

---

**IEEE P802.15**  
**Wireless Personal Area Networks**

---

Project	IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)		
Title	<b>LB97 resolutions - Kivinen</b>		
Date Submitted	11 July, 2014		
Source	Tero Kivinen INSIDE Secure Eerikinkatu 28 FI-00180 Helsinki Finland	Voice:	+358 20 500 7800
		Fax:	+358 20 500 7801
		E-mail:	kivinen@iki.fi
Re:	LB97 resolutions		
Abstract	LB 97 resolutions to CIDs 1028, 1150, 1026, 1145, 1146, 1148, 1147, 1153, 1101, 1151, 1105, 1104, 1154, R41 and R42		
Purpose	LB97 resolutions		
Notice	This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.		
Release	The contributor acknowledges and accepts that this contribution becomes the property of IEEE and may be made publicly available by P802.15.		

---

**CID 1028**

Robert Moskowitz	Verizon	313	9.2	16	Still need state machines for security flow	Provide state machine figures
---------------------	---------	-----	-----	----	---	-------------------------------

Outbound state machine done, see CID 1150 for it.

Inbound state machine done, see CIDs 1153, 1101, R41, R42 for it.

**CID 1150**

Tero Kivinen	INSIDE Secure	313	9.2.1	37	Swap steps c) and b). There is no point of checking length of the data expansion length if we are not protecting the frame. See CID 565.	Swap steps c) and b).
--------------	---------------	-----	-------	----	--	-----------------------

*New version of the Outbound state machine:*

The inputs to this procedure are the **frame to be secured** and the **SecurityLevel**, **KeyIdMode**, **KeySource**, and **KeyIndex** parameters from the originating primitive or automatic request PIB attributes. The outputs from this procedure are the **status** of the procedure and, if this **status** is SUCCESS, the **secured frame**.

The outgoing frame security procedure involves the following steps:

a) Do we need to secure the packet?

If the **SecurityLevel** parameter is zero, the procedure shall set the **secured frame** to be the **frame to be secured** and return with a **status** of SUCCESS.

b) Do we have security enabled?

If the *macSecurityEnabled* attribute is set to FALSE the procedure shall return with a **status** of UNSUPPORTED\_SECURITY.

c) Fetch the KeyDescriptor.

The procedure shall obtain the **KeyDescriptor** using the KeyDescriptor lookup procedure as described in 9.2.2 with the **device addressing mode** set to Destination Addressing Mode field, the **device PAN ID** set to Destination PAN Identifier field, and the **device address** set to Destination Address field. If that procedure fails, the procedure shall return with a **status** of UNAVAILABLE\_KEY.

d) Fetch frame counter.

If using TSCH mode, then ASN is used instead of frame counter, and this step is skipped. If the *FrameCounterPerKey* in the **KeyDescriptor** is set to FALSE, then the procedure shall set the **frame counter** to the *macFrameCounter* attribute, otherwise the procedure shall set the **frame counter** to the *KeyFrameCounter* of the **KeyDescriptor**. If the **frame counter** has the value 0xffffffff, the procedure shall return with a **status** of COUNTER\_ERROR.

e) Insert and fill auxiliary security header.

The procedure shall insert the auxiliary security header into the frame, with fields set as follows:

1) The Security Level field of the Security Control field shall be set to the **SecurityLevel** parameter.

2) The Key Identifier Mode field of the Security Control field shall be set to the **KeyIdMode** parameter.

3) If using TSCH mode Frame Counter Suppression field of the Security Control field is set to one, otherwise the  
d) Get SecurityLevelDescriptor  
See identically named step of section

9.2.3.

Frame Counter field shall be set to the **frame counter** and Frame Counter Suppression field of the Security Control field is set to zero.

