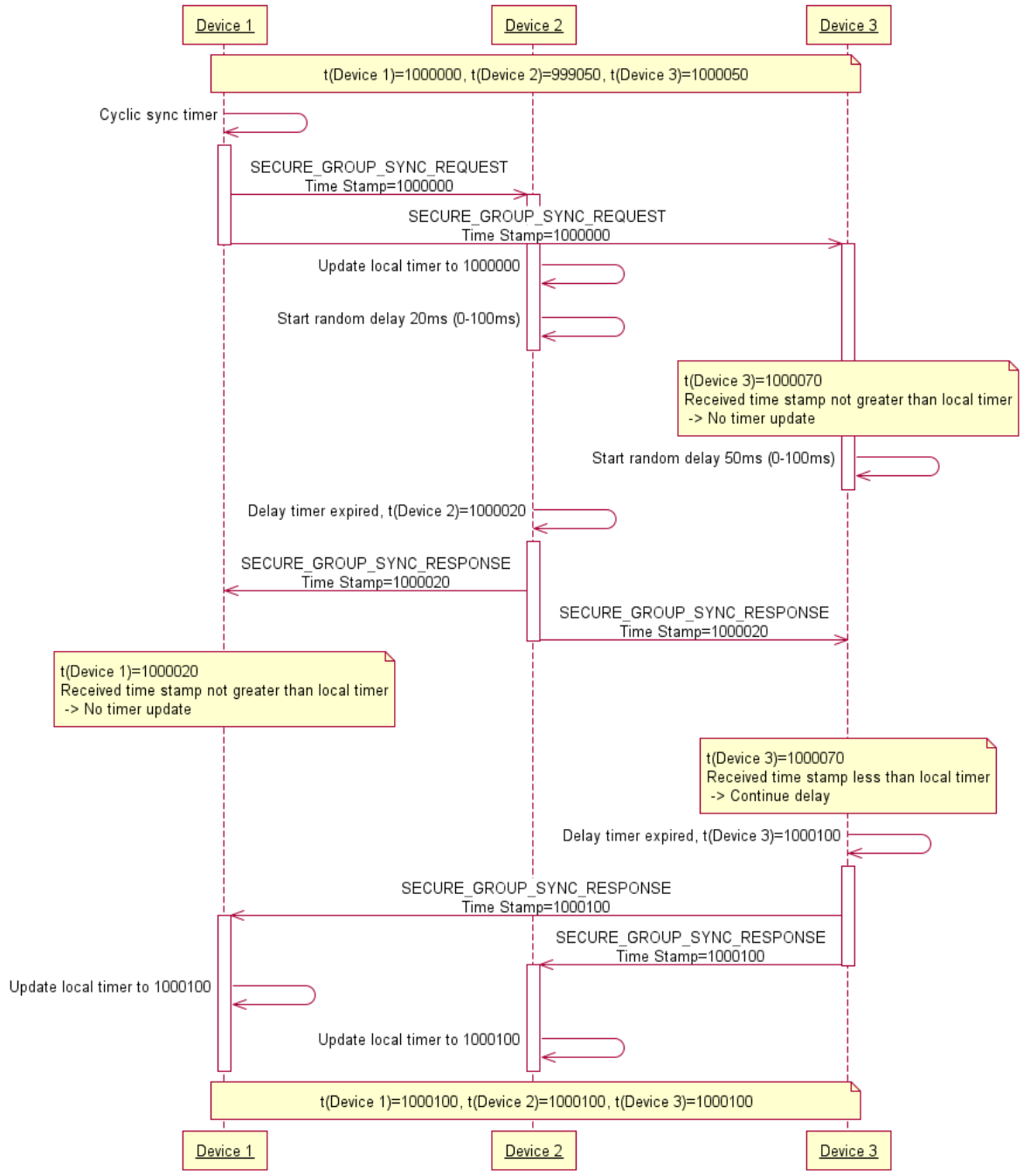


Figure 8 - Cyclic synchronization of timers

The figure shows how the devices interact sending and receiving group synchronization frames and finally arrive with all device timer synchronized.

2.6 Announcing KNXnet/IP secure capability

A KNXnet/IP secure device shall announce its capability of secure communication using the standard device description mechanism. For secure communication a Service Family ID of AAh is assigned. In the device description a SUPP_SVC_FAMILY DIB for the KNXnet/IP secure Service Family ID will be included.

