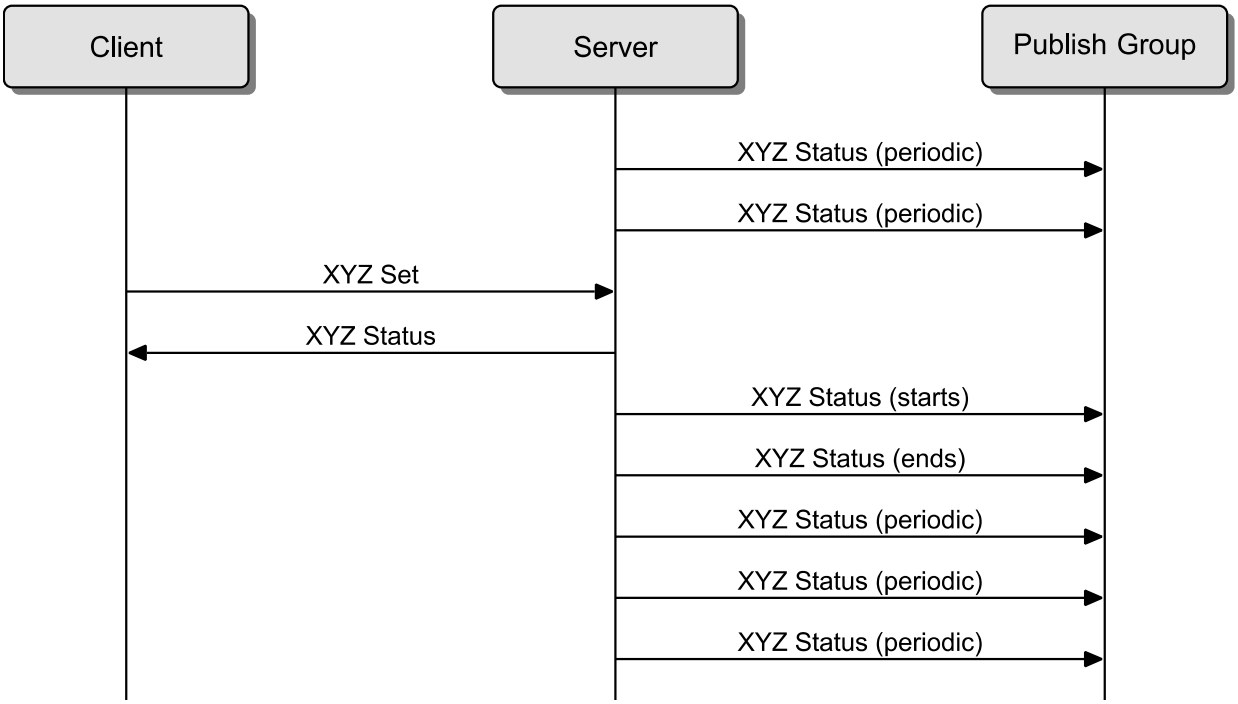

Figure 3.30: Acknowledged set message with periodic publishing

3.8 Mesh security

This section describes how mesh security is implemented.

3.8.1 Endianness

All multiple-octet numeric values in this layer shall be marshalled in “big endian,”, as described in Section 3.1.1.1.

3.8.2 Security toolbox

This section describes the eight functions that together provide a security toolbox for mesh networking.

3.8.2.1 Encryption function

The same encryption function *e*, as defined in Volume 3, Part H, Section 2.2.1 of the Core Specification [1], shall be used. This can be summarized as:

ciphertext = e(key, plaintext)

3.8.2.2 CMAC function

RFC4493 [9] defines the Cipher-based Message Authentication Code (CMAC) that uses AES-128 as the block cipher function, also known as AES-CMAC. The inputs to AES-CMAC are:

k is the 128-bit key

m is the variable length data to be authenticated

The 128-bit message authentication code (MAC) is generated[1] as follows:

$$MAC = AES\text{-}CMAC_k(m)$$

A node can implement AES functions in the host or can use the HCI_LE_Encrypt command (see Volume 2, Part E, Section 7.8.22 of the Core Specification [1]) in order to use the AES function in the controller.

3.8.2.3 CCM function

RFC3610 [10] defines the AES Counter with CBC-MAC (CCM) (see Volume 6, Part E, Section 1 of the Core Specification [1]). This specification defines AES-CCM as a function that takes four inputs and results in two outputs:

The inputs to AES-CCM are:

k is the 128-bit key

n is a 104-bit nonce

m is the variable length data to be encrypted and authenticated – also known as “plaintext”

a is the variable length data to be authenticated – also known as “Additional Data”

The ciphertext and mic are generated as follows:

$$ciphertext, mic = AES\text{-}CCM_k(n, m, a)$$

Where:

ciphertext is the variable length data after it has been encrypted

mic is the message integrity check value of m and a – also known as the “Message Authentication Code” or the encrypted authentication value U in RFC3610 [10].

If only the k, n, and m parameters are provided to the AES-CCM, then the additional data shall be zero length.

3.8.2.4 s1 SALT generation function

The inputs to function s1 are:

M is a non-zero length octet array or ASCII encoded string

[1] This is using the same notation used in other Bluetooth specifications. This is functionally the same as the notation as RFC 4493, where MAC = AES-CMAC(k, m).

If M is an ASCII encoded string, it shall be converted into an octet array by replacing each string character with its ASCII code preserving the order. For example, if M is the string “MESH”, this is converted into the octet array: 0x4d, 0x45, 0x53, 0x48.

ZERO is the 128-bit value:

0x0000 0000 0000 0000 0000 0000 0000 0000

The output of the salt generation function s1 is as follows:

$s1(M) = AES\text{-}CMAC_{ZERO}(M)$

3.8.2.5 k1 derivation function

The network key material derivation function k1 is used to generate instances of IdentityKey and BeaconKey.

The definition of this key generation function makes use of the MAC function $AES\text{-}CMAC_T$ with a 128-bit key T.

The inputs to function k1 are:

N is 0 or more octets

SALT is 128 bits

P is 0 or more octets

The key (T) is computed as follows:

$T = AES\text{-}CMAC_{SALT}(N)$

The output of the key generation function k1 is as follows:

$k1(N, SALT, P) = AES\text{-}CMAC_T(P)$

3.8.2.6 k2 network key material derivation function

The network key material derivation function k2 is used to generate instances of EncryptionKey, PrivacyKey, and NID for use as Master and Private Low Power node communication.

The definition of this key generation function makes use of the MAC function $AES\text{-}CMAC_T$ with a 128-bit key T.