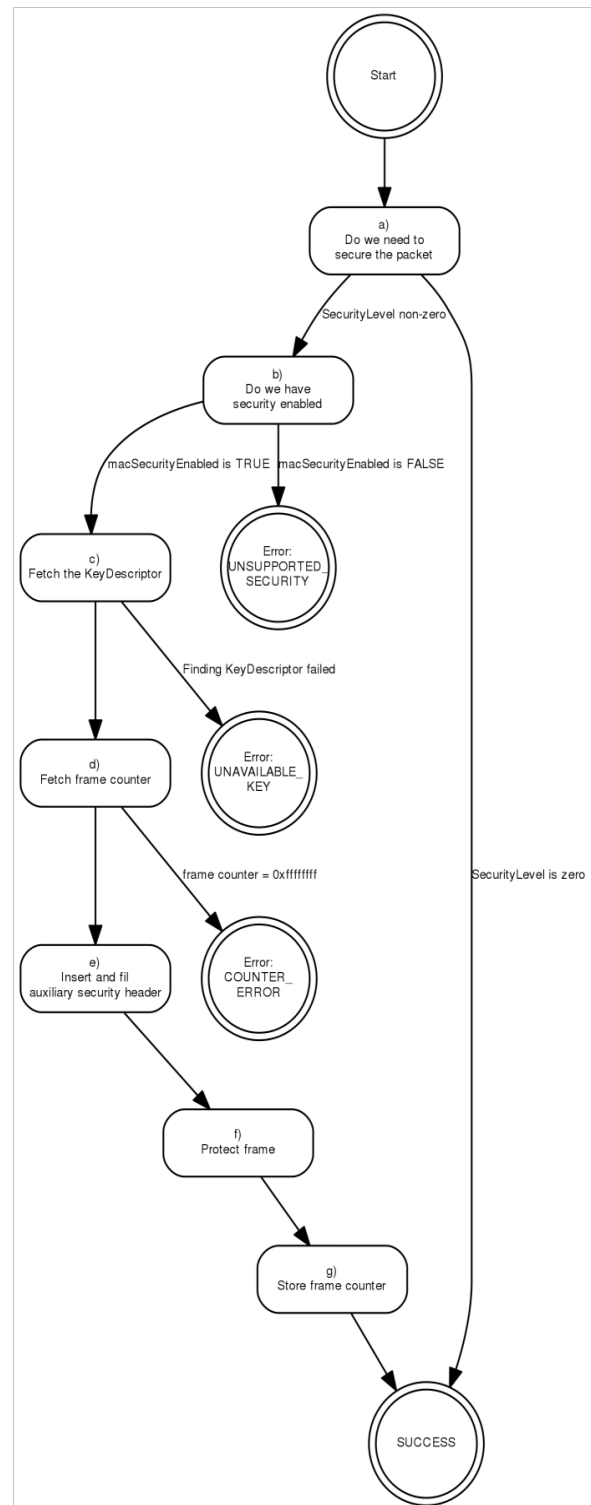
4) If the **KeyIdMode** parameter is set to a value not equal to zero, the Key Source and Key Index fields of the Key Identifier field shall be set to the **KeySource** and **KeyIndex** parameters, respectively.

#### f) Protect frame.

For frames specified in the Table 146, the **Private Payload field** and **Open Payload field** shall be set as indicated there. For frames not specified in Table 146, the **Private Payload field** shall be set to contain all MAC payload fields, and the **Open Payload field** shall be empty. The procedure shall then use the **Private Payload field**, the **Open Payload field**, the *macExtendedAddress*, the **frame counter**, the **SecurityLevel** parameter, and the Key element of the *KeyDescriptor* to produce the **secured frame** according to the CCM\* transformation process defined in 9.3.4.

#### g) Store frame counter

If not using TSCH mode, the procedure shall increment the **frame counter** by one



and store it back (either to the *macFrameCounter* attribute or to the *KeyFrameCounter* of the *KeyDescriptor*).

h) The procedure shall return with a **status** of SUCCESS.

–

#### *New version of section 9.2.2 KeyDescriptor lookup procedure*

The inputs to this procedure are the *KeyIdMode*, *KeySource*, *KeyIndex*, *device addressing mode*, *device PAN ID*, and *device address*. The outputs from this procedure are a **status** and, if passed, a *KeyDescriptor*.

