System Specification

enocean alliance
No Wires. No Batteries. No Limits.

Security of EnOcean Radio Networks

V2.4

San Ramon, CA, USA, 2018

Ver.	Editor	Change	Date
1.0	MF	Creation	05.03.2012
1.1	MF	Inclusion of AES CBC. Secure telegrams with/without non-secure RORG.	03.07.2012
1.2	MF	Page numbering	31.07.2012
1.3	AP	Removing ARC4 and editorial changes	31.10.2012
1.4	MH	Editorial on CMAC computing	22.11.2012
1.5	MH	Added PSK Teach In and VAES extensions above 16 bytes.	17.06.2013
1.6	MF	Abstraction from telegram to message	03.07.2013
1.7	MF	Teach-in unidirectional/bidirectional procedure described	10.07.2013
1.8	MF	RLC synchronization. Message-to- ERP1 and ERP2 conversion	11.07.2013
1.9	MF	PSK CRC explained. Description of teach-in procedure improved	26.07.2013
2.0	MH	Added High Security definition - renamed public key to VAES INIT Vector - CMAC in teach-in telegram eliminated	11.08.2014
2.1	FG	Added Test vectors	09.11.2016
2.2	MF	Test vectors encryption and CMAC calculations described in detail	12.07.2017
2.3	MH	Best practice examples, Secure chaining, 32 bit RLC. Removed Mutual authentication with RLC as NONCE from high security.. Changed RLC description.	09.05.2018
2.4	TM	Add secure chaining example, updated the sec_cdm mechanism	27.11.2018

copyright or trademark rights, or (2) any implied warranties of merchantability, fitness for a particular purpose, title or non-infringement.

In no event will the EnOcean Alliance be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for any other direct, indirect, special or exemplary, incidental, punitive or consequential damages of any kind, in contract or in tort, in connection with this document or the information contained herein, even if advised of the possibility of such loss or damage. All Company, brand and product names may be trademarks that are the sole property of their respective owners.

The above notice and this paragraph must be included on all copies of this document that are made.

The EnOcean Alliance Security of EnOcean networks Specification is available free of charge to companies, individuals and institutions for all non-commercial purposes (including educational research, technical evaluation and development of non-commercial tools or documentation.)

This specification includes intellectual property („IPR“) of the EnOcean Alliance and joint intellectual properties („joint IPR“) with contributing member companies. No part of this specification may be used in development of a product or service for sale without being a participant or promoter member of the EnOcean Alliance and/or joint owner of the appropriate joint IPR. EnOcean Alliance grants no rights to any third party IP, patents or trademarks.

These errata may not have been subjected to an Intellectual Property review, and as such, may contain undeclared Necessary Claims.

EnOcean Alliance Inc.
2400 Camino Ramon, Suite 375
San Ramon, CA 94583
USA
Graham Martin Chairman & CEO EnOcean Alliance

Content

1 Introduction

This document specifies the security concept proposal for the Energy Harvested Wireless protocol (EHW). This concept was specially designed for devices powered by energy harvesters and is based on the EHW lower protocol lower layers. The objective of the design was to keep the energy requirements and µC resources for the implementation of the security as low as possible.

1.1 Terms & Abbreviations

Term / Abbr.	Description
µC	Microcontroller (external)
API	Application Programming Interface
APP	Application
Data	Payload of EHW telegram
EEP	EnOcean Equipment Profile
EHW	Energy Harvested Wireless protocol
EO	EnOcean
ESP3	EnOcean Serial Protocol V3
RLC	Rolling Code
CMAC	Cipher Based Message Authentication Code
Gateway	Module with a bidirectional serial communication connected to a HOST
Device	Customer end-device with an integrated EnOcean radio module
Nonce	Number which is used once

Table 1. Abbreviations used in this document.

2 Scenarios

The EHW protocol is used in a variety of applications. However, the major usage is in products within building automation. Some typical situations where a secure protocol can be required are:

- Thermostat vacation mode - A family leaves their home for several weeks. They set their thermostat in vacation mode. Obscuring data from the thermostat will prevent an intruder to know the occupancy state of the house.
- Window control – A building has a window installed whose opening is controlled by temperature and CO2 sensors. Applying a secure radio protocol the window receiver should ignore messages that have been already used. This will prevent an intruder to gain unauthorized access to the building by recording packets and replaying them.

