IEEE P802.21  
Media Independent Handover Services

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clause 9.4 and Clause 9.5 restructure proposal | | | | |
| Date: 2013-11-xx | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Lily Chen | NIST | 100 Bureau Dr. Gaithersburg, MD 20878 | +1 (301) 975-6974 | Lily.Chen at nist.gov |
| Antonio de la Oliva | UC3M |  |  | Aoliva at it.uc3m.es |

Abstract

This contribution proposes an outline to restructure clause 9.4 and 9.5 in IEEE P802.21d/D2.0 to resolve the comments received in ballot 7a.

**Background:**

Based on the comments received in ballot 7a, clause 9.4 and clause 9.5 in IEEE P802.21d/D2.0 need to be restructured. Here is a summary about the issues with the current structure.

1. Without introducing the tree structure for device keys, the content is hard to understand (see comment #107).
2. Subclause 9.4.1 is titled as protection mechanisms. However, 9.4.1 is more on some basic assumptions about the command center.
3. Subclauses 9.4.1 and 9.4.2 have content overlap about the command center. The descriptions are inconsistent (see comments #33, #97, etc.).
4. Subcluse 9.4.2.4 is misplaced. The content is needed for the description of the commend center in 9.4.2.1-9.4.2.3.
5. Subclause 9.4.3 and subclause 9.4.4 have the same title.
6. Clause 9.5 shall not be a stand alone clause.

Therefore, a new structure is proposed for clause 9.4 and 9.5. The basic idea is to separate the manipulation part and group command protection part. The new proposed clause 9.4 will address the group manipulation, while 9.5 will specify group command protection.

**New Structure Proposal**:

----------

1. MIH protocol protection

***NOTE—Editor: The following Clauses are not changed, they appear in order to be able to reference them later.***

* 1. Protection established through MIH (D)TLS
  2. Key establishment through an MIH service access authentication
     1. MIH service access authentication
     2. Key derivation and key hierarchy
  3. MIH message protection mechanisms for EAP-generated SAs

***Insert the following Clause:***

* 1. Group manipulation for multicast MIH messages

A multicast MIH message is an MIH message sent to a group of recipients. A multicast recipient can be a mobile node (MN) or a point of service (PoS). Each group is identified by an MIHF group ID. A group is dynamic in the sense that some of the group members may leave, while the new member may join. The group is managed through group manipulation commands. A series of group commands may follow a group manipulation command that defines a target group of MNs.

A series of group commands may follow a group manipulation command that defines a target group of MNs. A group command is issued, for instance, to instruct the group that the members should handover to a PoA or that they should update their configuration parameters. A payload of a group command can be protected (encrypted) using the SA derived from the MGK. The following two steps describe how group manipulation and command delivery are performed:

* Step 1: A Command center, which is an MIH PoS, issues a group manipulation command to instruct MNs to join or leave a group. A group manipulation command may also be used to update a group key stored at an MN. Group manipulation commands may be delivered to MNs through existing multicast channels. A group is associated with a multicast channel. The address used by this multicast channel can be provided by the group manipulation command itself. Note that an MN who is listening to the multicast channel may not be in the group.
* Step 2: A Command center issues to a group of MNs a group command (not a group manipulation command) to instruct the MNs in the group to take an action. The target group is designated by the MIHF Group ID field in the group command. A group command is delivered through the multicast channel associated with the MIHF Group ID. A group command may alternatively take two types of payload: Encrypted and non-encrypted. If a payload is encrypted, it is encrypted with a key derived from the current group key.
  + 1. Key distribution for multicast MIH message protection

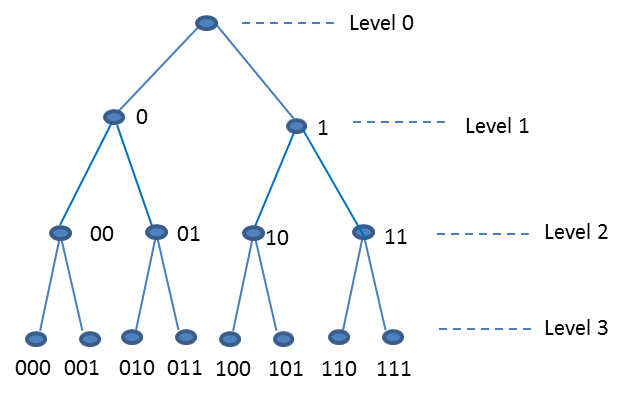
Multicast MIH messages are protected through group session keys. A group key can be a master group key (MGK) or a group session key derived from an MGK. An MGK is distributed through a group manipulation command. The command carries a special data field, called group key block (GKB), in which the encrypted MGK is included.

It is assumed that each potential recipient is provisioned with a set of device keys. Depending on the members in the group, an MGK is encrypted by a specific set of device keys so that each member in the group can decrypt one of the encrypted versions to obtain the MGK. A recipient belonging to the group can use one of its device keys to recover MGK, from which it can derive the group session keys used to protect the multicast MIH messages.

* + - 1. Device key assignment through a key tree

The device key assignment mechanism specified in this standard is based on a binary key structure, called a key tree. A key tree is a binary tree in depth d, where d is a system constant. The root of the key tree is called a level 0 node. At level k, there are 2k nodes, 0 ≤ k ≤ d. Each level k node can be represented as a k-bit string. The string is called the index of the node. For example, when d > 1, the level 2 nodes are represented by the indices 00, 01, 11, 10. Each node is assigned to a key, called a node key and identified by the Node Index. For example, the node keys assigned to level 2 nodes are denoted as k<00>, k<01>, k<10>, k<11>.

For a key tree in depth d, all the level d nodes are called leaf nodes. Each potential recipient (an MN or a PoS) is assigned to a leaf node. For a recipient assigned to a leaf node with index x0x1…xd-1, d device keys k<x0>, k<x0x1>, …, k<x0x1…xd-1> are provisioned. Figure x illustrates a key tree in depth 3. Table 3 lists the device key assignment for each recipient represented by the leaf Node Index.



1. —A key tree in depth 3
2. —Device key assignments for MNs through a depth-3 key tree

|  |  |
| --- | --- |
| Node | Device keys |
| 000 | k<0>, k<00>, k<000> |
| 001 | k<0>, k<00>, k<001> |
| 010 | k<0>, k<01>, k<010> |
| 011 | k<0>, k<01>, k<011> |
| 100 | k<1>, k<10>, k<100> |
| 101 | k<1>, k<10>, k<101> |
| 110 | k<1>, k<11>, k<110> |
| 111 | k<1>, k<11>, k<111> |

When determining the system constant d for a key tree, 2d must be no less than the number of all the recipients to be grouped in the system. In this standard, each leaf index is represented as an octet string. Therefore, the depth of a key tree is chosen as a multiple of 8.

For each leaf node, the Node Index is considered as the binary representative of an integer, called leaf number. For example, in the key tree in Figure 24, the leaf nodes with the Node Index 000 has leaf number 0, 001 has leaf number 1, 010 has leaf number 2, etc.