The procedure involves the following steps:

- a) If the *KeyIdMode* parameter is set to 0x00, then for each *KeyIdLookupDescriptor* in the *macKeyIdLookupList*:
  - 1) If the *device addressing mode* is set to NONE, then the *device PAN ID* shall be set to *macPANId*. Otherwise, the *device PAN ID* shall be the value passed to the procedure.
  - 2) If the *device addressing mode* is set to NONE and the frame type is beacon, then the *device address* shall be *macCoordExtendedAddress*.
  - 3) If the *device addressing mode* is set to NONE and the frame type is not beacon, then:
    - i) If the *macCoordShortAddress* attribute is set to 0xffff, then the *device address* shall be set to the *macCoordExtendedAddress*.
    - ii) If the *macCoordShortAddress* attribute is set to a value of 0x0000–0xffffd, then the *device address* shall be set to the *macCoordShortAddress*.
    - iii) If the *macCoordShortAddress* attribute is set to 0xffff, the procedure shall return with a failed **status**.
  - 4) If the *device addressing mode* is set to SHORT or EXTENDED, then the *device address* shall be the value passed to the procedure.
  - 5) If the *device addressing mode*, *device PAN ID*, and *device address* match the *DeviceAddrMode*, *DevicePANId*, and *DeviceAddress* of a *KeyIdLookupDescriptor*, then the procedure returns with the corresponding *KeyDescriptor* and passed **status**.
- b) If the *KeyIDmode* parameter is set to 0x01 and the *KeySource* matches *macDefaultKeySource*, then for each *KeyIdLookupDescriptor* in the *macKeyIdLookupList*, if the *KeyIndex* matches the *KeyIndex* of a *KeyIdLookupDescriptor*, then the procedure returns with the corresponding *KeyDescriptor* and passed **status**.
- c) If the *KeyIDmode* parameter is set to 0x02 or 0x03, then for each *KeyIdLookupDescriptor* in the *macKeyIdLookupList*, if the *KeySource* and *KeyIndex* match the *KeySource* and *KeyIndex* of a *KeyIdLookupDescriptor*, then the procedure returns with the *KeyDescriptor* and passed **status**.
- d) The procedure shall return with a failed **status**.

**CID 1026, 1145, 1146, 1148, 1147**

This is already done in the DF3 draft.

Toyoyuki Kato	Anritsu Engineering Co.,Ltd,	314	9.2.1	17	"Table 146" is incorrect.	Correct it appropriately.
Tero Kivinen	INSIDE Secure	314	9.2.1	25	In Table 146 we say that in beacons following fields are not encrypted: superface specifications, gts info, pending address, and both header IEs and Payload IEs. Are Payload IEs really meant to be sent unencrypted?	Add Payload IEs to the private payload fields.
Tero Kivinen	INSIDE Secure	314	9.2.1	26	In Table 146 we say that in Data only the Data Payload field is encrypted, all other fields are unencrypted. This includes both header and Payload IEs. I would assume we want to make Payload IEs encrypted.	Add Payload IEs to the private payload fields.
Tero Kivinen	INSIDE Secure	314	9.2.1	31	In Table 146 we say that there is no Open Payload fields for the MAC Command frame with version number $\geq 2$ , but the private payload fields does not list command identifier. So we do not include Command Identifier in either column, so we do not know whether it is open or private field?	Add Command Identifier to the Private Payload field column.
Tero Kivinen	INSIDE Secure	314	9.2.1	33	In Table 146 we say that for Acknowledgement frames the full Information Elements field is encrypted, this includes both Header IE and Payload IE fields. Is this intended, or should we only include Payload IE here? If Header IE field is also included to be protected, then we most likely want to protect them also in other frame formats.	Clarify.

Change inbound and outbound processing rules to say that Table 146 only contains exceptions to the generic rule, and that generic rule is that Private Payload Field contains all MAC Payload fields, and Open Payload Field is empty. Change the Table 146 to contain:

Frame type	Private Payload Field	Open Payload Field
Beacon (Frame Version < 2)	Beacon Payload	All other fields in the MAC Payload
MAC Command (Frame Version < 2)	Content	Command Identifier