  Automated meter reading – Polling a thermostat per radio enables the collection of meter information without the necessity of a person entering the house. By obscuring data from the thermostat it will be prevented that an intruder obtains private information from the transmitted data. To prevent an intruder forging the heating consumption the system must resist reply-attacks.

2.1 Attacker scenarios

In such fashions could be compromised the normal operation of an EO radio system:

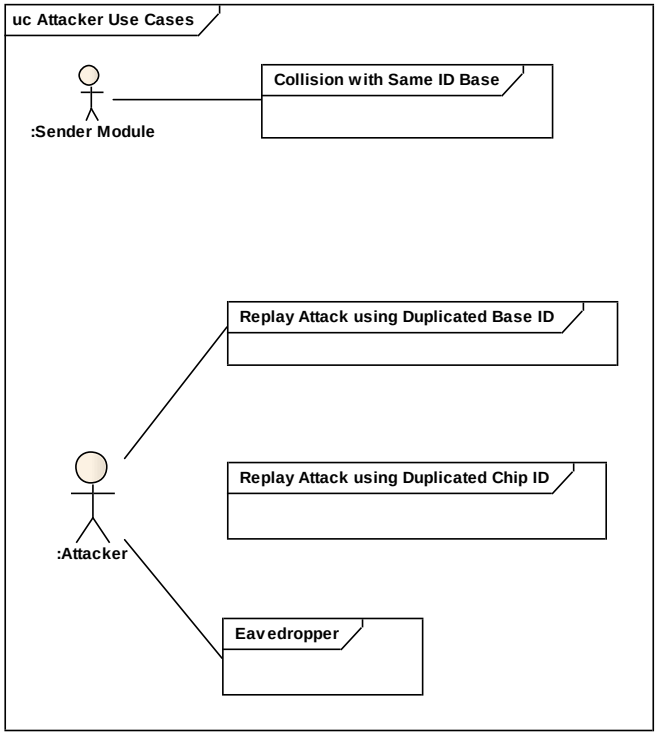


Figure 1. The picture shows the possible interferences and attacks in the actual EO radio communication. Attacks other than the ones shown in this figure will not be considered in this

documentation and in the implementation of a secure radio protocol. Attacks not considered include, for instance, the physical manipulation of a sender or a receiver module; continuous wave sending; power-supply sabotage. From the four scenarios depicted, the first (Collision with the same ID base) corresponds to an unintentional control of a receiver by another EO user. The last three scenarios represent, however, an intentional attack. Unauthorized listening of information that is being transmitted is called *eavesdropping*.

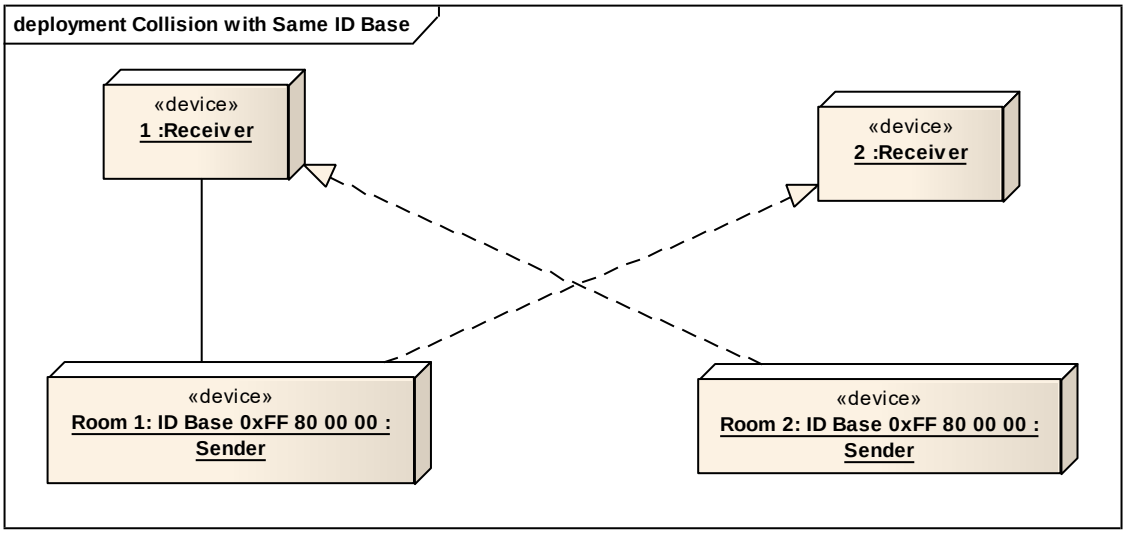


Figure 2. This scenario is not a proper attack. It's simply an ID duplication within the range of the base ID (see what is EnOcean Base ID). The duplication happens because two different users programmed their sender modules with identical ID within the base ID range. When the user in room 1 sends a signal with Sender 1 he controls unluckily also Receiver 2, in room 2. A symmetrical situation happens when the Sender 2 in room 2 is operated.