The inputs to function k2 are:

N is 128 bits

P is 1 or more octets

The key (T) is computed as follows:

$T = AES\text{-}CMAC_{SALT}(N)$

SALT is the 128-bit value computed as follows

SALT = s1(“smk2”)

The output of the key generation function k2 is as follows:

T0 = empty string (zero length)
T1 = AES-CMAC$_T$ (T0 || P || 0x01)
T2 = AES-CMAC$_T$ (T1 || P || 0x02)
T3 = AES-CMAC$_T$ (T2 || P || 0x03)

k2(N, P) = (T1 || T2 || T3) mod 2^{263}

3.8.2.7 k3 derivation function

The derivation function k3 is used to generate a public value of 64 bits derived from a private key.

The definition of this derivation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k3 are:

N is 128 bits

The key (T) is computed as follows:

T = AES-CMAC$_{SALT}$ (N)

SALT is a 128-bit value computed as follows:

SALT = s1(“smk3”)

The output of the derivation function k3 is as follows:

k3(N) = AES-CMAC$_T$ (“id64” || 0x01) mod 2^{64}

3.8.2.8 k4 derivation function

The derivation function k4 is used to generate a public value of 6 bits derived from a private key.

The definition of this derivation function makes use of the MAC function AES-CMAC$_T$ with a 128-bit key T.

The inputs to function k4 are:

N is 128 bits

The key (T) is computed as follows:

T = AES-CMAC$_{SALT}$ (N)

SALT is a 128-bit value computed as follows:

SALT = s1(“smk4”)

The output of the derivation function k4 is as follows:

K4(N) = AES-CMAC$_T$ (“id6” || 0x01) mod 2^6

3.8.3 Sequence number

The sequence number, a 24-bit value contained in the SEQ field of the Network PDU, is primarily designed to protect against replay attacks. Elements within the same node may or may not share the sequence number space with each other. Having a different sequence number in each new Network PDU for every message source (identified by the unicast address contained in the SRC field) is critical for the security of the mesh network.

With a 24-bit sequence number, an element can transmit 16,777,216 messages before repeating a nonce. If an element transmits a message on average once every five seconds (representing a fairly high frequency message for known use cases), the element can transmit for 2.6 years before the nonce repeats.

Each element shall use strictly increasing sequence numbers for the Network PDUs it generates. Before the sequence number approaches the maximum value (0xFFFFFF), the element shall update the IV Index using the IV Update procedure (see Section 3.10.5). This is done to ensure that the sequence number will never wrap around.

3.8.4 IV Index

The IV Index is a 32-bit value that is a shared network resource (i.e., all nodes in a mesh network share the same value of the IV Index and use it for all subnets they belong to).

The IV Index starts at 0x00000000 and is incremented during the IV Update procedure as described in Section 3.10.5. The timing when the value is incremented does not have to be exact, since the least significant bit is communicated within every Network PDU. Since the IV Index value is a 32-bit value, a mesh network can function approximately 5 trillion years before the IV Index will wrap.

The IV Index is shared within a network via Secure Network beacons (see Section 3.9.3). IV updates received on a subnet are processed and propagated to that subnet. The propagation happens by the device transmitting Secure Network beacons with the updated IV Index for that particular subnet. If a device on a primary subnet receives an update on the primary subnet, it shall propagate the IV update to all other subnets. If a device on a primary subnet receives an IV update on any other subnet, the update shall be ignored.

If a node is absent from a mesh network for a period of time, it can scan for Secure Network beacons (see Section 3.10.1) or use the IV Index Recovery procedure (see Section 3.10.6), and therefore set the IV Index value autonomously.

3.8.5 Nonce

The nonce is a 13-octet value that is unique for each new message encryption. There are four different nonces that are used, as shown in Table 3.44. The type of the nonce is determined by the first octet of the nonce, referred to as the Nonce Type.

Nonce Type	Nonce	Description
0x00	Network nonce	Used with an encryption key for network authentication and encryption
0x01	Application nonce	Used with an application key for upper transport authentication and encryption
0x02	Device nonce	Used with a device key for upper transport authentication and encryption
0x03	Proxy nonce	Used with an encryption key for proxy authentication and encryption
0x04–0xFF	RFU	Reserved for Future Use

Table 3.44: Nonce types

Note: The TTL is used within the network nonce but not within the application nonce, device nonce, or proxy nonce. This means that when a message is relayed and the TTL is decremented, the application nonce or device nonce does not change; however, the network nonce does change, allowing the authentication of the TTL value.

Note: The DST is used within the application nonce and device nonce but not in the network nonce. This means that the destination of the application or device message may be authenticated, but at the network layer the destination address is encrypted.

3.8.5.1 Network nonce

The network nonce is defined in Table 3.45 and illustrated in Figure 3.31.

Field	Size (octets)	Notes
Nonce Type	1	0x00
CTL and TTL	1	See Table 3.46
SEQ	3	Sequence Number
SRC	2	Source Address
Pad	2	0x0000
IV Index	4	IV Index

Table 3.45: Network nonce format

Field	Size (bits)	Notes
CTL	1	See Section 3.4.4.3
TTL	7	See Section 3.4.4.4

Table 3.46: CTL and TTL field format

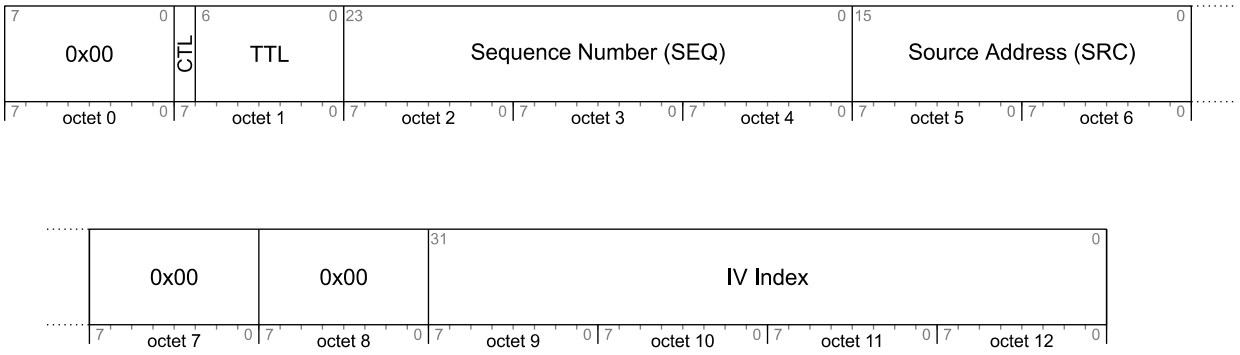

Figure 3.31: Network nonce format

The network nonce is used with an encryption key for network data authentication and encryption (see Section 3.8.7.2).

3.8.5.2 Application nonce

The application nonce is defined in Table 3.47 and illustrated in Figure 3.32.