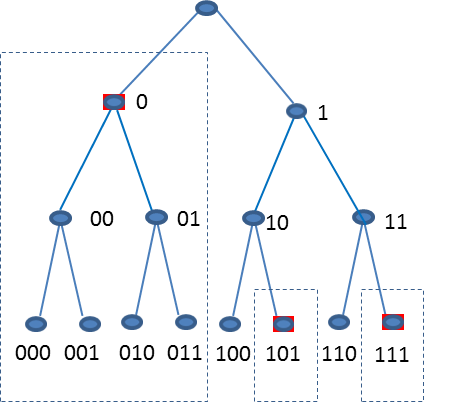
* + - 1. Complete subtree

A complete (k-depth) subtree is a subtree which has a group of 2k leaf nodes such that their indices have a common prefix of d-k bits. For the key tree in Figure 24, nodes represented with indices 000 and 001 form a depth-1 complete subtree, while nodes represented with indices 000, 001, 010, and 011 form a 2-depth complete subtree.

When the device keys are assigned through a key tree as introduced in 2.4.1.1, all the recipients corresponding to the leaf nodes in a complete subtree share a common node key (as one of the device keys), determined by the d-k bit prefix. In the example above, the recipients corresponding to the leaf nodes represented with indices 001 and 001 share node key k<00>. Therefore, a subset of members in a group corresponds to the leaf nodes in a complete subtree that can decrypt the MGK encrypted by the common shared key.

In order to distribute a MGK to a group of recipients, the first step is to sort the corresponding leaf nodes to non-overlap complete subtrees so that each leaf node belongs to one complete subtree. Note that a single leaf node can be considered as a depth-0 complete subtree. In this case, the MGK will be encrypted by the minimum number of keys for the group.

In the example illustrated in Figure 25, the group represented by the leaf nodes 000, 001, 010, 011, 101, 110 can be sorted as one depth-2 complete subtree 000, 001, 010, 011 and two depth-0 complete subtrees 101 and 110 as illustrated in Figure 25. A complete subtree can be identified by its root node. The aforementioned complete subtrees are identified by node 0, 101, and 111, respectively. For the MNs in such a group, the MGK shall be encrypted by k<0>, k<101> and k<110>.



1. —Three complete subtrees for the group with nodes 000, 001, 010, 011, 101, 111

The algorithm of sorting a group of leaf nodes to non-overlap complete subtrees is introduced in the following:

def CreateCompleteSubtree(I, T, R):

# Input I: List of indices of leaf nodes to be included in the group

# Input T: The entire tree that covers all leaf nodes

# Input R: Root node of the entire tree

# Output S: Complete Subtree for the group.

S=[]

def check(n):

# Input n: subtree root node

# Output 0, 1

# 0 : Some node in the subtree is a non-member of the group.

# 1 : All nodes in the subtree are members of the group.

if n.left==None and n.right==None: # n is leaf

if n.index.val in I:

S.append(n)

return 1

return 0

# n is non-leaf

lval=check(n.left)

rval=check(n.right)

if lval\*rval>0:

S.remove(n.left)

S.remove(n.right)

S.append(n)

return 1

return 0

check(R)

return S

* + 1. GKB generation by the complete subtree method

A GKB contains a complete subtree part and a group key data part. A group key data part appears when a GKB is used to deliver a group key.

A complete subtree part identifies all the complete subtrees covered by the group such that each member represented by a leaf node in the group belongs to one and only one complete subtree. Each complete subtree is identified by its root node. Each root node has a Node Index. A Node Index is a pair of binary strings (Node Depth, Node Index Value).The first binary string, Node Depth, indicates the binary length of the index of the root node. The second binary string, Node Index Value, is the index of the root node.

The Node Depth is represented by an octet. The corresponding integer L is the binary length of the index of the root node representing the complete subtree. The Node Index Value is represented by ⎡L/8⎤ octets, where ⎡x⎤ is the ceiling function, which takes the minimum integer which is larger than or equal to x. For example, ⎡1.2⎤ = 2. With the ceiling function, to demonstrate, if the value of a Node Depth, L, is not greater than 8, the size of the binary string used to represent the Node Index Value is 1 octet (= 8 bits) and if the value of a Node Depth, L, is greater than 8 and not greater than 16, the size of the binary string used to represent the Node Index Value is 2 octets (= 16 bits). A Node Index Value is left aligned in the network byte order. A Node Index Value shall have a zero padding added to the right. An example of Node Index is (0x05, 0b10011000). This Node Index represents the node ‘10011’, which is the root node of the complete subtree in a key tree. The depth of the key tree is a system parameter and configured to the applications. If the key tree has depth 8, then the root node ‘10011’ identifies a depth-3 complete subtree. Another example of Node Index is (0x0e, 0b1100101000011100), which represents the node ‘11001010000111’. If it is in a depth-16 key tree, then the node identifies a depth-2 complete subtree.

A complete subtree part is a list of Node Indices. The Node Indices are ordered based on the following rule. Let (L1, I1) and (L2, I2) be two Node Indices. (L1, I1) appears in the front of (L2, I2), if and only if Int(d, I1) < Int(d, I2), where Int(d, Ih) is a function to convert binary index Ih to an d-bit long integer. It uses Ih as the binary values in the most significant bits of a d-bit integer. For example, if d=8, (0x05, 0b10011000) is converted to 27+24+23 = 152. When another Node Index is (0x08, 0b00011001) (with index 00011001 and convert to 24+23+1=25), then (0x05, 0b10011000) appears first. A group key data part of a GKB is a sequence of ciphertexts of the encrypted group key, where a group key is encrypted by Node Keys corresponding to all the root nodes of the complete subtrees covered by the group. There is a one-one correspondence between the complete subtree part and the group key data part in a GKB. The number of Node Indices, i.e. the number of complete subtrees, in the complete subtree part is equal to the number of ciphertexts of the encrypted group key in the group key data part. And, the n-th ciphertext of the group key is encrypted by the Node Key assigned to the root node designated by the n-th Node Index. The encryption is made using the AES-ECB mode.

In the example illustrated in Figure 25, the group with nodes 000, 001, 010, 011, 101, 111 is covered by three complete subtrees at root node 0, 101, and 111. Therefore, the complete subtree part in the GKB includes Node indices for node 0, 101, and 111. The key data part, the ciphertexts of MGK encrypted by k(0), k(101), and k(111) as indicated in Figure 26.



1. —GKB for the group with nodes 000, 001, 010, 011, 101 and 111

Notice that the example is for illustration purpose. The key tree has depth 3, not a multiple of 8. The Node Index Value is not illustrated with the binary string in Figure 26. In fact, Node Index (0) can be represented as (0x01, 0b000000000), Node Index (101) as (0x03, 0b10100000) and Node Index (111) as (0x03, 0b11100000).