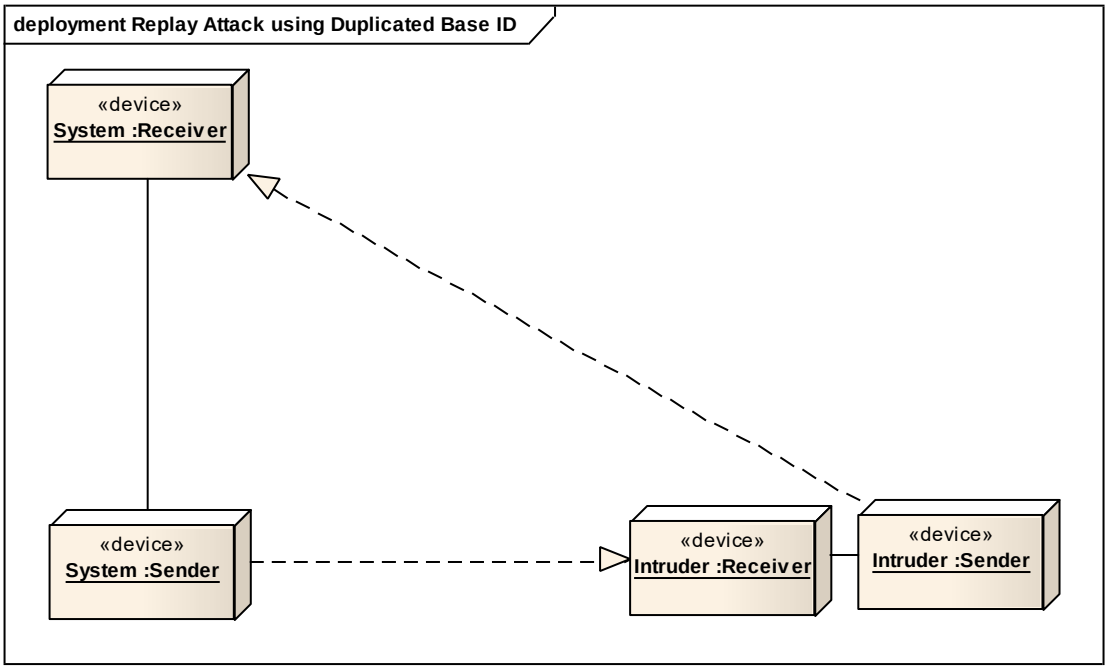


Figure 3. In this scenario an attacker programs the ID base taking intentional control of the system receiver. To do this, the intruder simply listens to the sender ID. It programs then this same Base ID in his sender module, and controls the system receiver. All this is possible using EO tools without performing reverse engineering.

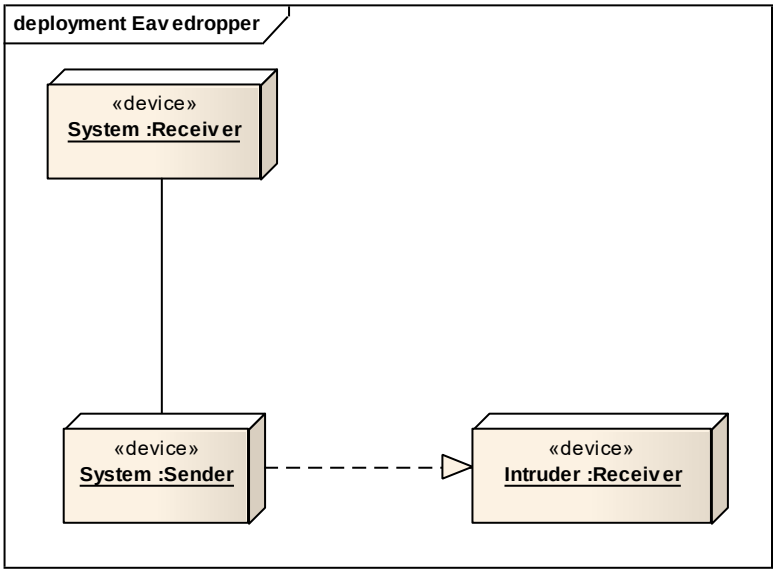


Figure 4. With the only help of an EO receiver it is possible to hear the communication between an EO sender and a receiver. This is undesired if the information being transmitted is private. This is the case of metering reading systems.