Field	Size (octets)	Notes
Nonce Type	1	0x01
ASZMIC and Pad	1	See Table 3.48
SEQ	3	Sequence Number of the Access message (24 lowest bits of SeqAuth in the context of segmented messages)
SRC	2	Source Address
DST	2	Destination Address
IV Index	4	IV Index

Table 3.47: Application nonce format

Field	Size (bits)	Notes
ASZMIC	1	SZMIC if a Segmented Access message or 0 for all other message formats
Pad	7	0b0000000

Table 3.48: ASZMIC and Pad field format

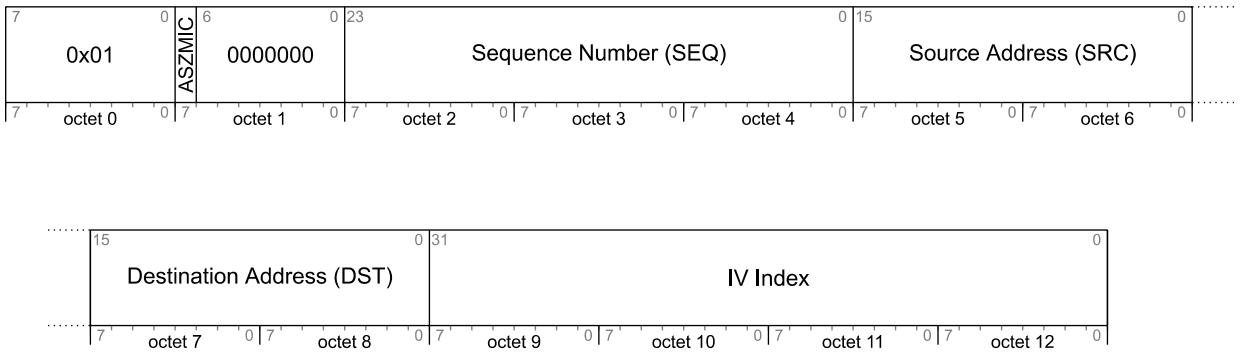

Figure 3.32: Application nonce format

The application nonce is used with an application key for application data authentication and encryption (see Section 3.8.6).

3.8.5.3 Device nonce

The device nonce is defined in Table 3.49 and illustrated in Figure 3.33.

Field	Size (octets)	Notes
Nonce Type	1	0x02
ASZMIC and Pad	1	See Table 3.50
SEQ	3	Sequence Number of the Access message (24 lowest bits of SeqAuth in the context of segmented messages)
SRC	2	Source Address
DST	2	Destination Address
IV Index	4	IV Index

Table 3.49: Device nonce format

Field	Size (bits)	Notes
ASZMIC	1	SZMIC if a Segmented Access message or 0 for all other message formats
Pad	7	0b0000000

Table 3.50: ASZMIC and Pad field format

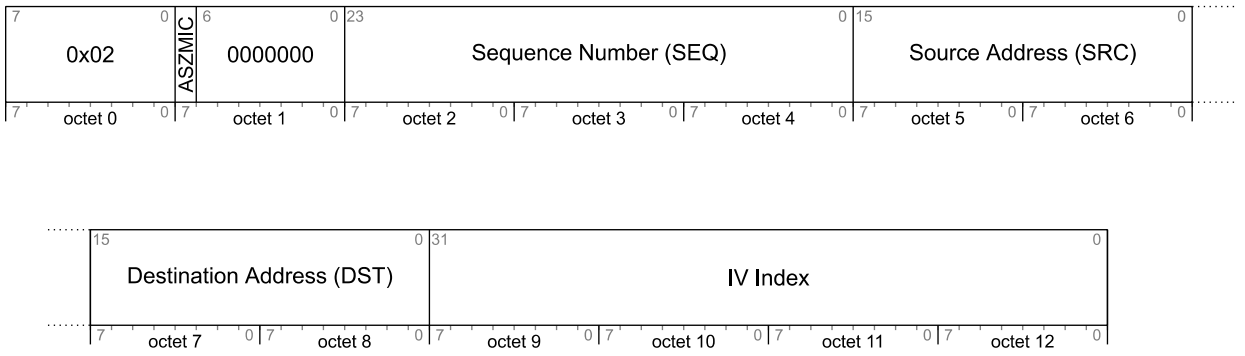

Figure 3.33: Device nonce format

The device nonce is used with a device key for application data authentication and encryption specific for a given device (see Section 3.8.6).

3.8.5.4 Proxy nonce

The proxy nonce is defined in Table 3.51 and illustrated in Figure 3.34.

Field	Size (octets)	Notes
Nonce Type	1	0x03
Pad	1	0x00
SEQ	3	Sequence Number
SRC	2	Source Address
Pad	2	0x0000
IV Index	4	IV Index

Table 3.51: Proxy nonce format

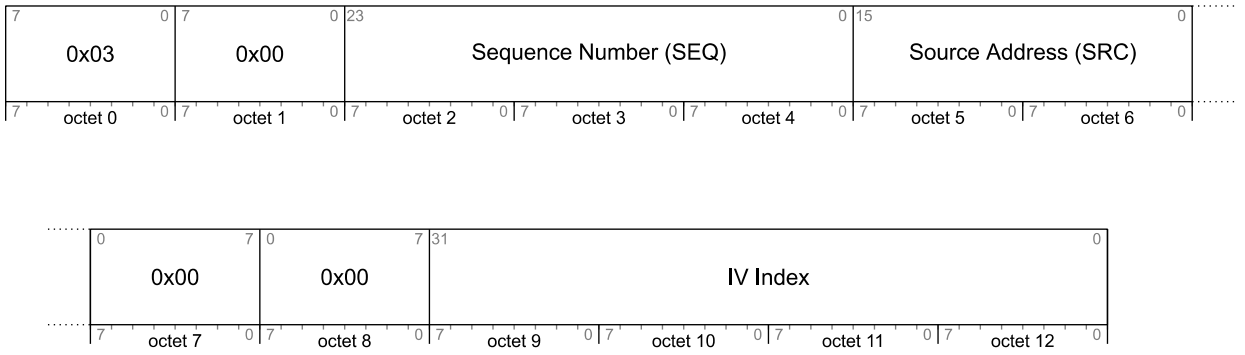

Figure 3.34: Proxy nonce format

The proxy nonce is used with an encryption key for proxy configuration message authentication and encryption (see Section 6.5).

3.8.6 Keys

The Mesh Profile specification defines two types of keys: application keys (AppKey) and network keys (NetKey). AppKeys are used to secure communications at the upper transport layer and NetKeys are used to secure communications at the network layer. Both types of keys are shared between nodes. There is also a device key (DevKey), which is a special application key that is unique to each node, is known only to the node and a Configuration Client, and is used to secure communications between the node and a Configuration Client.