A Command center has a component called GKB Generator. A GKB Generator receives all the device keys assigned to all the MNs associated to a group and a MGK. The MGK is a master group key for that group. MNs and the CC can generate a media independent group encryption key (MIGEK) and a media independent master group key confirmation key (MIGKCK) from MGK (see **Error! Reference source not found.**). The mechanism to provide all device keys to the GKB generator is out of the scope of this specification. This mechanism can just encompass the explicit provision of the device keys to the GKB Generator or the random seed used to derive them. On receiving those data, a GKB Generator outputs a GKB, or several GKBs.

[The content below shall be moved to 9.4.2.3 Fragmented GKB.]

* + - 1. Master group key wrapping

[This should use the content in 9.4.2.5.1 encapsulation. But new text is needed. The current text is just an example. It needs to be more general.]

**Error! Reference source not found.** provides an example of creation of GKBs at a GKB Generator. In the last example, two GKBs are given: GKB2-1 = {[8, 15], {11, 100}, {<k(11)>[Kb], <k(100)>[Kb]}} and GKB2-2 = {[0, 7], {0011, 0100}, {<k(0011)>[Kb], <k(0100)>[Kb]}}. The Subgroup Ranges [8, 15], [0, 7] are expressed by sequences of two octets: (0x08, 0x0f) and (0x00, 0x07). They make SubgroupRange TLVs. The Node Indices for the nodes labeled ‘11’ and ‘100’ are (0x02, 0b1100) and (0x03, 0b1000) respectively though the size 4 of the Node Indices is not compliant to this specification. Those Node Indices make a Complete Subtree TLV. Likewise, the Node Indices for the node labeled ‘0011’ and ‘0100’ are (0x04, 0b0011) and (0x04, 0b0100). Those Node Indices also make a Complete Subtree TLV. The Key Data TLVs for GKB2-1 and GKB2-2 are made of {<k(11)[Kb], <k(100)>[Kb]} and {<k(0011)>[Kb], <k(0100)>[Kb]}.

* + - 1. Master group key unwrapping

[This should use the content in 9.4.2.5.2 decapsulation.]

At first, the decapsulation procedure for a GKB with a group key data part is described as follows:

1. An MIHF finds a Node Index in the complete subtree part of the GKB and a Device Key in the device keys that the MIHF itself owns such that the Node Index and the Node Index of the Device Key are identical. Suppose that the Node Index thus found is the n-th Node Index in the complete subtree part. If the MIHF fails to find such Node Indices, the procedure shall terminate.

* If the procedure terminates here, it means that the MN does not belong to the group designated by the TargetIdentifier defined in **Error! Reference source not found.**. The MN shall leave the group if it currently belongs to the group.

1. Using the Node Key in the Device Key found in a), the MIHF decrypts the n-th encrypted group key in the group key data part. The result of the decryption is a group key KG.

* The group key KG is the group key for the group designated by the TargetIdentifier. The MN shall belong to the group.

1. Check the MAC in the VerifyGroupCode field using the group key KG. If it fails, the decapsulation procedure shall abort.

If a GroupKeyData TLV is absent in an MIH\_Net\_Group\_Manipulate indication message, the following procedure applies:

1. An MIHF tries to find a Node Index in the complete subtree part of the GKB and a Node Index in the device keys, which the MIHF itself owns such that the two Node Indices are identical.

* If the MIHF fails to find a matching pair, it means that the MN does not belong to the group designated by the TargetIdentifier defined in **Error! Reference source not found.**. The MN leaves the group if it currently belongs to the group.
* If the procedure succeeds to find a matching pair, it means that the MN belongs to the group designated by the TargetIdentifier defined in **Error! Reference source not found.**. The MN joins in the group if it does not currently belong to the group.

Note that an MN need not necessarily have device keys when GKBs without keys are used. Then, an MN is only required to have a sequence of Node Indices.

* + - 1. Fragmented GKB

Assume that each GKB has a length limit. When the length of a GKB exceeded the limit, fragment is needed. That is, instead of one GKB, multiple GKB fragments are transmitted. Notice that the number of ciphertexts in a GKB, which is the major factor in determining the length of a GKB, is determined by the number of complete subtrees the group covers. If the limit is represented by the number of complete subtrees and denoted as *GkbFragmentThreshold*, when a GKB contains more than *GkbFragmentThreshold* of Complete Subtrees, it is fragmented into multiple GKB fragments such that each GKB fragment contains at most *GkbFragmentThreshold* Complete Subtrees. When a GKB is transmitted in multiple fragments, in order to avoid that one potential recipient would need to look into every fragment to find out whether it is included as an eligible recipient for that GKB fragment, Subgroup Range is introduced to label each fragment. A Subgroup Range is defined by a pair of integers [a, b] such that a potential recipient with leaf number r will look into the fragment if a ≤ r ≤ b. Each GKB fragment is associated with exactly one Subgroup Range. For example, for a depth-3 key tree as illustrated in Figure 36, the Subgroup Range [0, 3] corresponds to the leaf nodes with indices 000, 001, 010, 011. The Subgroup Ranges for GKB fragments of the same GKB satisfy all of the following conditions:

1. Union of all Subgroup Ranges is equal to the group range defined as the range of the leaf indices of the group management tree. For example, for a depth-3 key tree as illustrated in Figure 36, the Subgroup Range can be [0, 3], [4, 5], [6, 7].
2. Intersection of any two Subgroup Ranges is empty.
3. Each complete subtree must be in one and only one Subgroup Range.

An algorithm by which Complete Subtrees and Subgroup Ranges that satisfy these conditions is defined as follows.

def CreateCompleteSubtreeFragments(I, T, R, M):

# Input I: List of indices of leaf nodes to be included in the group

# Input T: The entire tree that covers all leaf nodes]

# Input R: Root node of the entire tree

# Input M: Maximum number of subtrees in per fragment

# Output O: List of (S, minr, maxr):

# S: Subtrees covering the group.

# minr: Lower bound of Subgroup Range

# maxr: Upper bound of Subgroup Range

O=[]

S=[]

depth=int(math.log(len(T)+1,2)-1)

def rightmost\_leaf\_number(n):

# Input n: subtree root node

# Output y: rightmost leaf number under the subtree

h=n.index.len # hierarchy level of node n

x=int(n.index.val, 2) # node index in decimal

y=(x+1)\*(2\*\*(depth-h))-1

return y

def check(n):

# Input n: subtree root node

# Output 0, 1

# 0 : Some node in the subtree is a non-member of the group.

# 1 : All nodes in the subtree are members of the group.

global minr

rv=0

if n.left==None and n.right==None: # n is leaf

if n.index.val in I:

S.append(n)

return 1

return 0

# n is non-leaf

lval=check(n.left)

rval=check(n.right)

if lval\*rval>0:

S.remove(n.left)

S.remove(n.right)

S.append(n)

rv=1

elif lval+rval>0:

if len(S) > M: # one fragment is ready

maxr=rightmost\_leaf\_number(S[M-1])

O.append((S[0:M], minr, maxr))

S[0:M]=[] # Remove the appended subtrees

minr=maxr+1

rv=0

if n==R: # Root node. len(S)>0

maxr=2\*\*depth-1

O.append((S, minr, maxr))

return rv

check(R)

return(0)

[This should use the content in original 9.4.1. Subgroup range needs to be introduced. Then use the paragraph above the Table 26 and the table. The last column of Table 26 does not seem right. The expected behavior is not join or leave. It is to process the GKB or not.]