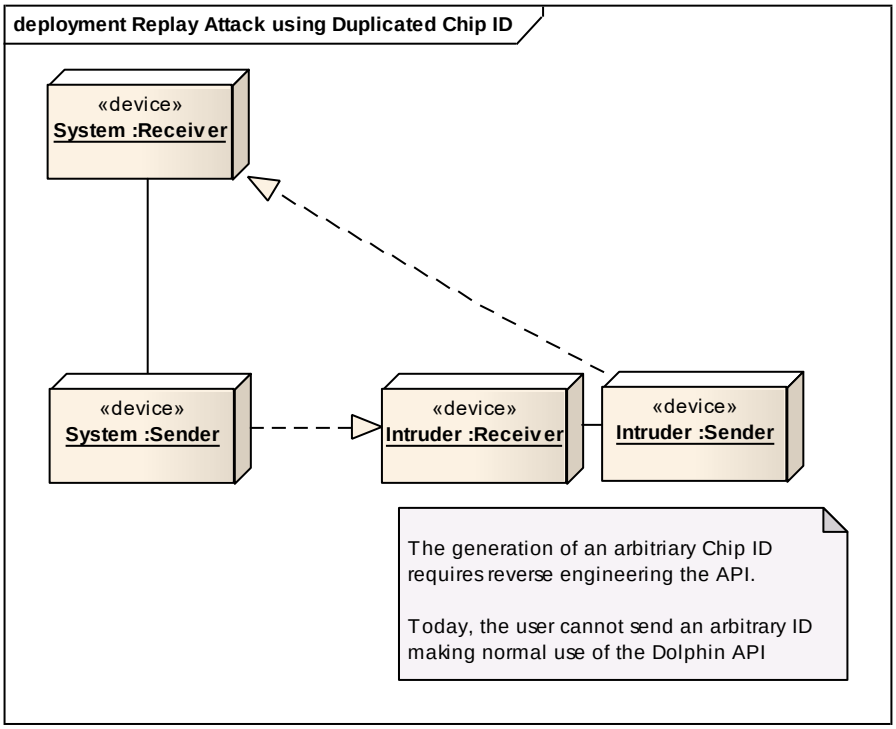


Figure 5. In a more sophisticated attack version the attacker listens to the telegrams in the air. The attacker sends a telegram with the same chip ID (replay) as the one in the system sender. This can be done by using a HW tool that reproduces listened telegrams, or by applying reverse engineering to the Dolphin API and modifying the sendTelegram functions which contain the chip ID.

2.2 System Architecture

Typical system architecture using EHW protocol is showed on the figure below:

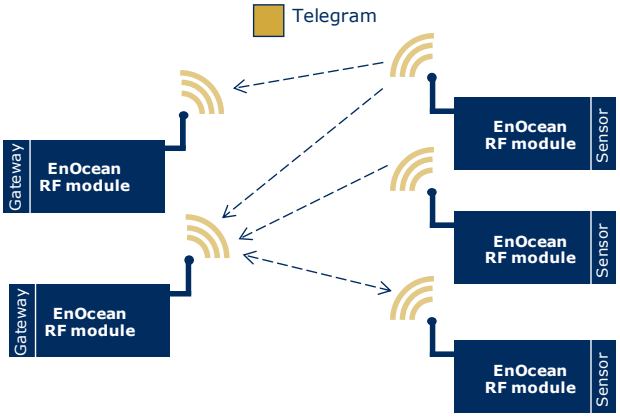


Figure 6. EHW system architecture

In an EHW system most common communication pattern is unidirectional. However there are installations where bi-directional communication is required for instance between two Gateways, in

SmartACK or in Remote Management concept. Thus the security concept has to be designed in a way that it can be applied to N:M communication patterns.