2.7 KNXnet/IP secure Parameter Object

The device management of a security enabled KNXnet/IP device shall expose a new parameter object called “KNXnet/IP secure Parameter Object” (Object Type = 20). This parameter object shall include all security specific parameters listed in this section.

The security related parameters of this interface object (described below) shall only be accessible via a secure connection (KNXnet/IP secure device management or KNX secured APCIs). They shall not be accessible via any other standard- or non-standard means.

2.7.1 PID_GROUP_KEY (PID=51)

- Property name: KNXnet/IP secure Group Key
- Property Datatype: PDT_GENERIC_16
- Datapoint Type: None
- Access: Data: x/x
  Secure: x/3

This parameter shall contain the key used for encryption, decryption and MAC calculation in secure group communication.

The group key shall not be readable and shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

In ex-factory state and after every factory reset the group key shall be zero.

2.7.2 PID_DEVICE_AUTHENTICATION_CODE (PID=52)

- Property name: KNXnet/IP Device Authentication Code
- Property Datatype: PDT_GENERIC_16
- Datapoint Type: None
- Access: Data: x/x
  Secure: x/3

This parameter shall contain the device authentication code used for authentication in secure unicast communication setup.

It shall be is set by the tool software when the device is configured for the first time. It is recommended that each device’s authentication code should be individual (randomly chosen by the tool software), but the user can manually set this code to a user-specific value to simplify connection handling.

The device authentication code shall not be readable and shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

In ex-factory state and after every factory reset the device authentication code shall be the factory default setup key printed on the device (this is also used for secured application layer services in ex-factory state).

2.7.3 PID_PASSWORD_HASHES (PID=53)

- Property name: KNXnet/IP secure Password Hashes
- Property Datatype: PDT_GENERIC_16[]
- Datapoint Type: None
- Access: Data: x/x
  Secure: x/3

This array shall contain the lower 16 octets of the SHA-256 (NIST FIPS PUB 180-2, Secure Hash Signature Standard) hash of the passwords used for authenticating the client during secure unicast connection setup.

The index of the password in the array identifies a user (= Authorization Context, see 2.4.6 SECURE_CHANNEL_AUTHORIZE frame). Therefore the minimum size is 1 (the first entry is always reserved for the management client granting full access) and the maximum size is 127 (management client + maximum 126 additional limited users).

The password hashes shall not be readable and shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

In ex-factory state and after every factory reset shall have the passwords set to the empty string (resulting in the fixed hash 27ae41e4649b934ca495991b7852b855). The tool software shall remind the user if the password has not yet been changed for a device. For changing the passwords, the KNX restriction of at most one active device management connection for a given time applies. This ensures that an attacker is unable to interfere with the password change by opening a parallel connection.

A password change shall only apply to new connections. User connections authenticated using the old password shall remain authenticated after the password change.

The password shall only be used for authentication. For each new session a fresh session key shall be derived from the password. Thus the password is only needed at connection setup time, not during later communication. Once the connection is open, the session key derived from the password shall be used for encryption and message authentication.

2.7.4 PID_SECURED_SERVICES (PID=54)

- Property name: KNXnet/IP secure Secured Services
- Property Datatype: PDT_BITSET_16
- Datapoint Type: None
- Access: Data: 3/x
  Secure: 3/3

This bitset shall indicate the services for which security is enforced. A bit value of '1' shall mean “insecure communication disallowed/security required”, '0' shall mean “insecure communication allowed/security optional”. It applies to both IP unicast and IP multicast connections.

Bit	Description
0	Device Management
1	Tunnelling
2	Routing
3	Remote Logging
4	Remote Configuration and Diagnosis
5	Object Server
6	Reserved
7	Reserved
8	Reserved
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved

If the security flag for a specific service is not set, secure connections as well as insecure connections shall be accepted for that service type. If the security flag for a specific service is set, only secure connections shall be accepted.

By always allowing secure connections it is possible to configure security-related parameters in an existing installation without compromising security.

When a client tries to insecurely connect to a server service which has security enabled, the server shall response with an error code of E_CONNECTION_TYPE.

The secured services bitset shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

2.7.5 PID_MULTICAST_LATENCY_TOLERANCE (PID=55)

- Property name: KNXnet/IP secure Multicast Latency Tolerance
- Property Datatype: PDT_UNSIGNED_INT
- Datapoint Type: 7.002 DPT_TimePeriodMsec
- Access: Data: 3/x
  Secure: 3/3

This parameter specifies the length of the acceptance window for accepting incoming multicast frame with a past timestamp (sequence identifier).