* + 1. Secure group manipulation procedures

[This should use the content in 9.4.2.]

Although there are detailed procedures of an MIH User at a Command center to prepare an MIH request for group manipulation, handover or configuration update depends on implementations of the User. An overview of the behaviors of an MIH User is given in **Error! Reference source not found.**, which also defines a series of actions to be performed by an MIHF which receives an indication message of group manipulation, handover or configuration update.

In case the MIH PDU is protected through GKB-generated MIH SA as specified in Clause **Error! Reference source not found.**, the MIHF of PoS generates a Signature TLV consisting of a SIGNATURE\_DATA and a CERT\_SERIAL\_NUMBER. The SIGNATURE\_DATA is created by signing an MIH\_Group\_Manipulate command or a group addressed command using a signing key corresponding with a verification key specified by CERT\_SERIAL\_NUMBER.

In case the MIH PDU received contains a Signature TLV, then the MIHF of MN retrieves CERT\_SERIAL\_NUMBER and SIGNATURE\_DATA from the Signature TLV. Then, the MIHF verifies the SIGNATURE\_DATA using a verification key specified by the CERT\_SERIAL\_NUMBER.

Figure 27 illustrates group manipulation command distribution initiated by a Command center via a multicast channel. The MIH User of the Command center generates an MIH\_Net\_Group\_Manipulate.request, described in **Error! Reference source not found.**, and then it passes the request to the MIHF of the Command center. Upon receiving the request, the MIHF generates MIH\_Net\_Group\_Manipulate indication (Note that the decision on sending an indication message or a request message depends on the ResponseFlag parameter of the MIH\_Net\_Group\_Manipulate.request primitive), described in **Error! Reference source not found.**, and sends it to the MNs via multicast mechanisms. When an MN receives the MIH\_Net\_Group\_Manipulate indication message, the MIHF of the MN processes the message. After processing the message, the MIHF sends MIH\_Group\_Manipulate.indication to the MIH User of the MN.



1. —Example of group manipulation distribution using multicast mechanisms

**9.4.3.1 Procedures for group manipulation command sender (GMCS)**

[Here we assume that group manipulation command sender is a PoS, where the MIH user is the command center.]

**9.4.3.1.1 MIH user of a GMCS**

[This can use the content in 9.4.2.1. Here MIH user of a GMCS is a command center.]

Note that this section is informative.

Required components in an MIH User of a PoS (Command Center) relevant to group manipulation and group commands are listed as follows:

* A GKB Generator.
* All the MIHF IDs and all the Device Keys each of which is uniquely associated with one of the MIHF IDs. Each Device Key accompanies a Leaf Number.
* A *Group Management Database* which stores a *Groups Table*, a *Members Table* and a *Memberships Table*. The *Groups Table* stores the existing groups. This table is formed at least for two columns: MIHF Group ID and MGK. A row of the table indicates that a group designated by the MIHF Group ID exists and has the corresponding MGK for the master group key. The *Members Table* stores the group members (MNs): It at least has the following three columns: MIHF ID, Device Key and Leaf Number. A row of the table indicates that an MN designated by the MIHF ID exists and has the specific Device Key with the corresponding Leaf Number. And, the *Memberships Table* stores associations between the groups and the members: It at least has the following two columns: MIHF Group ID and MIHF ID. A row of the table indicates that the MN designated by the MIHF ID belongs to the group designated by the MIHF Group ID.

An MIH User generates MIH\_Net\_Group\_Manipulate.request described in **Error! Reference source not found.** as follows:

1. Define a group to manipulate:
   1. If it is a new group, choose a TargetGroupIdentifier by consulting with the *Group Management Database*. A TargetGroupIdentifier should be an MIHF Group ID that is not currently in use as an MIHF Group ID for an existing group. Then, decide group members, i.e. MNs, which belong to the group. Choose an MGK for the group. Add a row to the *Groups Table*: The row contains the chosen TargetGroupIdentifier and the MGK. Finally, add rows to the *Memberships Table* for all the group members: Each of the rows contains the TargetGroupIdentifier and the MIHF ID of a group member.
   2. For an already existing group, obtain all the group members in the group. As necessary, add new group members to them and remove group members from them so that the members of the group are updated. Choose an MGK for the group. It may be equal to the current MGK. If a new MGK is chosen, update with the new MKG each row in the *Groups Table* containing the TargetGroupIdentifier of the existing group. Add the rows to the *Memberships Table* where the rows have the TargetGroupIdentifier and the MIHF IDs of new group members. Remove the rows from the *Memberships Table* where the rows have the TargetGroupIdentifier and the MIHF IDs of group members, which are removed from the group.
2. Send to the GKB Generator all the Device Keys and the associated Leaf Numbers of the group members determined in a) and the MGK. Then, the MIH User receives from the GKB generator a GKB or a set of GKBs: A GKB contains a CompleteSubtree field, a GroupKeyData field and optionally a SubgroupRange field. A SubgroupRange is a pair of Leaf Numbers and defines a range of Leaf Numbers. A simple example that shows how to make those fields is given in Figure 46. A GKB contains a SubgroupRange field if it is one of fragmented GKBs. Note that one MIH\_Net\_Group\_Manipulate.request contains one and only one GKB. Plural GKBs result in plural requests. Optionally a VerifyGroupCode is also generated.



1. — Flow diagram of the generation process of the GKB parameters
2. (Optional) Construct the UserSpecificData field.
3. Choose a DestinationIdentifier. A DestinationIdentifier is an MIHF Group ID, which represents an existing group. The SubgroupRange indicates the MNs that are the distribution targets of the GKB. If an MN is in the range, it should receive the GKB or fragmented GKB. At least, an MIHF Broadcast Identifier is assumed to exist. Other initial groups may exist though they are out of the scope of this specification.
4. Generate an MIH\_Net\_Group\_Manipulate.request from the DestinationIdentifier, the TargetGroupIdentifier, the SubgroupRange (an option), the VerifyGroupCode (an option), the UserSpecificData (an option), the CompleteSubtree and the GroupKeyData (an option). Set the GroupKeyUpdateFlag if the MGK of the group designated by the TargetGroupIdentifier should be updated. Send it to the local MIHF.

Figure 47, shows a flow diagram summarizing the steps performed by the MIH User on a PoS, described in this Clause.



1. — Summary of steps performed by PoS MIH User

**9.4.3.1.2 MIHF of a GMCS**

[This can use the content in 9.4.2.2.]