3 Security for operation mode

To handle security of EnOcean Radio networks 4 new ORGs will be created according the following table:

R-ORG	Description
0x30	Secure telegram. This message was not created from a non-secure counterpart. A message with this RORG was created by the secure application and the data may interpreted by a Teach-In-Process outside of this specification (i.e. EEP or GP).
0x31	Secure message that transport the original R-ORG non secure code.
0x32	Non-secure message type that results from the decryption of a secure message with R-ORG 0x30.
0x35	Secure Teach-In telegrams transmit private key and rolling to the communication partner

Table 2. New ORGs for security handling.

The EHW security is implemented on the OSI presentation layer of the EHW protocol stack.

The security strategies discussed in the next chapters are applied to messages. Messages abstract the concept of telegrams. Messages do not specify a certain byte order. They are to be imagined as C structures with all necessary members to contain the information stored in a telegram. A message is a generalisation of serial and radio telegrams.

A message contains all fields that a telegram may have: the R-ORG, DATA, Sender ID, receiver ID, repeater counter as well as the security specific members like RLC, CMAC, SLF that will be explained in the next chapters.

EO secure messages follow a flexible schema made up of different freely combinable security mechanisms. Through the combination of all security strategies a higher degree of security can be reached at the expense of a high amount of energy spent due to higher processing and transmission time.

The security mechanism may transform the DATA and R-ORG fields of the non-secure message. Other fields like the message sender ID, receiver ID, repeater counter are not affected or altered. The RLC and CMAC may be added. Not modified fields like sender ID, received ID or repeater counter are not depicted through the chapter when a message is represented.

The following figure depicts the relevant secure radio message structure members, together with examples of application scenarios, attacks that can be blocked and the qualitative energy factor they represent.

MESSAGE	Scenario	Replay Attacks	Eaves-dropping	Energy & resource demand
R-ORG DATA	Unsecured	Yes	Yes	Energy Harvester
R-ORG S DATA	Thermostat & Vacation Mode	Yes	No	
R-ORG S DATA RLC CMAC	Window Control	No	Yes	
R-ORG S DATA RLC CMAC	Automated Meter Reader	No	No	Line Powered

Table 3. The table shows the different secure messages structures for typical applications EnOcean secure messages can be built up with a higher or lower degree of security, which allows to find a balance between security and energy consumption. A secure message is identified by specific R-ORG codes represented as R-ORG S. Obscured will be only the DATA of the message and optionally the original R-ORG. Message members in ocher indicate the usage of encryption. **Any combination of DATA encrypted/not encrypted, RLC present/not present, and CMAC present/not present is possible.** In the table only some typical combinations were depicted.

The structure of the secure telegram is negotiated between the devices within the teach-in procedure. See 4.

3.1 Message structure

Figure 7. Relevant message members. In what follows the DATA member includes the DATA and OPTIONAL DATA.

3.1.1 R-ORG

The message R-ORG identifies the type of message that is being sent or received. The message R-ORG is 1-byte long. Secure messages are identified by the codes 0x30, 0x31 or 0x35, denoted generally *R-ORG S* in the Table 3.

A code different from those identifies a non-secure message.

3.1.2 DATA

Under DATA will always be understood the telegram DATA field and the OPTIONAL DATA, if it exists. The DATA is a byte concatenation of both DATA + OPTIONAL DATA.

The data is encrypted using any of the encryption algorithms described in section 5.

The DATA encryption algorithm is indicated to the receiver during the teach-in procedure.

The length of the DATA and the interpretation is defined by the application.

3.1.3 RLC (ROLLING CODE)

This field contains a code that changes according to a predefined algorithm (see 5.7). Sender and receiver must keep this code synchronize. The code changes every time a message is sent by the radio transmitter. If a message RLC does not coincide with the expected RLC value the receiver rejects the message.

RLC is MSB first.

The teach-in procedure synchronizes the RLC in the receiver with the one in the receiver.

The RLC field is actually optional. Although the RLC may not be sent by the transmitter it can be used internally in the transmitter and receiver modules to perform the calculation of the CMAC code and DATA field encryption. See 5.8. By reception of the CMAC the receiver indirectly test the correctness of the RLC.

RLC field format: RLC byte length, RLC algorithm and RLC presence in the telegram- are communicated to the receiver during the teach-in procedure.

Details of the RLC algorithm are explained in section 5.7

3.1.3.1 Synchronizing RLC when synchronization window is lost

In the event that the RLC in the receiver losses the RLC window no communication between sender and receiver will be possible.