This acceptance window is configurable to suit the needs of the specific installation (should be as small as possible, to be secure, but it can be bigger if there is more latency on the network). The maximum age of a telegram to be accepted as a valid telegram affects the reliability of the system. A greater time span will improve tolerance against network latencies while a shorter time span will improve the protection against replay attacks. A reasonable value for this time span is largely dependent on the expected network latencies.

The multicast latency tolerance shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

2.7.6 PID_DOMAIN_SEND_MASK (PID=56)

- Property name: KNXnet/IP secure Domain Send Mask
- Property Datatype: PDT_BITSET_16
- Datapoint Type: None
- Default value: 0001h
- Access: Data: 3/x
  Secure: 3/3

This parameter specifies the domain mask included into transmitted multicast frames. It shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

2.7.7 PID_DOMAIN_ACCESS_MASK (PID=57)

- Property name: KNXnet/IP secure Domain Access Mask
- Property Datatype: PDT_BITSET_16
- Datapoint Type: None
- Default value: FFFFh
- Access: Data: 3/x
  Secure: 3/3

This parameter specifies the domain mask received multicast frames are checked against. It shall only be writable over a secure connection which itself was authenticated using the management password and/or using secured application layer services.

2.7.8 Manufacturing requirements for secure devices

The proposed solution requires a device specific seed for a cryptographically secure pseudo random number generator (CSPRNG) to be programmed into the device during the manufacturing process:

To implement a random number generator meeting the security requirements of the cryptographic algorithms used, a random seed which should be at least 128 bits long is needed. The individual seed shall be programmed into every device during manufacturing. The seed may be a true random number or it may be derived from a CSPRNG available in the production process. If a random number generator is not available in the production process, a hash of the device's private key might also be used as seed for the device internal CSPRNG.

Some IC vendors offer very small hardware devices to be placed on a PCB which contain a unique number. These devices may also be used to derive a random seed from.

2.7.9 Implementation considerations

2.7.9.1 Random number generation

The secure implementation of the proposed protocol – like any other cryptographic protocol - needs a source of cryptographically secure random numbers at runtime. Because in a typical embedded KNX device there is hardly any entropy to derive random numbers from, a software implemented cryptographically secure pseudo random number generator (CSPRNG) should be used for calculating random numbers.

For computing a different sequence of random numbers in every device, the CSPRNG must be seeded with a device specific seed. This seed may be a random number individually programmed into every device during manufacturing. Alternatively the device's individual private key might be used to seed the CSPRNG.

For computing a different sequence of random numbers in the same device after every power cycle, an additional seed must be used that changes with every power cycle. A simple persistent counter which is increased for every power cycle may be used to implement this. Alternatively the persistent secure IP group sequence counter detailed in the following section may be used.

Immediately after power-up, the RAM is likely to contain random values which might be used as a source of entropy.

During runtime any available entropy should be fed into the CSPRNG. A suitable source of entropy for this is the system timer value read on the occurrence of certain external events like receiving a frame from the KNX TP line or on the Ethernet connection.

2.7.9.2 Persisting the group sequence counter

The group sequence counter must be monotonically increasing at least every millisecond. It shall under no circumstances be decremented because this would weaken the resistance against replay attacks. To achieve this, the sequence counter must be persisted during power-off conditions. Even better it should be increased during power-off conditions using an RTC.

Depending on the actual hardware implementation there may be enough time after detecting a low power condition to persist the group counter to non-volatile (flash) memory.

If the hardware implementation cannot guarantee the above condition, the group sequence counter can be written to flash by software in regular intervals of e.g. one hour. After power-up when reading the stored sequence counter, the worst case is assumed that the value has been stored at the very beginning of an interval while the power was lost at the end of an interval. To ensure monotonic incrementing under this assumption, the counter difference corresponding to one interval is added to the counter just read.