**CID 1153, 1101, R41, R42**

Tero Kivinen	INSIDE Secure	315	9.2.3	49	Both steps c) and steps e) will set “key identifier mode” / “KeyIdMode”, “key source”/“KeyIndex” and “key index”/“KeyIndex”. I think we need to do this only once.	Modify the step c) so it will copy the values out from the auxiliary security header, and remove the copying from step e), so all auxiliary security header processing is in one step. We need to check which names of those local variables needs to be used in the rest of the processing steps, and perhaps also change them to use separate typographical look.
Tero Kivinen	INSIDE Secure	316	9.2.3	50	CID 470 was not done: In step l) the step will check the frame counter value of 0xffffffff. With TSCH mode the frame counter is not used, instead of 5-octet absolute slot number ASN is used, this test is not needed. Actually the current draft accidentally did that change for step i) not for step l), so move the text from step “i) ... procedure shall determine whether the frame to be unsecured ...” to step “l) The procedure shall set frame counter ...”.	Add “If not using TSCH mode” in front of step l.
Tero Kivinen	INSIDE Secure	316	9.2.3	35	Steps i), j) and k) should be folded in to the step h. It is stupid to call subprocedure to fetch the SecurityLevelDescriptor and then check it here, as we could call SecurityLevelDescriptor validation procedure, that will get that SecurityLevelDescriptor and verify the packet is according to it, and then either return error (either UNAVAILABLE_SECURITY_LEVEL or IMPROVED_SECURITY_LEVEL) or SUCCESS, or pass forward.	As described in the Comment section.
Tero Kivinen	INSIDE Secure	316	9.2.3	49	Steps l and m should be combined.	Replace with “If not using TSCH mode, the procedure shall set frame counter to the Frame Counter field of the frame to be unsecured. If frame counter has the value 0xffffffff, or if the frame counter is less than the FrameCounter element of the DeviceDescriptor, the procedure shall return with a status of COUNTER_ERROR.”

*New version of the Incoming state machine:*

The input to this procedure is the **frame to be unsecured**. The outputs from this procedure are the **status** of the procedure and, if this **status** is SUCCESS, the **unsecured frame**, the **security level**, the **KeyIdMode**, the **KeySource**, the **KeyIndex**, and the **iestatus** for each information element in the the frame. The **iestatus** can either be SKIP or PROCESS, and it will tell the user of the information element whether it should skip the information element processing or whether it should process it.

All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure.

The incoming frame security procedure involves the following steps:

a) Do we have secured frame?

If the Security Enabled field of the **frame to be unsecured** is set to zero, the procedure shall use procedure described in section 9.2.3b.

b) Legacy security?

If the Frame Version field of the **frame to be unsecured** is set to zero, the procedure shall return with a **status** of UNSUPPORTED\_LEGACY.

c) Check for macSecurityEnabled

If the *macSecurityEnabled* attribute is set to FALSE, the procedure shall return **status** of UNSUPPORTED\_SECURITY.

d) Parse Auxiliary Security Header

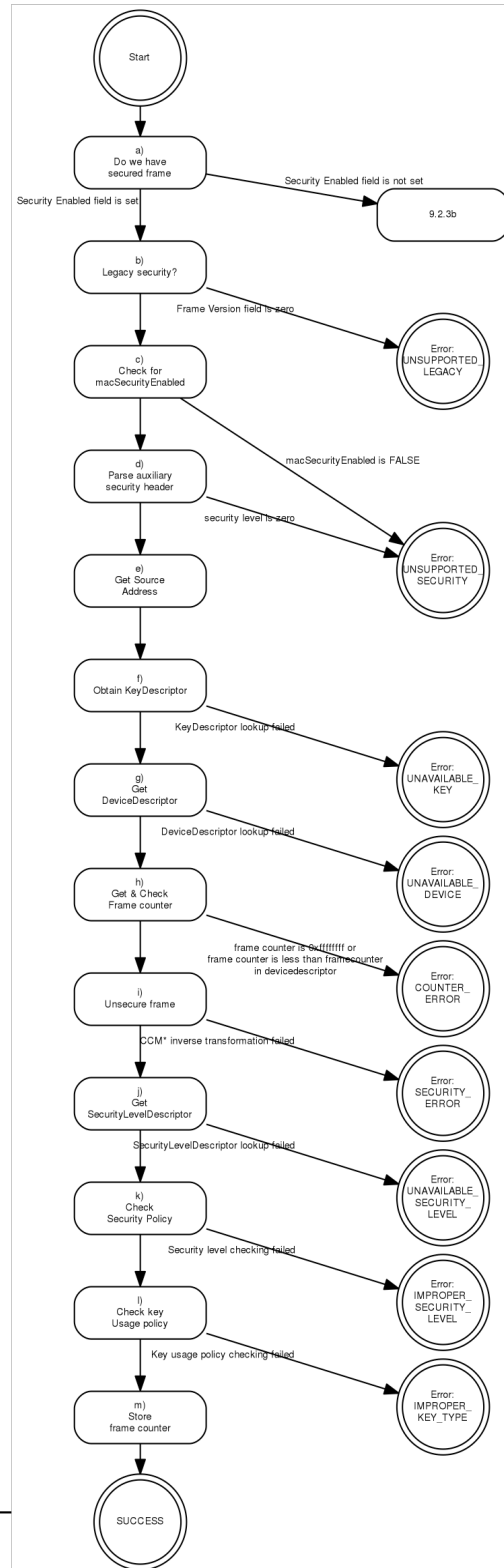
The procedure shall set the **security level** and the **key identifier mode** to the corresponding fields of the Security Control field of the auxiliary security header of the **frame to be unsecured**, and the **key source** and **key index** to the corresponding fields of the Key Identifier field of the auxiliary security header of the **frame to be unsecured**, if present. If the resulting **security level** is zero, the procedure shall return with a **status** of UNSUPPORTED\_SECURITY.