In order to synchronize the RLC in the receiver the sender must send a teach-in telegram. See chapter 4. The receiver does not have to be put into teach-in mode. The current RLC in the teach-in telegram can be used now by the receiver to synchronize with the sender.

3.1.4 CMAC

The CMAC is the cipher-based message authentication code field –with MSB first. Its mission is to guarantee that the message is genuine, meaning that no other party that the expected one transmitted the message.

The CMAC is dependent on the private key (see 4.1.5), a public vector, the message bytes and, if it exists, the rolling code (see 3.1.3).

For maximal security the RLC should be present but not transmitted. The MAC field code changes for every sent message. The way the CMAC changes is unpredictable unless the key and the RLC state are known.

The algorithm to calculate CMAC is explained in the chapter 5.8

3.2 Transforming secure and unsecure messages

This chapter explains how to create secure messages taking as basis an unsecure message and vice versa.

In the explanation message fields such as sender ID, receiver ID and other message fields, not modified by the security transformations, will be not indicated. The reader must suppose that these fields are part of the message.

3.2.1 Transforming an unsecure message into a secure message with R-ORG encapsulation

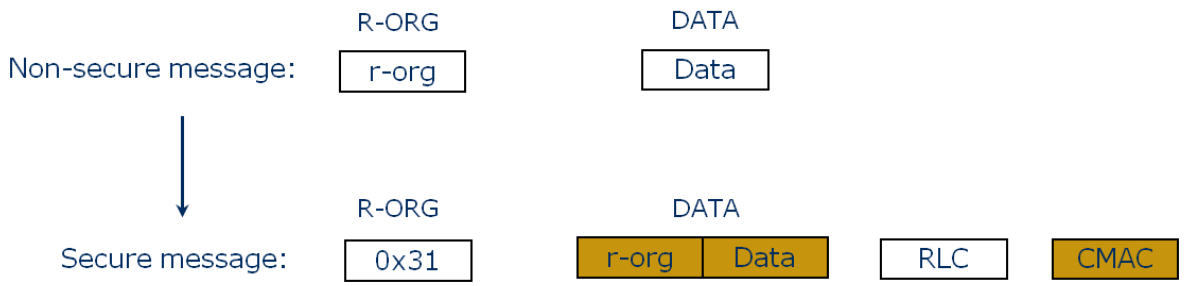


Figure 8. An unsecure message is transformed into a secure message that includes the original R-ORG by 1- transforming the message R-ORG code to 0x31. 2- The unsecure R-ORG field code and the unsecure DATA fields are concatenated and then encrypted. 3- The result of the encryption is stored in the most significant bytes of the secure message DATA field. 4- A rolling code, RLC, might be calculated (section 5.7). 5- Finally, it is possible to add a CMAC code (chapter 5.8). Other message fields like sender ID, receiver ID or repeater counter are not modified.

3.2.2 Transforming a secure message with R-ORG encapsulation into a non-secure message

In this case, a secure message with R-ORG 0x31 is transformed into a non-secure message. The transformation implies decrypting the encrypted R-ORG and Data information which is to be found concatenated in the DATA field of the message.

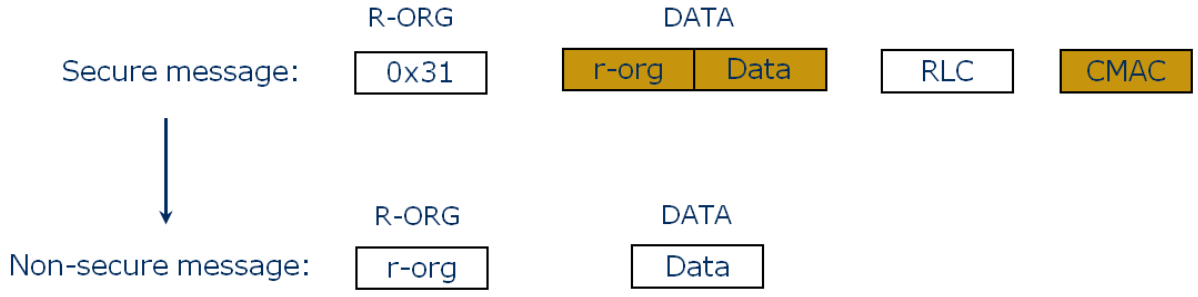


Figure 9. A secure message with R-ORG 0x31 contains the unsecured R-ORG encrypted in the DATA field. The DATA field of the message must be decrypted, maybe with the help of the RLC. The first decrypted byte represents the message R-ORG of the unsecure message. The rest of the decrypted DATA field bytes represent the unsecure message DATA bytes. CMAC does not play a role in the decryption. This field will be checked for authentication of the telegram.