Required components relevant to group manipulation and group commands are listed as follows:

* A signing key. The key is for creation of a signature of the Command center.
* A Device Key to retrieve a group key from a GKB which is received from the local MIH User.
* A *Multicast Address Database* which stores a multicast addresses table which has the following four columns: MIHF\_ID, Multicast Address, MGK and SAID. The Multicast Address on a row is associated with the group designated by the MIHF Group ID recorded on the same row. Additionally, the Multicast Address may accompany an attribute which indicates if it is defined at Layer 2 or 3 of the protocol stack. The MGK is the one derived from the latest GKB targetted to the group of the MIHF Group ID. The SAID column stores the SAID for the MGK. A *Multicast Address Database* may also have a *SAIDs Table* which has the following two columns at least: MIHF Group ID and SAID. A *SAIDs Table* stores all the SAIDs which have ever been assigned to the group of the MIHF Group ID. A *SAIDs Table* is used to check if a generated SAID is unique up to the group. If a SAID is always chosen in a unique way (e.g., monotoneously increasing), the *SAIDs Table* is not necessary.

It is assumed that the MIHF is able to obtain in some way a multicast address associated with a Group MIHF ID. The multicast address may be contained in the MIH\_Net\_Group\_Manipulate.request received from the MIH User. In this case, if the TargetGroupIdentifier in the received request is not registered in the database, obtain the multicast address associated with the TargetGroupIdentifier and update the database with the DestinationIdentifier and the associated multicast address. The MIHF of the Command center receives an MIH\_Net\_Group\_Manipulate.request, which is generated by the MIH User, the MIHF generates and sends an MIH\_Net\_Group\_Manipulate indication message to a multicast group. Note that this behavior depends on the ResponseFlag parameter. When “ResponseFlag=1”, the MIHF will generate MIH\_Net\_Group\_Manipulate request message. When “ResponseFlag=0”, the MIHF will generate MIH\_Net\_Group\_Manipulate indication message.

In the following we detail the steps performed to generate the message, herein we assume “ResponseFlag=0”:

1. Generate a Source MIHF ID TLV using its own MIHF ID.
2. Generate a Destination MIHF ID TLV from the DestinationIdentifier in the received MIH\_Group\_Manipulate.request.
3. Generate a Group Identifier TLV from the TargetIdentifier in the received MIH\_Group\_Manipulate.request.
4. Generate, as needed, a Multicast Address TLV from the multicast address corresponding to the TargetIdentifier in the received MIH\_Net\_Group\_Manipulate.request. The Multicast Address Database can serve for the purpose of finding the multicast address.
5. (Optional) Generate a SubgroupRange TLV from the SubgroupRange in the received MIH\_Net\_Group\_Manipulate.request.
6. (Optional) Generate a Verify Group Code TLV from the VerifyGroupCode in the received MIH\_Net\_Group\_Manipulate.request.
7. (Optional) Generate an Aux Data TLV from the UserSpecificData in the received MIH\_Net\_Group\_Manipulate.request.
8. Generate a Complete Subtree TLV from the CompleteSubtree in the received MIH\_Net\_Group\_Manipulate.request.
9. Generate a Group Key Data TLV from the GroupKeyData in the received MIH\_Net\_Group\_Manipulate.request.
10. Process the GKB (the Complete Subtree TLV and the Group Key Data TLV) using the Device Key assigned to the MIHF, and obtain the MGK. If the MIHF fails to obtain the master group key, the MIHF shall cancel the rest of the process.
11. Ask the *Multicast Address Database* and obtain the current MGK and the current SAID for the TargetGroupIdentifier. If the obtained MGK is equal to the MGK derived in j), do nothing here. Otherwise, generate a new SAID which is unique up to the group, and update the MGK and the SAID on the row for the TargetGroupIdentifier in the *Multicast Addresses Table*.
12. Generate a Signature TLV as shown in **Error! Reference source not found.** using the signing key of the Command center.
13. Generate an MIH\_Net\_Group\_Manipulate indication using the preceding TLVs. If necessary, its Service Specific TLVs are so encrypted that they make a Security TLV. Send the MIH\_Net\_Group\_Manipulate indication message to the multicast address corresponding to the DestinationIdentifier.

Figure 48, shows a flow diagram summarizing the steps performed by the MIHF at a PoS, described in this Clause.



1. —Summary of steps performed by PoS MIHF

**9.4.3.2 Procedures for group manipulation command recipients (GMCR)**

[Here a recipient can be an MN or a PoS. Only the MIHF is involved. ]

Required components relevant to group manipulation and group commands are listed as follows:

* A Device Key.
* A certificate of a Command Center which contains a verification key. The verification key is for verification of a signature made by the Command Center.
* A *Group Database* which stores a groups table, which has the following three columns at least: MIHF Group ID, MGK and Multicast Address. A row of the table specifies that this MN belongs to the group designated by the MIHF Group ID. The group has the MGK as the master group key and is associated with the Multicast Address. The Multicast Address may have an attribute which indicates if it defined at Layer 2 or 3 of the protocol stack.

When a client MN receives a group manipulation command, i.e., an MIH\_Net\_Group\_Manipulate indication/request message, issued by a Command center, the MIHF of the MN processes the command. Suppose at first that the GKB in the group manipulation command has a group key data element:

1. The MIHF obtains a Source Identifier from the Source MIHF ID TLV.
2. The MIHF verifies the Signature TLV using the verification key corresponding to the obtained SourceIdentifier. If the verification fails, the MIHF shall cancel the following steps and stop processing the command.
3. The MIHF checks the DestinationIdentifier in the Destination MIHF ID TLV. If the DestinationIdentifier does not match one of the following MIHF IDs, the MIHF shall cancel the following steps and stop processing the command: (i) An MIHF Group ID corresponding to a broadcast address, (ii) an MIHF Group ID which is registered with a multicast address in the Group Database, or (iii) the MN's own MIHF ID.
4. Decrypt the payload if it is encrypted, i.e., if it is a Security TLV. The decryption key is the one associated with the DestinationIdentifier in the Group Database.
   1. In case an MN cannot decrypt the Security TLV, the message will be silently discarded.
5. If a SubgroupRange TLV exists in the indication, the MIHF obtains a SubgroupRange and checks whether its own Leaf Number is contained in the SubgroupRange or not. If it is not, the MIHF shall cancel the following steps and stop processing.
6. The MIHF obtains the TargetIdentifier in the Group Identifier TLV.
7. A GKB is composed of the Complete Subtree TLV, the Group Key Data TLV and optionally the Verify Group Code TLV. The MIHF processes the Complete Subtree TLV and the Group Key Data TLV as described in **Error! Reference source not found.**. If a Verify Group Code TLV exists, the MIHF verifies the group key derived from the GKB. If an MGK is obtained (and verified), go to the next step. Otherwise, go to Step i). In case the Verify Group Code TLV is not present in the GKB, if an MGK is obtained, go to the next step without verification of the obtained MKG. Otherwise, go to Step i). The processing of all these parameters is shown in Figure 49.