e) Get Source Address

The **device PAN ID** shall be set to the Source PAN Identifier field, if it is present. If the PAN ID compression field is set to one, then the **device PAN ID** shall be set to the Destination PAN Identifier field. The **device addressing mode** shall be set according to the Source Addressing Mode field, as defined in Table 147. The **device address** shall be set to the Source Address, if present.

f) Obtain KeyDescriptor

The procedure shall obtain the **KeyDescriptor** using the KeyDescriptor lookup procedure as described in 9.2.2 with using the **KeyIdMode**, **KeyIndex**, **KeySource**, **device addressing**





*mode*, *device PAN ID*, and *device address*. If that procedure fails the procedure shall return with a *status* of UNAVAILABLE\_KEY.

#### g) Get DeviceDescriptor

The procedure shall obtain the *DeviceDescriptor* using the *DeviceDescriptor* lookup procedure described in 9.2.4, and if that succeeded and *FrameCounterPerKey* element of *KeyDescriptor* is set to TRUE the procedure shall obtain the *DeviceFrameCounter* using the *DeviceFrameCounter* lookup procedure described in 9.2.4b. If either of those procedures fail, then the procedure shall return with a *status* of UNAVAILABLE\_DEVICE.

#### h) Get & Check Frame Counter

If using TSCH mode, then ASN is used instead of frame counter, and this step is skipped. The procedure shall set *frame counter* to the *Frame Counter* field of the *frame to be unsecured*. If the *FrameCounterPerKey* element of the *KeyDescriptor* is set to FALSE then *frame counter check value* is set to be the *DeviceFrameCounter* element of *DeviceDescriptor*, otherwise the *frame counter check value* is set to be the *FrameCounter* element of the *DeviceFrameCounter*. If *frame counter* has the value 0xffffffff, or the *frame counter* is less than the *frame counter check value*, the procedure shall return with a *status* of COUNTER\_ERROR.

#### i) Unsecure frame

For frames specified in the Table 146 the *Private Payload field* and *Open Payload field* shall be set as indicate there. For frames not specified in Table 146 the *Private Payload field* shall be set to contain all MAC payload fields, and the *Open Payload field* shall be empty. The procedure shall then use the *Private Payload field*, the *Open Payload field*, the *ExtAddress* element of the *DeviceDescriptor*, *frame counter*, the *security level*, and the *Key* element of the *KeyDescriptor* to produce the *unsecured frame*, according to the CCM\* inverse transformation process described in the security operations, as described in 9.3.5. If the CCM\* inverse transformation process fails, the procedure shall return with a *status* of SECURITY\_ERROR.