3.2.3 Transforming an unsecure message without R-ORG encapsulation into a secure message

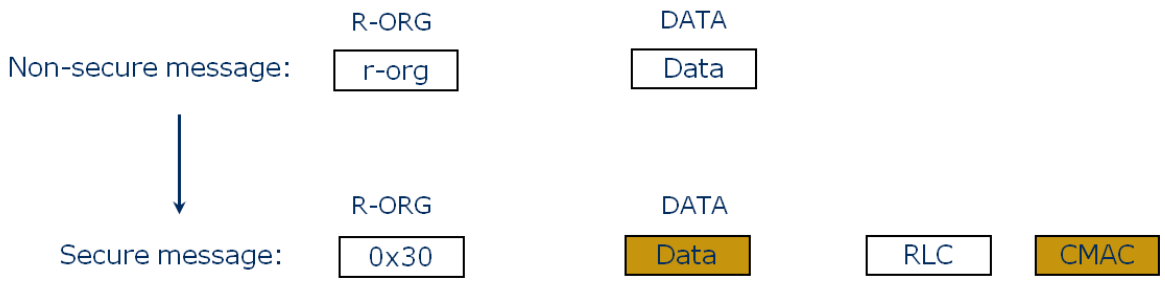


Figure 10. An unsecure message is transformed into a secure message that does not include the original R-ORG by 1- rewriting the message R-ORG code to 0x30. 2- The unsecure DATA field is encrypted. 3- The result of the encryption is stored in the secure message DATA field. 4- A rolling code might be calculated. 5- Finally, it is possible to add a CMAC code. The purpose of not including the original unsecure message in the secure message DATA field is to save energy at transmission time.

3.2.4 Transforming a secure message without R-ORG encapsulation

In this case, a secure message with R-ORG 0x30 is transformed into a non-secure message. The transformation implies decrypting the DATA field information.

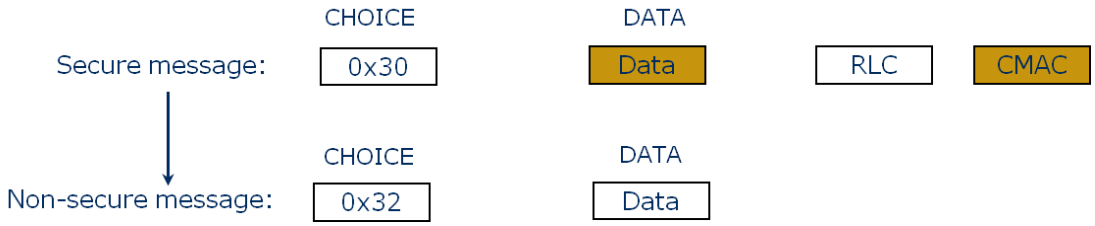


Figure 11. The secure message with R-ORG 0x30 does not include the non-secure R-ORG information in any form. In the conversion from secure to non-secure message the latter becomes the R-ORG = 0x32.The DATA field is decrypted, maybe using the RLC field. The CMAC does not play a role in the decryption. It is only used for authentication purposes.

4 Security for teach-in mode

To configure the details of the secure communication in operation mode a teach-in procedure mode must be executed. Within the teach-in procedure the parts involved in the communication transmit to each other the encryption method, key, rolling code, rolling code size and CMAC size that will be used during the operation mode.

The teach-in procedure can be set up to be an unidirectional or bidirectional process. See 4.1.2 to learn how to configure this option.