Application keys are bound to network keys. This means application keys are only used in a context of a network key they are bound to. An application key shall only be bound to a single network key. A device key is implicitly bound to all network keys.

An example of binding application keys to network keys and models is illustrated in Figure 3.35.

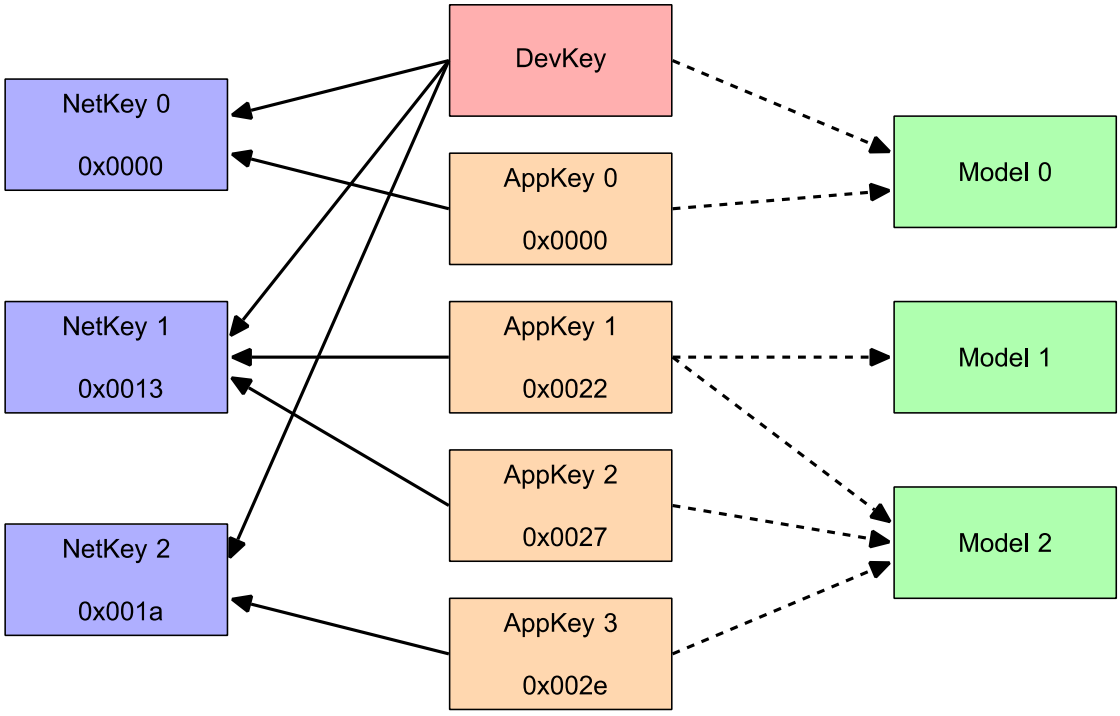

Figure 3.35: Application key binding example

3.8.6.1 Device key

The device key (DevKey) is an access layer key known only to the node and a Configuration Client. The device key shall be bound to every network key known to the node. Those bindings cannot be changed. An illustration of the device key derivation is shown in Figure 3.36.

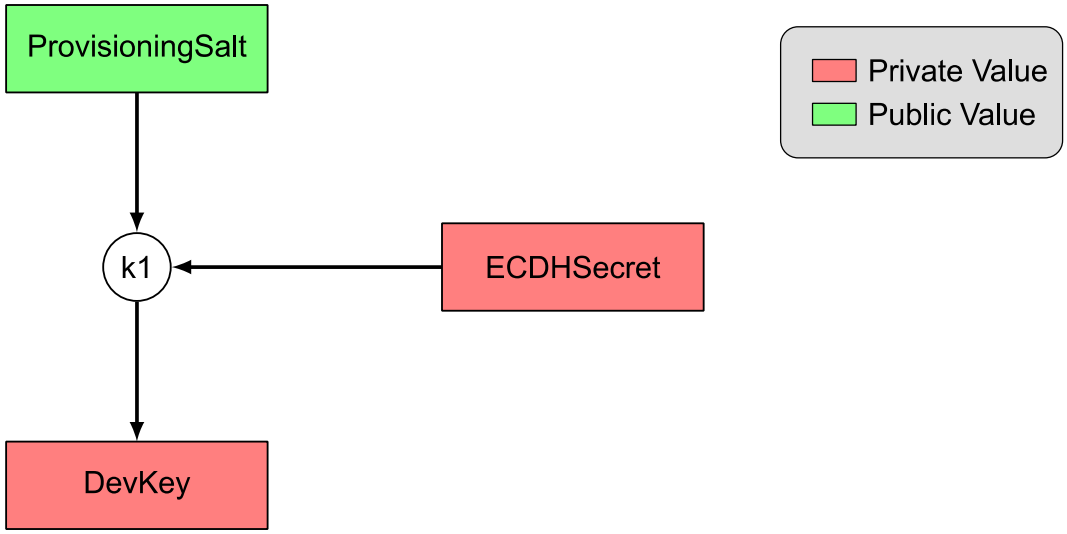

Figure 3.36: Device key derivation

The DevKey shall be derived from the ECDHSecret and ProvisioningSalt as described by the formula below:

DevKey = k1(ECDHSecret, ProvisioningSalt, “prdk”)

The ProvisioningSalt is defined in Section 5.4.2.5 and the ECDHSecret is defined in Section 5.4.2.3.

3.8.6.2 Application key

The application key (AppKey) shall be generated using a random number generator compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1].

The application key identifier (AID) is used to identify the application key. An illustration of the AID derivation is shown in Figure 3.37.

AID= k4(AppKey)

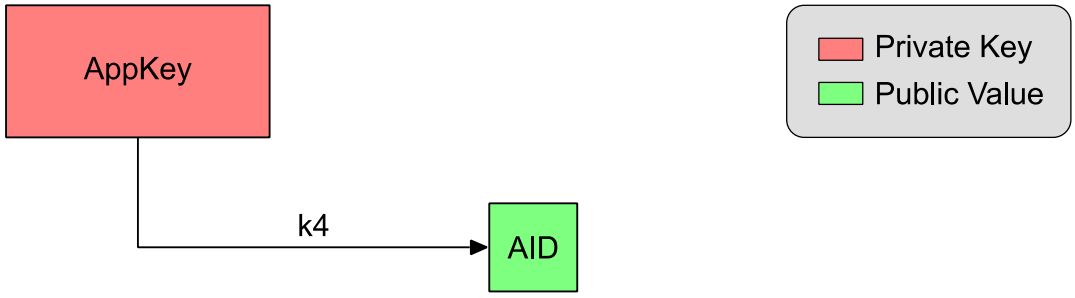

Figure 3.37: AID derivation

3.8.6.3 Network key

The network key (NetKey) shall be generated using a random number generator compatible with the requirements in Volume 2, Part H, Section 2 of the Core Specification [1]. An illustration of the network key hierarchy is shown in Figure 3.38.