1. —MGK generation and verification process
2. The MIHF checks whether the TargetIdentifier obtained in Step f) has already been registered or not in the Group Database. If it has been, go to Step j) [Stay]. Otherwise, go to Step k) [Join].
3. The MIHF checks whether the TargetIdentifier has already been registered or not in the Group Database. If it has been, go to Step m) [Leave]. Otherwise, go to Step j) [Stay].
4. [Stay] The MIHF throws an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field of the indication shall be “Unchanged successful” (5). The procedure of command processing terminates.
5. [Join] The MIHF obtains a multicast address associated with the TargetIdentifier and starts listening to it. The messages come through the multicast channel may be encrypted with the group key obtained in Step g). The multicast address may be obtained from a server (Note that this operation is out of the scope of this specification). Or, the received indication may accompany it in the Multicast Address TLV. Save in the Group Database the TargetIdentifier, the associated multicast address and the group key obtained in Step f).
6. The MIHF issues an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field must be “Join operation successful” (0). The procedure of command processing terminates.
7. [Leave] The MIHF finds the multicast address recorded on the same row as the TargetIdentifier obtained in Step f) and the MIHF stops listening to it. The MIHF removes the row that has the TargetIdentifier.
8. The MIHF throws an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field must be “Leave operation successful” (3). The procedure of command processing terminates.

Then, suppose that the GKB in the group manipulation command has no group key data part:

1. The MIHF obtains a Source Identifier from the Source MIHF ID TLV.
2. The MIHF verifies the Signature TLV using the verification key corresponding to the obtained SourceIdentifier. If the verification fails, the MIHF shall cancel the following steps and stop processing the command.
3. The MIHF checks the DestinationIdentifier in the Destination MIHF ID TLV. If the DestinationIdentifier does not match one of the following MIHF IDs, the MIHF shall cancel the following steps and stop processing the command: (i) An MIHF Group ID corresponding to a broadcast address, (ii) an MIHF Group ID which is registered with a multicast address in the *Group Database*, or (iii) the MN's own MIHF ID.
4. Decrypt the payload if it is encrypted, i.e., if it is a Security TLV. The decryption key is the one associated with the DestinationIdentifier in the *Group Database*.
5. If a SubgroupRange TLV exists in the indication, the MIHF obtains a SubgroupRange and check whether its own Leaf Number is contained in the SubgroupRange or not. If it is not, the MIHF shall cancel the following steps and stop processing.
6. The MIHF obtains a TargetIdentifier in the Group Identifier TLV.
7. A GKB is composed of the Complete Subtree TLV. The MIHF processes the Complete Subtree TLV as described in **Error! Reference source not found.**. If the MIHF succeeds to find a matching pair of GKB Indices, go to the next step. Otherwise, go to Step i).
8. The MIHF checks whether the TargetIdentifier obtained in Step f) has already been registered or not in the Group Database. If it has been, go to the Step j) [Stay]. Otherwise, go to Step k) [Join].
9. The MIHF checks whether the TargetIdentifier obtained in Step f) has already been registered or not in the *Group Database*. If it has been, go to Step m) [Leave]. Otherwise, go to Step j) [Stay].
10. [Stay] The MIHF issues an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field of the indication must be “Unchanged successful” (5). The process terminates.
11. [Join] The MIHF obtains a multicast address associated with the TargetIdentifier and starts listening to it. The multicast address may be obtained from a server. Or, the received indication may accompany it in the Multicast Address TLV. Save in the Group Database the TargetIdentifier, the associated multicast address.
12. The MIHF issues an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field must be “Join operation successful” (0). The procedure of command processing terminates.
13. [Leave] The MIHF finds the multicast address recorded on the same row as the TargetIdentifier obtained in f) and the MIHF stops listening to it. The MIHF discards the row that has the TargetIdentifier.
14. The MIHF issues an MIH\_Net\_Group\_Manipulate.indication described in **Error! Reference source not found.** to the MIH User. The GroupStatus field must be “Leave operation successful” (3). The procedure of command processing terminates.

Figure 50 summarizes the steps followed by the MIHF on the MN upon reception of an MIH\_Net\_Group\_Manipulation.indication.



1. —Summary of steps performed by the MN MIHF

Subclause **Error! Reference source not found.** introduces a mechanism enabling the MN to trigger the Join/Leave operations controlled by the Command center. In order to do so, the MIH User located at the MN notifies the Command center of its desire to Join or Leave a group through the use of the MIH\_MN\_Group\_Manipulate primitive. The Command center, upon receiving the associated request message, performs the same process as defined in this Clause, for the use of the MIH\_Net\_Group\_Manipulate, although in this case, the group to be manipulated is provided by the MN. The resulting GKB parameters are returned to the MN in the MIH\_MN\_Group\_Manipulate response message.

**9.5 Multicast MIH message protection**

**9.5.1 Group session key derivation**

[This can use the content in 9.4.3.]

Fig943a

1. —Key derivation example

When an MN successfully recovers a GKB, it obtains a master group key (MGK). The following three keys are derived from MGK:

* Group key confirmation key (MIGKCK) used as a key confirmation key to confirm that the correct MGK is obtained through a Message Authentication Code (MAC);
* Group manipulation encryption key (MIGMEK) used to protect a group manipulation command;
* Group encryption key (MIGEK) used to protect the MIH Group command.”

The deriving key is specified by the different multicast ciphersuites described in 2.1.2. For the key derivation, the following notations and parameters are used.

* *K*: key derivation key. It is truncated from a master group key (MGK). The length of *K* is determined by the pseudorandom function (PRF) used for key derivation. If HMAC-SHA-1 or HMAC-SHA-256 is used as a PRF, then the full MGK is used as key derivation key, *K*. If CMAC-AES is used as a PRF, then the first 128 bits of MGK are used as derivation key, *K*.
* *L*: The binary length of derived keying material MIGSK. *L* is determined by selected multicast ciphersuites described in 2.1.2.
* *h*: The output binary length of PRF used in the key derivation. That is, *h* is the length of the block of the keying material derived by one PRF execution. Specifically, for HMAC-SHA-1, *h* = 160 bits; for HMAC-256, *h* = 256 bits; for CMAC-AES, *h* = 128 bits.
* *n*: The number of iterations of PRF in order to generate *L*-bits keying material.
* *c*: The multicast ciphersuite code is a one octet string specified for each ciphersuite. The code is defined in 2.1.2.
* *v*: The length of the binary representation of the counter and the length of keying material L. The default value for *v* is 32.
* “MIGSK”: 0x4D4947534B, ASCII code in hex for string “MIGSK.”
* [a]2: Binary representation of integer *a* with a given length.

For given PRF, the key derivation for MIGSK can be described in the following procedures:

**Fixed input values**: h and v.

**Input**: *K*, *L*, and multicast ciphersuite code.

**Process**:

1. If *n* > 2v-1, then indicate an error and stop.
2. Result(0) := empty string.
3. For i = 1 to *n*, do
   1. *K*(i) := PRF(*K*, “MIGSK” || [i]2 || *c* || [*L*]2).
   2. Result(i) = Result(i-1) || *K*(i).
4. Return Result(n) and MIGSK is the leftmost *L* bits of Result(n).