Annex A
(informative)

Binary Examples of KNXnet/IP secure Frames

A.1 Example of SECURE_CHANNEL_REQUEST frame

A.1.1 Unencrypted Binary Example

```
      +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
 1    |              06h              |    header size
      +-------------------------------+
 2    |              13h              |    protocol version
      +-------------------------------+
 3    |              AAh              | \
      +- - - - - - - - - - - - - - - -+  > service type identifier AA01h
 4    |              01h              | /   SECURE_CHANNEL_REQUEST
      +-------------------------------+
 5    |              00h              | \
      +- - - - - - - - - - - - - - - -+  > total length, 50 octets
 6    |              32h              | /
      +-------------------------------+ - - - - - - - - - - - - - HPAI Control Endpoint
 7    |              08h              |    structure length
      +-------------------------------+
 8    |              01h              |    host protocol code, e.g. 01h for UDP over IPv4
      +-------------------------------+
 9    |              C0h              | \
      +- - - - - - - - - - - - - - - -+ |
10    |              A8h              | |
      +- - - - - - - - - - - - - - - -+  > IPv4 address of client's control endpoint,
11    |              C8h              | |   e.g. 192.168.200.12
      +- - - - - - - - - - - - - - - -+ |
12    |              0Ch              | /
      +-------------------------------+
13    |              C3h              | \
      +- - - - - - - - - - - - - - - -+  > UDP port number of client's control endpoint,
14    |              B4h              | /   e.g. 50100
      +-------------------------------+ - - - - - - - - - - - - - Unencrypted Data
15    |              ...              | \
      +- - - - - - - - - - - - - - - -+ |
...   |              ...              |  > Diffie-Hellman Client Public Value X
      +- - - - - - - - - - - - - - - -+ |
50    |              ...              | /
      +-------------------------------+
```

A.1.2 Encrypted Binary Example

TODO Add example when first (reference) implementation exists.
NOTE 1 The SECURE_CHANNEL_REQUEST frame itself is not encrypted, because this frame initiates the handshake process from unencrypted to encrypted communication. After all the ECDH parameters in this frame shall reflect a real-world binary example in the future.

A.2 Example of SECURE_CHANNEL_RESPONSE frame

A.2.1 Unencrypted Binary Example (Success)

```
      +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
  1   |              06h              |    header size
      +-------------------------------+
  2   |              13h              |    protocol version
      +-------------------------------+
  3   |              AAh              | \
      +- - - - - - - - - - - - - - - -+  > service type identifier AA02h
  4   |              02h              | /   SECURE_CHANNEL_RESPONSE
      +-------------------------------+
  5   |              00h              | \
      +- - - - - - - - - - - - - - - -+  > total length, 60 octets
  6   |              3Ch              | /
      +-------------------------------+ - - - - - - - - - - - - - - Unencrypted Data
  7   |              00h              | \
      +- - - - - - - - - - - - - - - -+  > Secure Channel Index, e.g. 01h
  8   |              01h              | /
      +-------------------------------+
  9   |              ...              | \
      +- - - - - - - - - - - - - - - -+ |
 ...  |              ...              |  > Diffie-Hellman Server Public Value Y
      +- - - - - - - - - - - - - - - -+ |
 44   |              ...              | /
      +-------------------------------+ - - - - - - - - - - - - - - Encrypted Data
 45   |              ...              | \
      +- - - - - - - - - - - - - - - -+ |
 ...  |              ...              |  > Message Authentication Code
      +- - - - - - - - - - - - - - - -+ |
 60   |              ...              | /
      +-------------------------------+
```

A.2.2 Unencrypted Binary Example (Error)