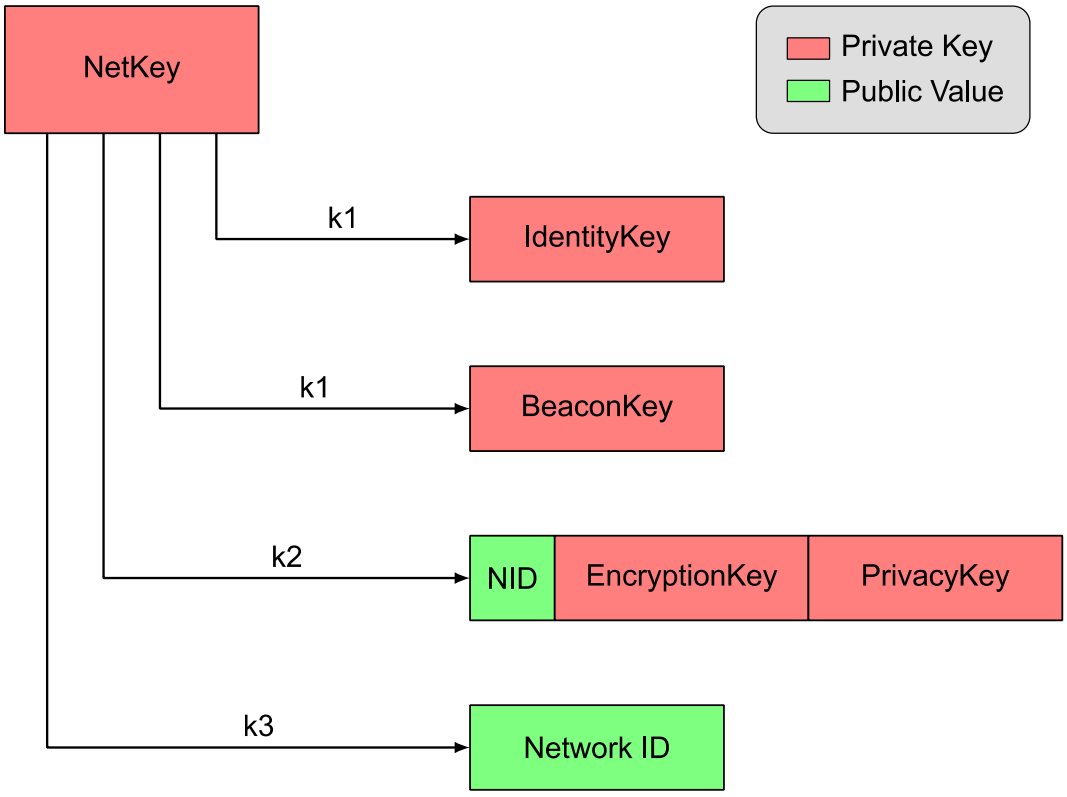

Figure 3.38: Network key hierarchy

3.8.6.3.1 NID, Encryption Key, and Privacy Key

Each Network PDU is secured using security material that is composed of the NID, the Encryption Key, and the Privacy Key.

The NID is a 7-bit value that identifies the security material that is used to secure this Network PDU.

Note: There are up to 2^{121} possible keys for each NID; therefore, the NID value can only provide an indication of the security material that has been used to secure this Network PDU.

The NID, EncryptionKey, and PrivacyKey are derived using the k2 function with security credentials as inputs.

The master security material is derived from the master security credentials using:

NID || EncryptionKey || PrivacyKey = k2(NetKey, 0x00)

The friendship security material is derived from the friendship security credentials using:

NID || EncryptionKey || PrivacyKey = k2(NetKey, 0x01 || LPNAddress || FriendAddress || LPNCounter || FriendCounter)

Where:

The LPNAddress value is the unicast address set as source address in the Friend Request message that set up the friendship.

The FriendAddress value is the unicast address set as source address in the Friend Offer message that set up the friendship.

The LPNCounter value is the value from the LPNCounter field of the Friend Request message that set up the friendship.

The FriendCounter is the value from the FriendCounter field of the Friend Offer message that set up the friendship.

For Network PDUs that are sent between a Low Power node and Friend node that have a friendship relationship, the friendship security material is used.

For all other Network PDUs, the master security materials are used.

3.8.6.3.2 Network ID

The Network ID is derived from the network key such that each network key generates one Network ID. This identifier becomes public information.

Network ID = k3 (NetKey)

3.8.6.3.3 IdentityKey

The IdentityKey is derived from the network key such that each network key generates one IdentityKey.

salt = s1(“nkik”)

P = “id128” || 0x01

IdentityKey = k1 (NetKey, salt, P)

3.8.6.3.4 BeaconKey

The BeaconKey is derived from the network key such that each network key generates one BeaconKey.

salt = s1(“nkbk”)

P = “id128” || 0x01

BeaconKey = k1 (NetKey, salt, P)

3.8.6.4 Global key indexes

Network and application keys are organized within the mesh network into two lists, maintained by a Configuration Client: a list of network keys and a list of application keys. Each list is a shared mesh network resource and can accommodate up to 4096 keys. Keys are referenced using global key indexes: the NetKey Index and the AppKey Index. The key indexes are 12-bit values ranging from 0x000 to 0xFFF inclusive. A network key at index 0x000 is called the primary NetKey.

3.8.7 Message security

Messages are secured using AES-CCM at two different layers. Messages are encrypted and authenticated at the network layer and at the upper transport layer. Each message is also obfuscated to hide possible identifying information from the packets. This is illustrated in Figure 3.39.