**Output**: MIGSK.

The MIGSK is parsed in such a way that

MIGSK = MIGKCK || MIGEK.

With the above procedure, a key hierarchy is derived as shown in Figure 51.

**9.5.2 Multicast message encrypton**

[This can use the content in 9.4.3.] [AO]:I have used 9.4.4?

In order to issue an MIH\_Configuration\_Update indication message, the MIH User of the Command center generates an MIH\_Configuration\_Update.request described in **Error! Reference source not found.** and delivers it to the local MIHF. Upon receiving the request, the MIHF of the Command center behaves as follows:

1. The MIHF generates a Source MIHF ID TLV based on its own MIHF ID.
2. The MIHF generates a Destination MIHF ID TLV based on the DestinationIdentifier in the received request.
3. The MIHF generates a Configuration Data TLV from the ConfigurationData in the received request.
4. Consulting with the Multicast Address Database, the MIHF finds the multicast address associated with the DestinationIdentifer in the received request.
5. The MIHF generates an MIH\_Configuration\_Update indication message described in **Error! Reference source not found.**, and it sends it to the multicast address found in Step d).
   1. The Configuration Data TLV in the MIH\_Configuration\_Update indication message may be encrypted to make a Security TLV if necessary in the scheme described in **Error! Reference source not found.**.

When an MIHF of an MN receives an MIH\_Configuration\_Update indication message, it issues an MIH\_Configuration\_Update.indication described in **Error! Reference source not found.** to its MIH User, following the next steps:

1. The Destination Identifier is retrieved from the Destination MIHF ID TLV. The MIHF checks if the Destination Identifier is registered in the Group Database or not. If it is not, the message is not for the MN. Thus, it cancels the following steps and stops processing.
2. The Source Identifier is retrieved from the Source MIHF ID TLV.
3. The MIHF verifies the Signature TLV using the verification key corresponding with the preceding Source Identifier. If the verification fails, it cancels the following steps and abort.
4. The ConfigurationData is retrieved from the Configuration Data TLV. If it is encrypted, The MIHF decrypts the Security TLV with the group key associated with the Destination Identifier in the Group Database.
5. With this information, the MIHF generates an MIH\_Configuration\_Update.indication as described in **Error! Reference source not found.**.

Fig943b

1. —Example of configuration update distribution using multicast mechanisms
2. In order to fill the SourceIdentifer, TargetIdentifier and ConfigurationData fields of the MIH\_Configuration\_Update.indication primitive, the MIHF copies the Source Identifier, the Destination Identifier and the Configuration Data respectively of the MIH\_Configuration\_Update indication message.
3. Finally, the MIHF issues the MIH\_Configuration\_Update.indication created in f) to its MIH User.

**9.5.3 Signature and credential management**

[This can use the content in 9.4.5.]

In order to enable signing functionality, the message source requests credentials for public key using an out-of-band mechanism that is not specified in this document. The message source provides the credentials to destination devices. Message signing procedure, signature verification procedure and certificate management procedure are described in 2.1.1.1, 2.1.1.2 and 2.1.1.3, respectively.

In this specification, Elliptic Curve Digital Signature Algorithm (ECDSA) specified in in IEEE 802.1AR-2009, Secure Device Identity, by reference to NIST FIPS 186-4 and ANSI X9.62-2005 is used as the multicast signature scheme. In particular, NIST recommended elliptic curve P-256 and hash function SHA-256, specified in FIPS 180-4, are used to generate signatures. These algorithm identifiers are defined in Clause 2.1.1.4.

* + - 1. Multicast Message Signatures

Multicast Messages are signed with the message source using a private key of the message source. Integrity and proof of origin of a multicast message is verified by verifying the message signature with the public key of a message source.

On receipt of signed multicast message there is an optional response indicating the validity of signature. Message source requests credentials for key updates. Message source provides updates of credentials to destination devices (with overlap period).

The message content is signed using elliptical curve cryptography.

* + - 1. Signature Verification

The signature is verified using the message source signature verification key. The endpoints might have more than one key used for signature verification. This is to allow for key updates to happen in an efficient manner for large systems.

The message source will identify which key is to be used for the multicast message so that verification will utilize the correct key for signature verification.

* + - 1. Certificate Management

A root of trust will exist for the multicast nodes. The root of trust is envisioned to be a certificate authority. X.509 format certificates will be utilized. The root of trust will establish the binding between the identity of the message source and the public/private key pair used for signature generation and verification.

The certificate will include the identity of the certificate authority, the identity of the message source, the public key in use and the expiration date of the certificate and the certificate authority’s signature. For an endpoint (an MN or PoS) to trust the certificate it must have the certificate authority public key.

The initial certificates for multicast signature verification are distributed to multicast destinations as part of the provisioning process to the multi-node network. The certificates will include the certificate authority certificate used to verify the initial and updated certificates.

There will also be one or more certificates that are bound to the identity of the multicast source.

As part of the key update or revocation process, a new certificate will be provided to multicast destinations using the multicast mechanism. There needs to be a mechanism for multicast destinations to acknowledge the receipt of the multicast message.

When there is a suspicion that a certificate is compromised, a mechanism will be provided to revoke the certificate from service. This mechanism will utilize the multicast messaging mechanism. Multicast destinations will need to provide a reply that indicates they have successfully revoked the certificate.

* + - 1. Algorithm identifiers

The ECDSA signature method is defined in IEEE 802.1AR-2009, Secure Device Identity, by reference to NIST FIPS 186-4 and ANSI X9.62-2005.

If implementing ECDSA P-256, the SHA-256 message digest algorithm and the P-256 elliptic curve as defined in FIPS 186-4 Annex D, D.1.2.3, shall be used.

The signature algorithm shall be ecdsa-with-SHA256 as specified in IEEE 802.1AR by reference to RFC 5008. The object identifier is:

ecdsa-with-SHA256 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-sha2(3) 2 }

When the ecdsa-with-SHA256 algorithm identifier appears in the algorithm field as an AlgorithmIdentifier, the encoding shall omit the parameters field. That is, the AlgorithmIdentifier shall be a SEQUENCE of one component, the object identifier ecdsa-with-SHA256.

* + 1. Multicast Ciphersuites

The ciphersuites used for securing multicast MIH message are defined in Table 27.