#### j) Get SecurityLevelDescriptor

The procedure shall obtain the *SecurityLevelDescriptor* by passing the frame type and, if the frame is a MAC command, the Command Identifier, to the *SecurityLevelDescriptor* lookup procedure described in 9.2.5. If that procedure fails, the procedure shall return with a *status* of UNAVAILABLE\_SECURITY\_LEVEL.

#### k) Check Information Element Security

If the IE present field of the frame to be unsecured is set to one, the procedure shall determine whether the *frame to be unsecured* conforms to the security level policy by passing the *SecurityLevelDescriptor* and the *security level* to the incoming information element security level checking procedure, as described in 9.2.6b. That procedure will return the *iestatus* for each information element in the frame.

**l) Check information Element Key Usage Policy**

If the IE present field of the frame to be unsecured is set to one, the procedure shall determine whether the **frame to be unsecured** conforms to the key usage policy by passing the **iestatus**, **KeyDescriptor**, the frame type, and, if the frame is a MAC command, the Command Identifier field, to the incoming information element key usage policy checking procedure, as described in 9.2.7b. That procedure will update the **iestatus** for each information element in the frame as specified by the key usage policy.

**m) Check Security Policy**

The procedure shall determine whether the **frame to be unsecured** conforms to the security level policy by passing the **SecurityLevelDescriptor** and the **security level** to the incoming security level checking procedure, as described in 9.2.6. If that procedure returns with a failed status, the procedure shall return with a **status** of **IMPROPER\_SECURITY\_LEVEL**.

**n) Check Key Usage Policy**

The procedure shall determine whether the **frame to be unsecured** conforms to the key usage policy by passing the **KeyDescriptor**, the frame type, and, if the frame is a MAC command, the Command Identifier field, to the incoming key usage policy checking procedure, as described in 9.2.7. If that procedure fails, the procedure shall return with a **status** of **IMPROPER\_KEY\_TYPE**.

**o) Store frame counter**

If using TSCH mode, the ASN is used instead of frame counter, and this step is skipped. The procedure shall increment **frame counter** by one and if *FrameCounterPerKey* element of the **KeyDescriptor** is set to FALSE, then the *DeviceFrameCounter* element of the **DeviceDescriptor** is set to the resulting value, otherwise the *FrameCounter* element of the **DeviceFrameCounter** is set to the resulting value.

**p) Return SUCCESS**

The procedure shall return with a **status** of SUCCESS.

**9.2.3b Incoming frame security procedure for security level zero frames**

This procedure is used to process the frames which has security level of zero. The input to this procedure is the **frame to be unsecured**. The outputs from this procedure are the **status** of the procedure and, if this **status** is SUCCESS, the **unsecured frame**.

The incoming frame security procedure for security level zero frames involves following steps:

## a) Check for macSecurityEnabled

if the *macSecurityEnabled* attribute is set to FALSE, the procedure shall set the **unsecured frame** to be the **frame to be unsecured** and return with a **status** of SUCCESS.

## b) Get Source Address

See identically named step of the section 9.2.3.

## c) Get DeviceDescriptor

The procedure shall obtain the **DeviceDescriptor** using the **DeviceDescriptor** lookup procedure described in 9.2.4 using security level set to zero. If that procedure fails, then the procedure shall return with a **status** of UNAVAILABLE\_DEVICE.

## d) Get SecurityLevelDescriptor

See identically named step of section 9.2.3.