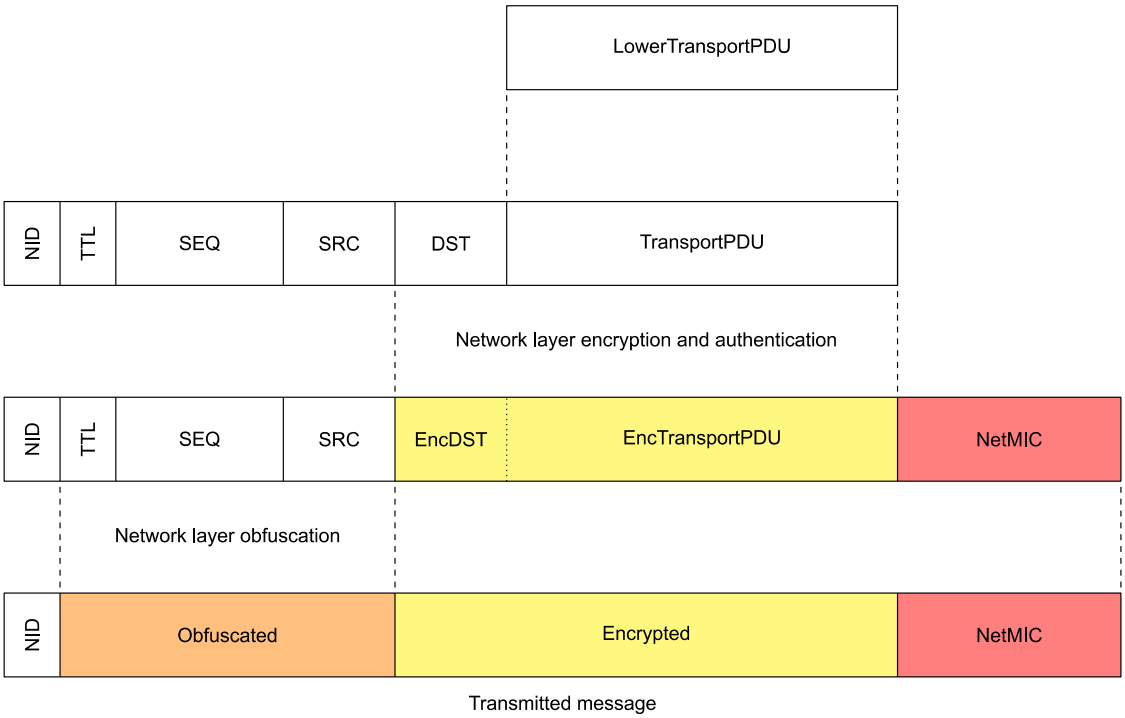

Figure 3.39: Example of network layer encryption, authentication, and obfuscation

Every message has a minimum of 64 bits of authentication information associated with it. This authentication information may be split between the network layer and upper transport layer.

Some messages, known as control messages, are not authenticated at the upper transport layer and therefore have a 64-bit NetMIC. Access messages are authenticated at the upper transport layer and therefore have a 32-bit NetMIC. Access messages that are sent in a single unsegmented message have a 32-bit TransMIC. Access messages that are segmented over multiple Network PDUs can have either a 32-bit or 64-bit TransMIC. This allows a higher layer to determine the level of authentication required to securely deliver the access message and therefore apply the appropriate size for the TransMIC.

3.8.7.1 Upper transport layer authentication and encryption

Authentication and encryption of the access payload is performed by the upper transport layer.

The access payload is encrypted and authenticated using AES-CCM. This is identical to the way that Bluetooth low energy encryption and authentication works. An illustration of the upper transport layer encryption is shown in Figure 3.40.

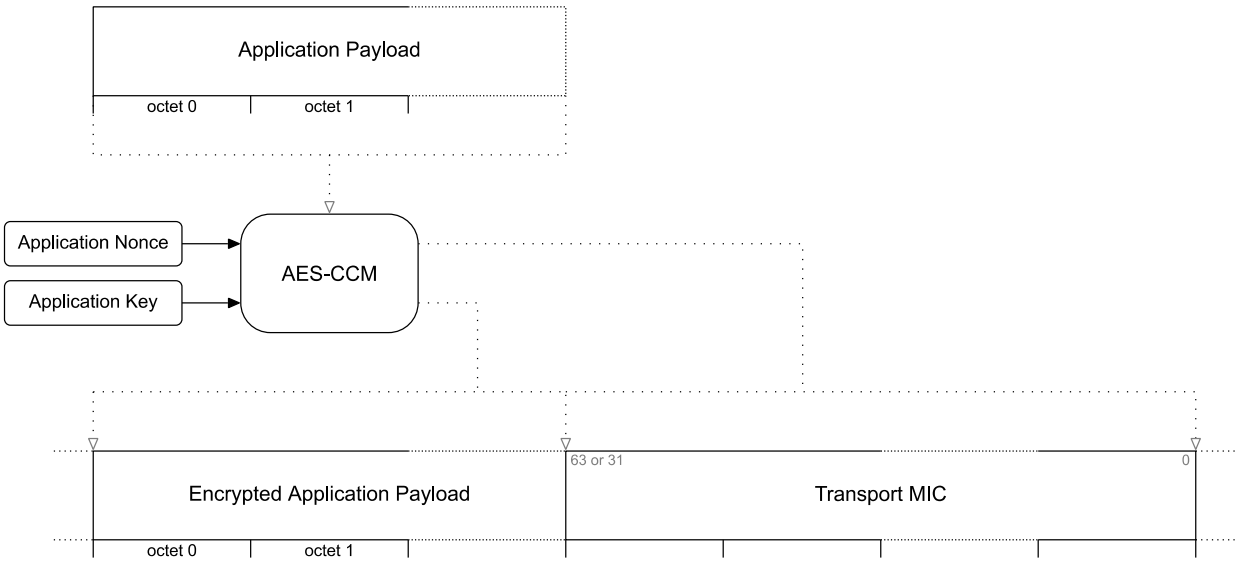

Figure 3.40: Upper Transport layer encryption

If the access payload is secured using the application key, then the access payload is encrypted using the application nonce and the application key.

If the access payload is secured using the device key, then the access payload is encrypted using the device nonce and the device key.

The nonce uses the sequence number and the source address, ensuring that two different nodes cannot use the same nonce. The IV Index is used to provide significantly more nonce values than the sequence number can provide for a given node. Management of the IV Index is described in Section 3.10.5.

Note: The authentication and encryption of the access payload is not dependent on the TTL value, meaning that as the access payload is relayed through a mesh network, the access payload does not need to be re-encrypted at each hop.

When using an application key and the destination address is a virtual address:

EncAccessPayload, TransMIC = AES-CCM (AppKey, Application Nonce, AccessPayload, Label UUID)

When using an application key and the destination address is a unicast address or a group address:

EncAccessPayload, TransMIC = AES-CCM (AppKey, Application Nonce, AccessPayload)

When using a device key and the destination address is a unicast address:

EncAccessPayload, TransMIC = AES-CCM (DevKey, Device Nonce, AccessPayload)

The concatenation of the encrypted access payload and the transport MIC is called the Upper Transport PDU:

Upper Transport PDU = EncAccessPayload || TransMIC

3.8.7.2 Network layer authentication and encryption

The destination address and the TransportPDU are encrypted and authenticated using AES-CCM. This is identical to the way that Bluetooth low energy encryption and authentication works.

All Network PDUs are encrypted using an Encryption Key that is derived from a network key (see Section 3.8.6.3.1).

An illustration of the network layer encryption is shown in Figure 3.41.