1. —Multicast Ciphersuites

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Encryption Algorithm | Digital Signature Algorithm | MAC Algorithm for Verify Group Code |
| 10010101 | AES\_CCM-128 | ECDSA-256 | AES\_CMAC-128 |
| 10010100 | AES\_CCM-128 | ECDSA-256 | NULL |
| 10000100 | NULL | ECDSA-256 | NULL |

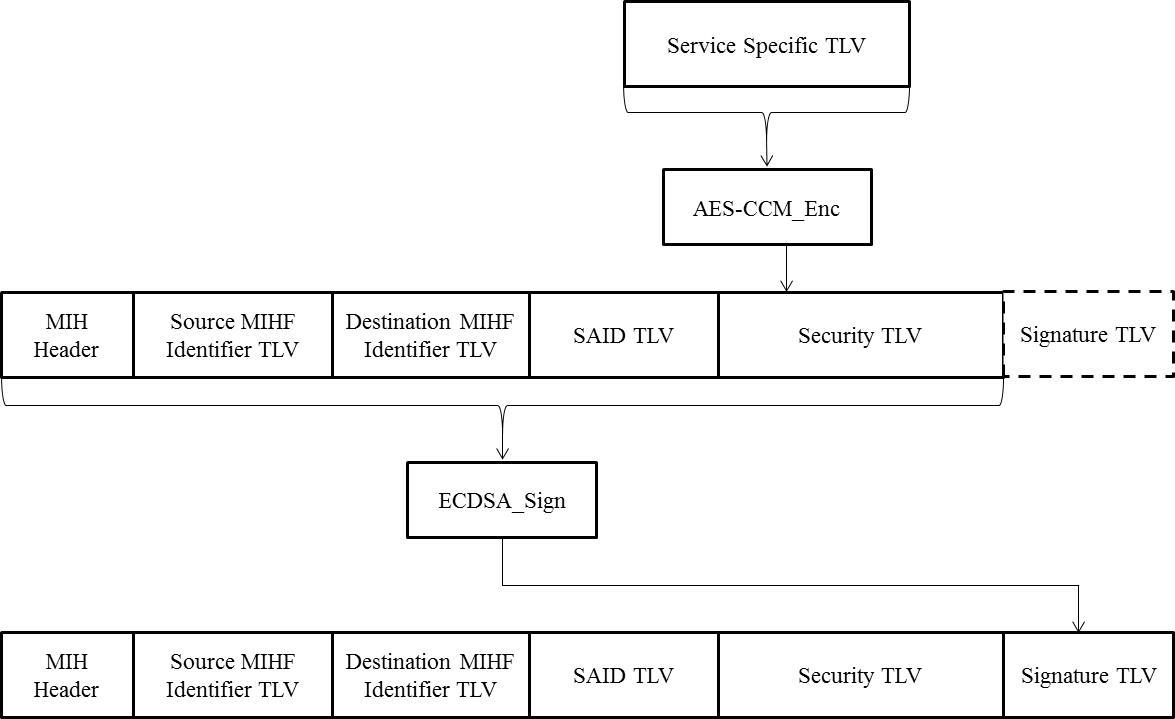
In Table 27, AES-CCM is an AES mode of operations specified in NIST SP 800-38C. AES-CCM provides confidentiality and data integrity.

In Table 27, ECDSA-256 uses curve P-256 and hash function SHA-256.

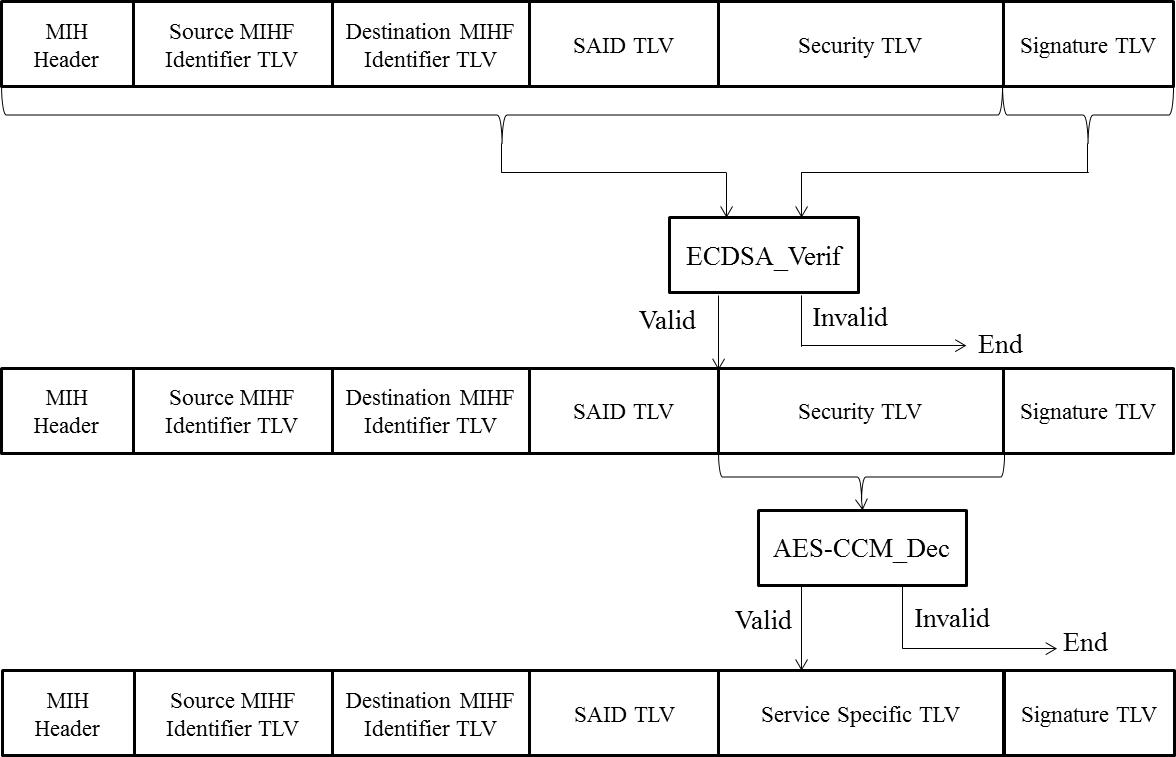
Notice that AES-CCM uses the group key, either MIGMEK or MIGEK. It can provide data integrity but not unique data origin authentication because the symmetric key is shared among a group of MNs. The data origin authentication is provided through ECDSA.

The support of code “10000100” is mandatory and all entities supporting this specification shall implement it.

The data protection procedure is illustrated in and Figure 53 and Figure 54.



1. —Encapsulation



1. —Decapsulation

**9.5.4 Common procedures**

[This can use the content in 9.5. The question is: common for what?]

**9.5.4.1 Sending**

When a PoS issues an MIH Service Specific TLV, the MIHF of the PoA generates a signature of the TLV using the signing key of the PoS and creates a Signature TLV from the generated signature.

**9.5.4.2 Receiving**

When an MN receives an MIH service specific TLV, the MIHF of the MN behaves as follows:

1. The MIHF verifies the signature in the Signature TLV using the verification key corresponding to the Source Identifier extracted from the received Source MIHF ID TLV. If the verification fails, it cancels the following steps and stops processing.
2. The Destination Identifier is extracted from the received Destination MIHF ID TLV. The MIHF checks if the Destination Identifier is registered as an MIH Group ID in the Group Database. If it is not, it cancels the following steps and stops processing.
3. If a Security TLV is found in the MIH Specific TLV, the MIHF decrypts the Security TLV using the MIGMEK derived from the MKG. The MGK is the group key corresponding to the Destination Identifier extracted in the previous step. The group key is found in the Group Database.