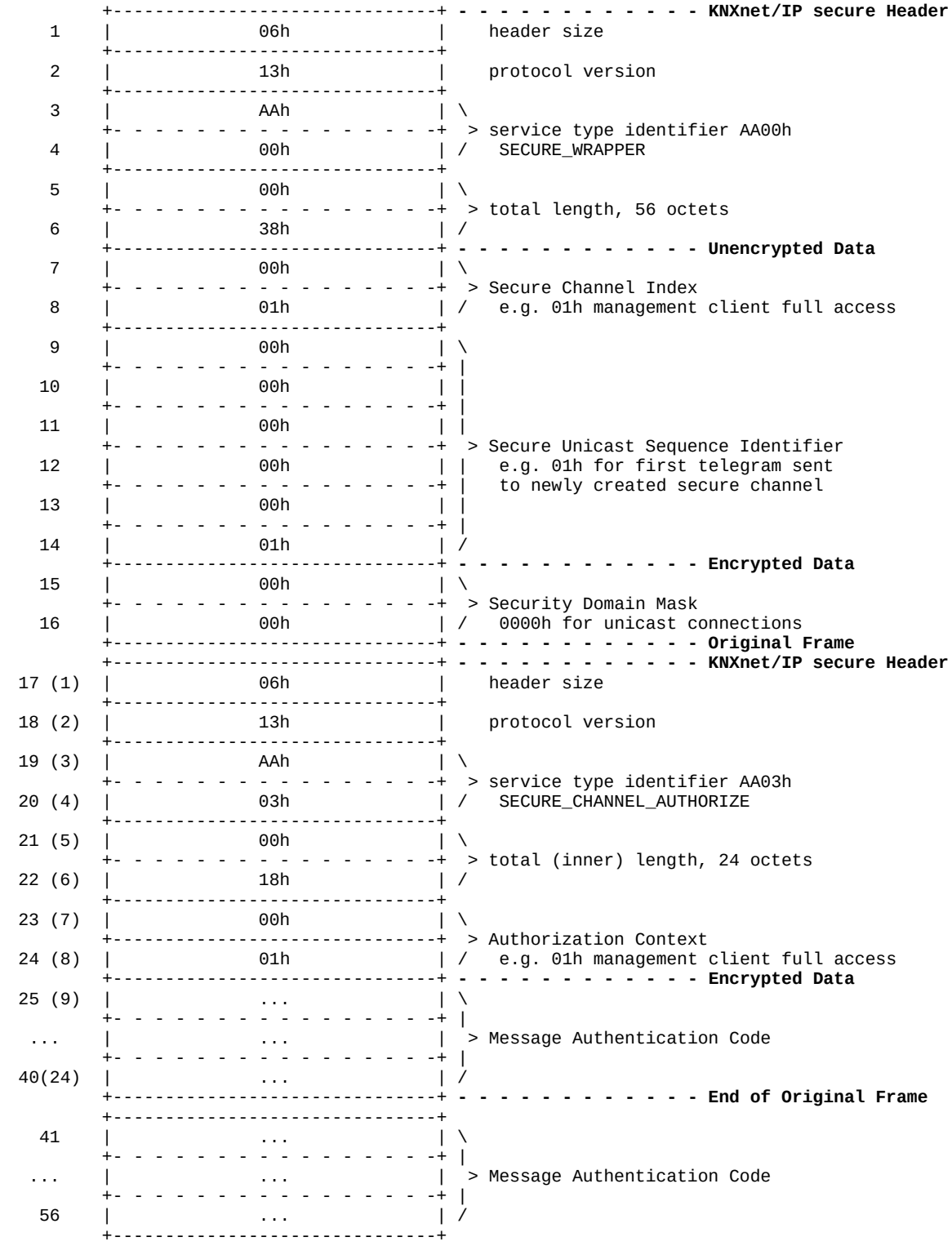
```
      +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
  1   |              06h              |    header size
      +-------------------------------+
  2   |              13h              |    protocol version
      +-------------------------------+
  3   |              AAh              | \
      +- - - - - - - - - - - - - - - -+  > service type identifier AA02h
  4   |              02h              | /   SECURE_CHANNEL_RESPONSE
      +-------------------------------+
  5   |              00h              | \
      +- - - - - - - - - - - - - - - -+  > total length, 8 octets
  6   |              08h              | /
      +-------------------------------+ - - - - - - - - - - - - - - Unencrypted Data
  7   |              00h              | \
      +- - - - - - - - - - - - - - - -+  > Secure Channel Index, 00h (error)
  8   |              00h              | /
      +-------------------------------+
```

A.2.3 Encrypted Binary Example (Success)

TODO Add example when first (reference) implementation exists.

A.3 Examples of SECURE_WRAPPER frames

A.3.1 Unencrypted SECURE_CHANNEL_AUTHORIZE frame



A.3.2 Encrypted SECURE_CHANNEL_AUTHORIZE frame

TODO Add example when first (reference) implementation exists.