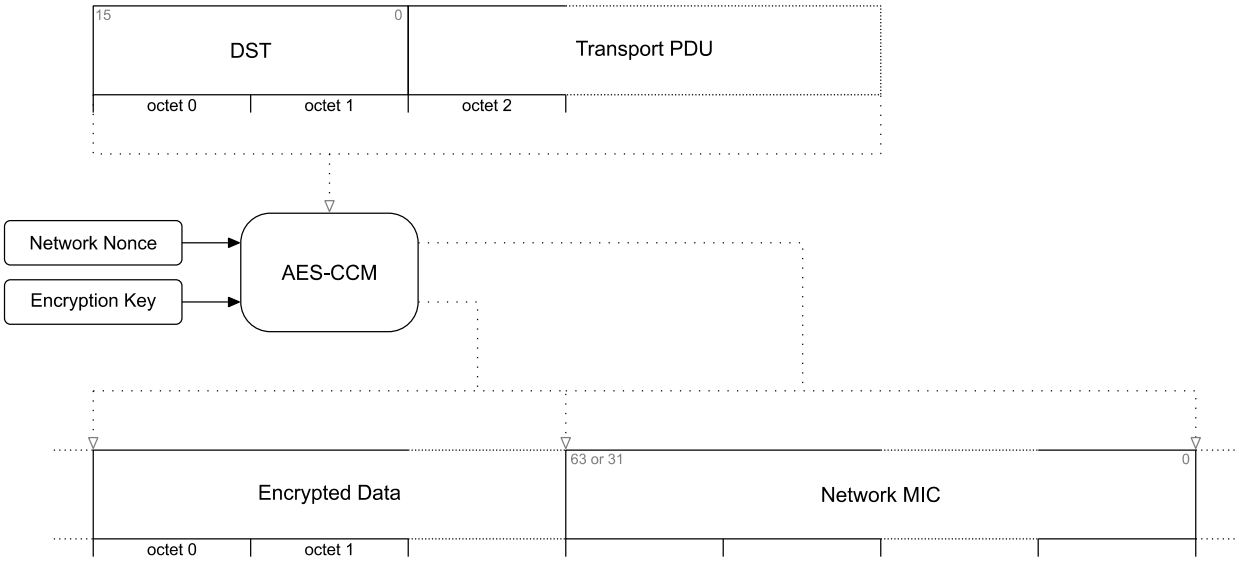

Figure 3.41: Network layer encryption

The following defines how this is performed:

EncDST || EncTransportPDU, NetMIC = AES-CCM (EncryptionKey, Network Nonce, DST || TransportPDU)

3.8.7.3 Network layer obfuscation

In order to obfuscate the Network Header (CTL, TTL, SEQ, SRC), these values shall be combined with a result of a single encryption function e, designed to prevent a passive eavesdropper from determining the identity of a node by listening to Network PDUs.

The obfuscation occurs after the application and network message integrity check values have been calculated. The obfuscation is calculated using information available from within the Network PDU. This obfuscation is designed only to help prevent a simple passive eavesdropper from tracking nodes. A determined attacker could still discover patterns within this obfuscation that can lead to the revealing of the source address or sequence number of a node. Critically, obfuscation does not enforce that inputs to the encryption function are unique.

Obfuscation does not protect the Privacy Key from compromise, and given the above design considerations for protection against only passive eavesdroppers, it is considered that the Privacy Key could be compromised with sufficient time. The design of obfuscation includes the IV Index, such that when the IV Index changes, any obfuscation attacks would have to start again.

To obfuscate the Network PDU, the first seven octets of the Network PDU that have already been encrypted are combined with the IV Index and a Privacy Key.

These first seven octets of the Network PDU that have been encrypted include both the EncDST and five octets of either the EncTransportPDU or the EncTransportPDU concatenated with the NetMIC. These octets are known as the PrivacyRandom value.

The Privacy Key is derived using a key derivation function from the network key (see Section 3.8.6.3.1) to protect the network key even if the Privacy Key is compromised.

The IV Index is concatenated with the PrivacyRandom value and used along with the Privacy Key as inputs to the encryption function e. The output of this is known as the PECB value.

The first six octets of the PECB value is then exclusive-ORed with the TTL octet, the sequence number, and the source address fields, and become the ObfuscatedData. The Network PDU is transmitted as the concatenation of the NID/IVI octet, the ObfuscatedData, the EncDST, the EncTransportPDU, and the NetMIC.

An illustration of the network layer obfuscation is shown in Figure 3.42.

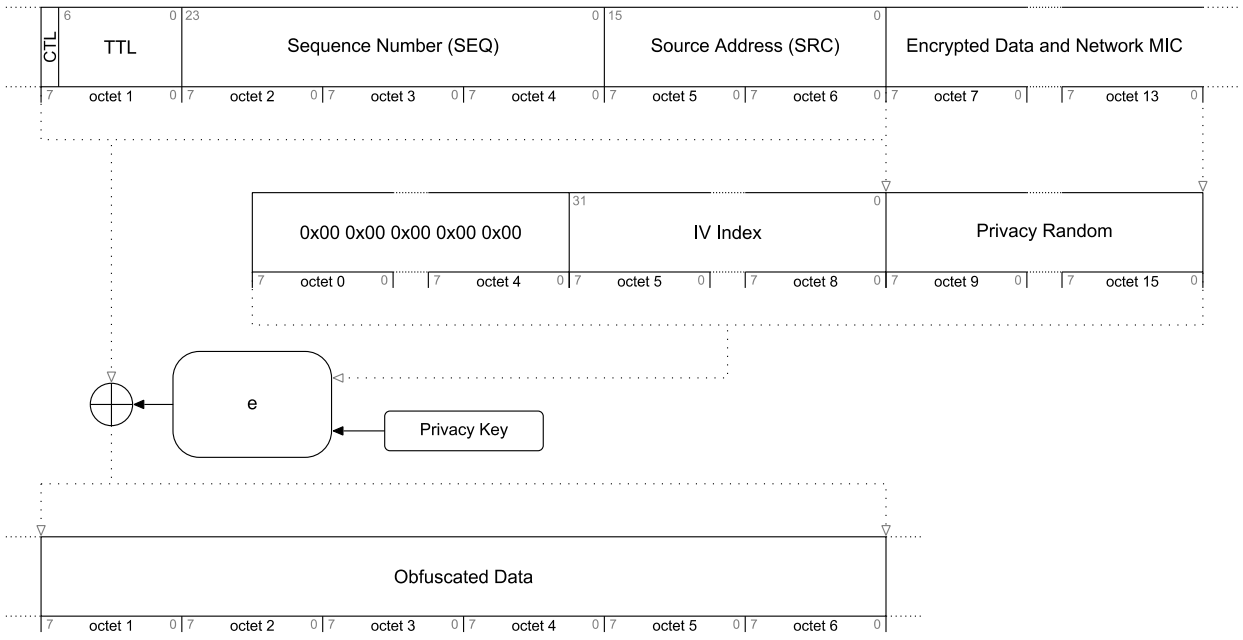

Figure 3.42: Network layer obfuscation

Privacy Random = (EncDST || EncTransportPDU || NetMIC)[0–6]

Privacy Plaintext = 0x0000000000 || IV Index || Privacy Random

PECB = e (PrivacyKey, Privacy Plaintext)

ObfuscatedData = (CTL || TTL || SEQ || SRC) ⊕ PECB[0–5]

When reversing this, the following operations are performed:

Privacy Random = (EncDST || EncTransportPDU || NetMIC)[0–6]

Privacy Plaintext = 0x0000000000 || IV Index || Privacy Random

PECB = e (PrivacyKey, Privacy Plaintext)

(CTL || TTL || SEQ || SRC) = ObfuscatedData ⊕ PECB[0–5]

3.8.8 Message replay protection

A message sent by a legitimate originating element can be passively received by an attacker and then replayed later without modification. This is called a replay attack.