## e) Check Information Element Security

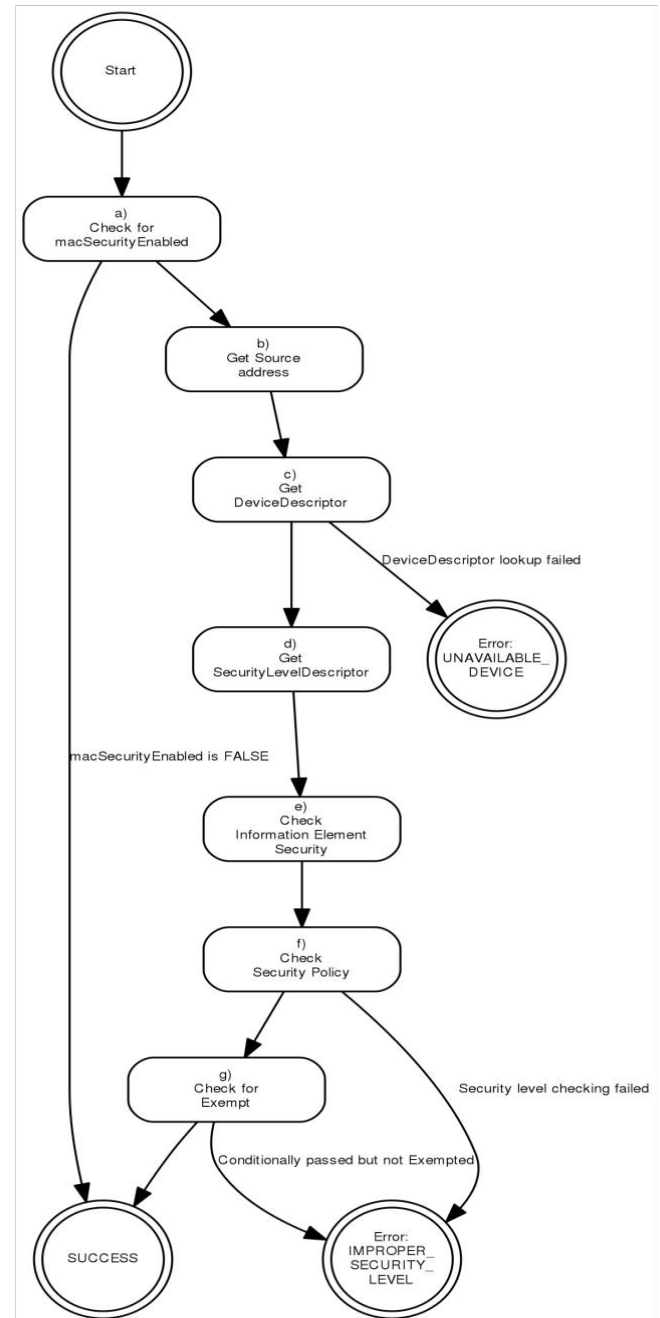
See identically named step of section 9.2.3.

## f) Check Security Policy

The procedure shall determine whether the **frame to be unsecured** conforms to the security level policy by passing the **SecurityLevelDescriptor** and the **security level** of zero to the incoming security level checking procedure, as described in 9.2.6. If that procedure returns with a failed status, the procedure shall return with a **status** of IMPROPER\_SECURITY\_LEVEL.

## g) Check for Exempt

If the incoming security level checking procedure of step f) had as output the 'conditionally passed' status and the *Exempt* element of the **DeviceDescriptor** is set to FALSE, the procedure shall return with a **status** of IMPROPER\_SECURITY\_LEVEL.



#### h) Return SUCCESS

The procedure shall set the **unsecured frame** to be the **frame to be unsecured** and return with a **status** of SUCCESS.

—

*New version of the DeviceDescriptor lookup procedure.*

### 9.2.4 DeviceDescriptor lookup procedure

The inputs to this procedure are the **device addressing mode**, the **device PAN ID**, and the **device address**. The output from this procedure is a passed or failed status and, if passed, a **DeviceDescriptor**.

The DeviceDescriptor lookup procedure involves the following steps:

- a) If the **device addressing mode** is set to NONE, then the **device PAN ID** shall be set to *macPANId*. Otherwise, the **device PAN ID** shall be the value passed to the procedure.
- b) If the **device addressing mode** is set to NONE, then:
  - 1) If the *macCoordShortAddress* attribute is set to 0xffffe, then the **device address** shall be set to the *macCoordExtendedAddress*.
  - 2) If the *macCoordShortAddress* attribute is set to a value of 0x0000–0xffffd, then the **device address** shall be set to the *macCoordShortAddress*.
  - 3) If the *macCoordShortAddress* attribute is set to 0xffff, the procedure shall return with a failed status.
- c) If the **device addressing mode** is set to SHORT or EXTENDED, then the **device address** shall be the value passed to the procedure.
- d) For each DeviceDescriptor in *macDeviceTable*, if the **device PAN ID** matches the *PANId* element and the **device address** matches the *ShortAddress* element, if the **device addressing mode** is set to SHORT, or the *ExtAddress* element, if the **device addressing mode** is set to EXTENDED, then the procedure shall return with the corresponding **DevicesDescriptor** and a passed status.
- e) The procedure shall return with a failed status.