A.3.3 Unencrypted SECURE_CHANNEL_STATUS frame

```
        +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
    1   |              06h              |    header size
        +-------------------------------+
    2   |              13h              |    protocol version
        +-------------------------------+
    3   |              AAh              | \
        +- - - - - - - - - - - - - - - -+  > service type identifier AA00h
    4   |              00h              | /   SECURE_WRAPPER
        +-------------------------------+
    5   |              00h              | \
        +- - - - - - - - - - - - - - - -+  > total length, 40 octets
    6   |              28h              | /
        +-------------------------------+ - - - - - - - - - - - - - Unencrypted Data
    7   |              00h              | \
        +- - - - - - - - - - - - - - - -+  > Secure Channel Index
    8   |              01h              | /
        +-------------------------------+
    9   |              00h              | \
        +- - - - - - - - - - - - - - - -+ |
   10   |              00h              | |
        +- - - - - - - - - - - - - - - -+ |
   11   |              00h              | |
        +- - - - - - - - - - - - - - - -+  > Secure Unicast Sequence Identifier
   12   |              00h              | |   e.g. 01h for first telegram sent
        +- - - - - - - - - - - - - - - -+ |   to newly created secure channel
   13   |              00h              | |
        +- - - - - - - - - - - - - - - -+ |
   14   |              01h              | /
        +-------------------------------+ - - - - - - - - - - - - - Encrypted Data
   15   |              00h              | \
        +- - - - - - - - - - - - - - - -+  > Security Domain Mask
   16   |              00h              | /   0000h for unicast connections
        +-------------------------------+ - - - - - - - - - - - - - Original Frame
        +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
17 (1)  |              06h              |    header size
        +-------------------------------+
18 (2)  |              13h              |    protocol version
        +-------------------------------+
19 (3)  |              AAh              | \
        +- - - - - - - - - - - - - - - -+  > service type identifier AA04h
20 (4)  |              04h              | /   SECURE_CHANNEL_STATUS
        +-------------------------------+
21 (5)  |              00h              | \
        +- - - - - - - - - - - - - - - -+  > total (inner) length, 8 octets
22 (6)  |              08h              | /
        +-------------------------------+
23 (7)  |              00h              |    status code 00h STATUS_AUTHORIZATION_SUCCESS
        +-------------------------------+
24 (8)  |              00h              |    Reserved
        +-------------------------------+ - - - - - - - - - - - - - End of Original Frame
        +-------------------------------+
   25   |              ...              | \
        +- - - - - - - - - - - - - - - -+ |
  ...   |              ...              |  > Message Authentication Code
        +- - - - - - - - - - - - - - - -+ |
   40   |              ...              | /
        +-------------------------------+
```

A.3.4 Encrypted SECURE_CHANNEL_STATUS frame

TODO Add example when first (reference) implementation exists.

A.3.5 Unencrypted ROUTING_INDICATION frame

	+-------------------------------+	- - - - - - - - - - - - -	**KNXnet/IP secure Header**
1	| 06h |		header size
	+-------------------------------+		
2	| 13h |		protocol version
	+-------------------------------+		
3	| AAh |	\	
	+- - - - - - - - - - - - - - -+	>	service type identifier AA00h
4	| 00h |	/	SECURE_WRAPPER
	+-------------------------------+		
5	| 00h |	\	
	+- - - - - - - - - - - - - - -+	>	total length, L+40 octets
6	| 28h |	/	
	+-------------------------------+	- - - - - - - - - - - - -	**Unencrypted Data**
7	| 00h |	\	
	+- - - - - - - - - - - - - - -+	>	Secure Channel Index
8	| 00h |	/	0000h for multicast messages
	+-------------------------------+		
9	| 00h |	\	
	+- - - - - - - - - - - - - - -+	|	
10	| 00h |	|	
	+- - - - - - - - - - - - - - -+	|	
11	| 4Eh |	|	
	+- - - - - - - - - - - - - - -+	>	Group Counter/Time Stamp
12	| 79h |	|	e.g. 1316565324
	+- - - - - - - - - - - - - - -+	|	
13	| 31h |	|	
	+- - - - - - - - - - - - - - -+	|	
14	| 4Ch |	/	
	+-------------------------------+	- - - - - - - - - - - - -	**Encrypted Data**
15	| 00h |	\	
	+- - - - - - - - - - - - - - -+	>	Security Domain Mask
16	| 01h |	/	sending to security group 0 by default
	+-------------------------------+	- - - - - - - - - - - - -	**Original Frame**
	+-------------------------------+	- - - - - - - - - - - - -	**KNXnet/IP Header**
17 (1)	| 06h |		header size
	+-------------------------------+		
18 (2)	| 10h |		protocol version
	+-------------------------------+		
19 (3)	| 05h |	\	
	+- - - - - - - - - - - - - - -+	>	service type identifier AA04h
20 (4)	| 30h |	/	SECURE_CHANNEL_STATUS
	+-------------------------------+		
21 (5)	| 00h |	\	
	+- - - - - - - - - - - - - - -+	>	total (inner) length, L+8 octets
22 (6)	| 08h |	/	
	+-------------------------------+	- - - - - - - - - - - - -	**cEMI Frame**
23 (7)	| **11h** |		message code (e.g. L_Data.req message)
	+-------------------------------+		
24 (8)	| **00h** |		additional information (none)
	+-------------------------------+		
25 (9)	| ... |	\	
	+-------------------------------+	|	
26(10)	| ... |	|	> Service Information (L bytes)
	+-------------------------------+	|	
L+24(L+8)	| ... |	/	
	+-------------------------------+	- - - - - - - - - - - - -	**End of Original Frame**
	+-------------------------------+		
L+25	| ... |	\	
	+- - - - - - - - - - - - - - -+	|	
...	| ... |	|	> Message Authentication Code
	+- - - - - - - - - - - - - - -+	|	
L+40	| ... |	/	
	+-------------------------------+		