Since the originating element has encrypted and authenticated the message using the correct keys, the receiver cannot determine whether it is under a replay attack solely by performing the message integrity checks (i.e., on the Network MIC and, if applicable, on the Transport MIC).

To increase protection against replay attacks, each element increases the sequence number for each new message that it sends. If a valid message has been received from an originating element with a specific sequence number, any future messages from the same originating element that contain numerically lower or equal sequence numbers than the last valid sequence number are very likely replayed messages and shall be discarded. Therefore, messages are delivered to the access layer in sequence number order.

If a lower IV Index from the same originating element has been received, the message shall be discarded.

A node shall implement replay protection for all Access and Control messages that are received from other elements, as well as for Proxy Configuration messages, if applicable.

If a node does not have enough resources to perform replay protection for a given source address, then the node shall discard the message immediately upon reception.

An implementation may perform the replay protection at any layer and in any order with respect to the message authentication steps (the network layer decryption and the transport layer decryption), in order to optimize the message processing flow, the number of cryptographic operations or the memory usage.

However, the implementation shall follow the fundamental requirement that it shall either be able to determine if a certain message is being replayed, or it shall discard the message immediately upon reception.

Figure 3.43 illustrates an example of a replay protection list implementation that handles a multi-segment message transaction which is under a replay attack. The sequence number of the last segment that has been received for this message is stored for that peer node in the replay protection list.

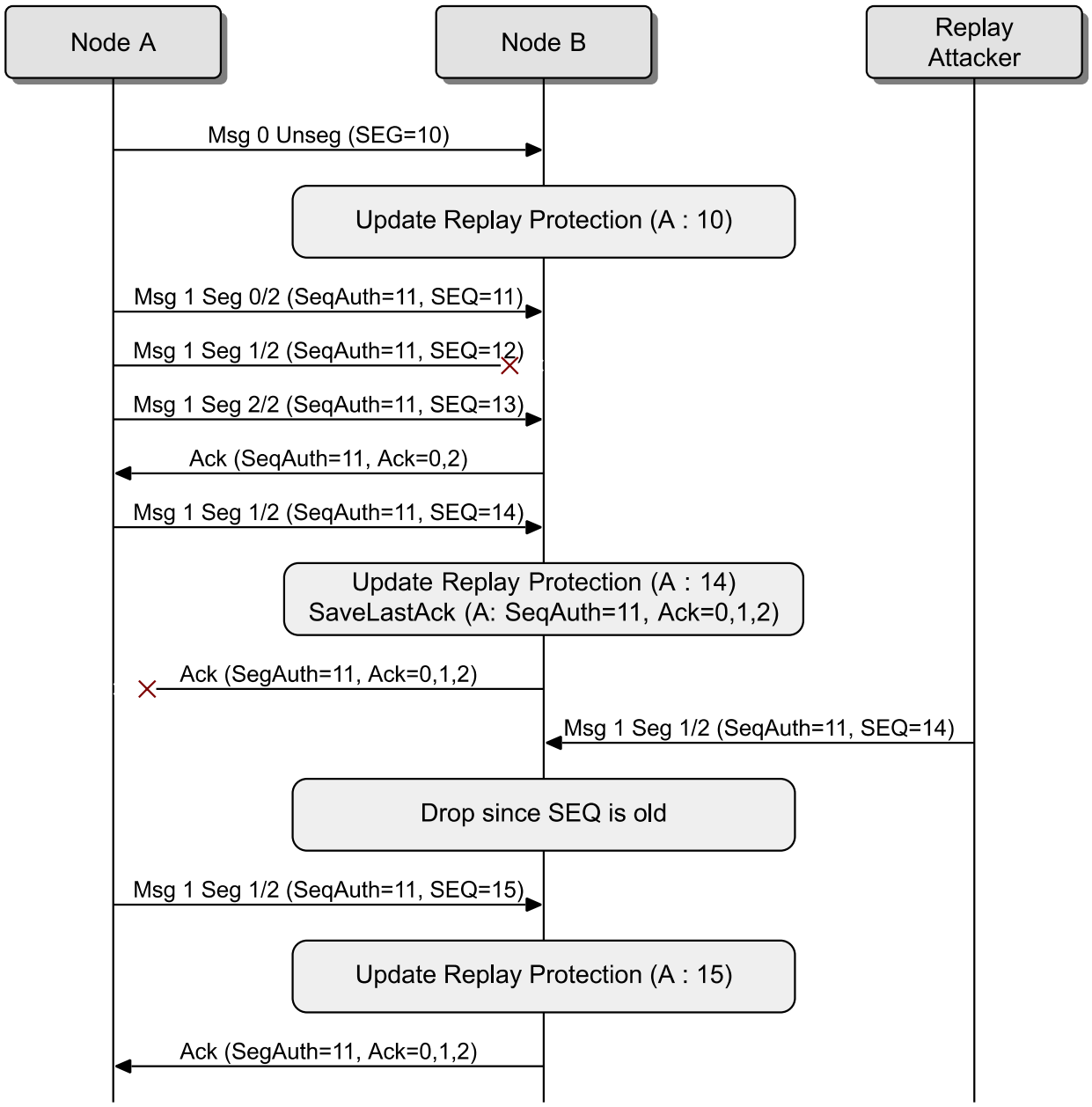

Figure 3.43: Example of updating replay protection list for segmented messages

3.9 Mesh beacons

Mesh beacons are packets advertised periodically by nodes and unprovisioned devices.

Mesh beacons are contained in a «Mesh Beacon» AD Type. The first octet of the Mesh Beacon advertising data payload (Beacon Type field) determines the type of beacon. Mesh beacons are forwarded to other bearers using the Proxy protocol (see Section 6).

The format of the Mesh Beacon AD Type is shown in Figure 3.44.