—

*New subsection 9.2.4b DeviceFrameCounter lookup procedure.*

*Here is the new subsection for fetching DeviceFrameCounter:*

#### **9.2.4b DeviceFrameCounters lookup procedure**

The inputs to this procedure are the **DeviceDescriptor** and **KeyDescriptor**. The output from this procedure is a passed or failed status and if passed a DeviceFrameCounter.

The DeviceFrameCounter lookup procedure involves the following steps:

- a) For each DeviceFrameCounter in *DeviceFrameCounterTable* of the **KeyDescriptor**, if the *ExtAddress* element of DeviceFrameCounter matches the *ExtAddress* element of the **DeviceDescriptor**, then the procedure shall return with corresponding DeviceFrameCounter and a passed status.
- b) The procedure shall return with a failed status.

–

*Highlighted version of 9.2.5, and verify it is correct.*

Work in progress.

–

*Highlighted version of 9.2.6, and verify it is correct.*

Work in progress.

–

*New subsection 9.2.6b*

Work in progress.

–

*Highlighted version of 9.2.7, and verify it is correct.*

Work in progress.

–

*New subsection 9.2.7b*

Work in progress.

–

**CID1151**

This is already done in the DF3 draft.

Tero Kivinen	INSIDE Secure	324	9.4.1.3	34	Remove extra "either".	Replace ".. is either an incrementing shared global frame counter such as ASN." with "... is an incrementing shared global frame counter such as ASN."
--------------	------------------	-----	---------	----	------------------------	--

Accept, as described in Proposed Change column.

**CID1105**

Tero Kivinen	INSIDE Secure	325	9.5	39	R29 was not done properly. The table 153 still refers to table 157 (DeviceDescriptor), when it should refer to 158 (DeviceDescriptorSecLevelZero). (And the link to table 157 is broken :-). Also the table 154 DeviceDescriptorList should refer to the table 157 (DeviceDescriptor).	In table 153, change macDeviceTable to macDeviceExemptTable, and change reference to table 158 instead of 157. Change description "DeviceDescriptorSecLevelZero for each device which is allowed to be exempted for the security." Change Type of DeviceDescriptorList to "Set of DeviceDescriptors, as defined in Table 157."
--------------	---------------	-----	-----	----	--	--

Accept, as described in Proposed Change column.

**CID1104**

Tero Kivinen	INSIDE Secure	325	9.5	46	R28 was not done. Move the macFrameCounter from the table 153 to KeyDescriptor table, i.e. table 154	Move the macFrameCounter from the table 153 to KeyDescriptor table, i.e. table 154
--------------	------------------	-----	-----	----	--	--

Accept, as described in Proposed Change column.



**CID1154**

Tero Kivinen	INSIDE Secure	522	23.3.2	14	The security of the fragmentation is still completely broken.	It needs to be fixed. It needs a bit different nonce generation format.
--------------	---------------	-----	--------	----	---	---

Add new subsection 9.3.2.3:

—

**9.3.2.3 CCM\* nonce for Fragments**

The CCM\* nonce for the fragments shall be formatted as shown in Figure xxx. The Source Address and security level as set as in defined in 9.3.2.1, and the Fragment Frame Counter is set to match the *phyFragmentFrameCounter*, and the Fragment number is set to match the fragment number of the fragment. Fragment indicator shall be set to 1.

Figure xxx – CCM\* nonce for fragments

Octets: 8	Bits: 0-25	26-31	32-35	36	37-39
Source Address	Fragment frame counter	Fragment number	Reserved	Fragment indicator	Security level

—

*We might want to split the security level field in the 9.3.2.1 to have 3 bits of security level and 5 bits of reserved just to be clear how the 3-bit security level field is formatted inside the 1-octet space reserved for it.*

*I am not sure whether it is good idea to refer to the *phyFragmentFrameCounter* here, or whether we should just use term *Fragment frame counter*, and modify the text in section 23.3.1 to match. The section 23.3 might need even more changes, it still needs to be checked properly.*