A.3.6 Encrypted ROUTING_INDICATION frame

TODO Add example when first (reference) implementation exists.

A.4 Example of SECURE_GROUP_SYNC_REQUEST frame

A.4.1 Unencrypted Binary Example

```
      +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
 1    |              06h              |    header size
      +-------------------------------+
 2    |              13h              |    protocol version
      +-------------------------------+
 3    |              AAh              | \
      +- - - - - - - - - - - - - - - -+  > service type identifier AA06h
 4    |              06h              | /   SECURE_GROUP_SYNC_REQUEST
      +-------------------------------+
 5    |              00h              | \
      +- - - - - - - - - - - - - - - -+  > total length, 28 octets
 6    |              1Ch              | /
      +-------------------------------+ - - - - - - - - - - - - - - Unencrypted Data
 7    |              00h              | \
      +- - - - - - - - - - - - - - - -+ |
 8    |              00h              | |
      +- - - - - - - - - - - - - - - -+ |
 9    |              00h              | |
      +- - - - - - - - - - - - - - - -+  > time stamp
10    |              00h              | |   e.g. 1
      +- - - - - - - - - - - - - - - -+ |
11    |              00h              | |
      +- - - - - - - - - - - - - - - -+ |
12    |              01h              | /
      +-------------------------------+ - - - - - - - - - - - - - - Encrypted Data
13    |              ...              | \
      +- - - - - - - - - - - - - - - -+ |
...   |              ...              |  > Message Authentication Code
      +- - - - - - - - - - - - - - - -+ |
28    |              ...              | /
      +-------------------------------+
```

A.4.2 Encrypted Binary Example

TODO Add example when first (reference) implementation exists.

A.5 Example of SECURE_GROUP_SYNC_RESPONSE frame

A.5.1 Unencrypted Binary Example

```
      +-------------------------------+ - - - - - - - - - - - - - KNXnet/IP secure Header
  1   |              06h              |    header size
      +-------------------------------+
  2   |              13h              |    protocol version
      +-------------------------------+
  3   |              AAh              | \
      +- - - - - - - - - - - - - - - -+  > service type identifier AA07h
  4   |              07h              | /   SECURE_GROUP_SYNC_RESONSE
      +-------------------------------+
  5   |              00h              | \
      +- - - - - - - - - - - - - - - -+  > total length, 28 octets
  6   |              1Ch              | /
      +-------------------------------+ - - - - - - - - - - - - - - - Unencrypted Data
  7   |              00h              | \
      +- - - - - - - - - - - - - - - -+ |
  8   |              00h              | |
      +- - - - - - - - - - - - - - - -+ |
  9   |              4Eh              | |
      +- - - - - - - - - - - - - - - -+  > time stamp
 10   |              79h              | |   e.g. 1316565326
      +- - - - - - - - - - - - - - - -+ |
 11   |              31h              | |
      +- - - - - - - - - - - - - - - -+ |
 12   |              4Eh              | /
      +-------------------------------+ - - - - - - - - - - - - - - - Encrypted Data
 13   |              ...              | \
      +- - - - - - - - - - - - - - - -+ |
...   |              ...              |  > Message Authentication Code
      +- - - - - - - - - - - - - - - -+ |
 28   |              ...              | /
      +-------------------------------+
```

A.5.2 Encrypted Binary Example

TODO Add example when first (reference) implementation exists.