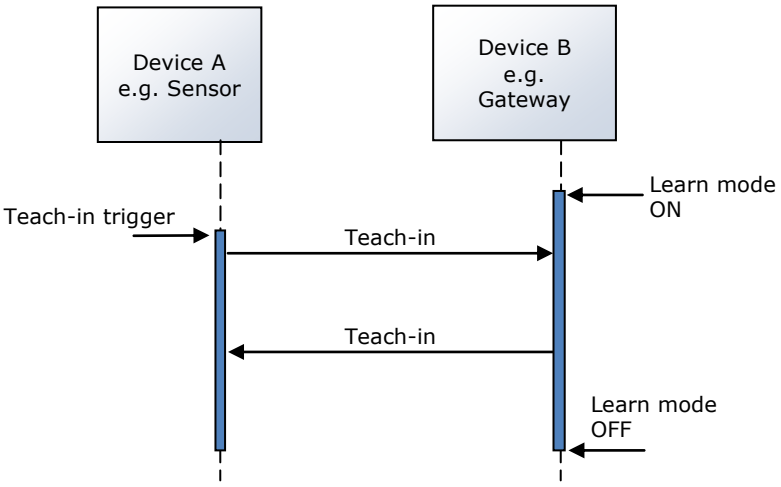


Figure 12. Schematic representation of the most general bidirectional teach-in procedure. In the case of unidirectional security teach-in Device B does not send a teach-in message.

Firstly, the Device B must be set in learning mode to accept the teach-in messages from Device A. The Device A sends the security teach-in message (are described in chapter 4.1) whenever its specific trigger is activated. After reception of the teach-in message the Device B stores the security parameters of Device A: these parameters include the Device's A private key, KEY_s, current RLC, RLC_s, RLC_s size and $CMAC_s$ size and way of encrypting information. The KEY_s and RLC_s can be sent encrypted by the sender using the so called pre-shared key, PSK_s. More details under chapters 4.1.2.3 and 4.2..

If the process is bidirectional the Device B, a gateway, for instance, answers back with a security teach-in message. This teach-in message contains as receiver-ID the ID of the Device A. If the Device B encrypts its teach-in message it will make use of the same PSK_s key of the Device A. In the second security teach-in depicted in the picture the Device B informs the Device A of its own KEY_g and RLC_g and $CMAC_g$. The format of the teach-in messages sent by Device A and Device B are the same.

The teach-in delivered by Device B must occur in worst case 500ms after the reception of the teach-in sent by Device A. The Device's A time-out for the reception of a teach-in is 750ms.

Before Device A sends the security teach-in message the receiver is put into teach-in mode – active listening for teach-in messages. The teach-in method is limited typically to 30 seconds. After this time-out the module leaves its teach-in mode, and returns typically to its operation mode. Teach-in messages are not accepted until the next activation of the teach-in mode.

The possible methods for the teach-in execution are:

- Via wireless from the transmitter to the receiver
- Via serial interface to the receiver through a third party
- Others

Execution via serial interface or other methods are not part of this specification and are rather application / use case specific.

The execution of the teach-in process via wireless leads to two possibilities:

- Teach-in message is sent in plain text (no encryption in the information is performed). This means that any listener can eavesdrop the information
- Parts of the teach-in message are encrypted. For the encryption a pre-shared key is used. Encrypted are the RLC and KEY. Message structure is listed below. Details about this execution can be found in chapter 4.2

4.1 Message structure

The teach-in message has the following secure-specific fields:

Figure 13. Secure teach-in message. Teach-in Info field contains information relative to the teach-in message itself. The SLF field indicates the format of the security parameters in operation mode. The RLC is the current RLC in the transmitter. The KEY contains the private key used for encryption in operation mode. RLC and KEY may be encrypted with a pre-shared key

This secure teach-in message only transfers the security specific data. This message does not contain the information about how the data has to be interpreted (except for field type PTM) in operation mode.

To enable profile interpretation a profile-teach-in message (EEP or GP) has to be transmitted after the secure teach-in. This profile teach-in is conducted already secured (encrypted) using the decrypted key that the secure teach-in transmitted.

4.1.1 R-ORG TS

Secure teach-in R-ORG is 1-byte long with code 0x35.

4.1.2 TEACH-IN INFO

7	6	5	4	3	2	1	0
IDX		**CNT**		**PSK**	**TYPE**	**INFO**	
				0 - No PSK 1 – PSK	0 – No PTM 1 – PTM	Interpretation depends on Type. See 4.1.2.5	

Table 4. Teach-in info byte fields

4.1.2.1 Field IDX

A tech-in message can be divided in several telegrams. This field indicates the order of those telegrams. The first telegram receives the IDX = 0. In the case that the teach-in message is not divided then IDX= 0 also. More about chaining in A.2.2 chapter

4.1.2.2 Field CNT

If IDX = 0

CNT indicates the number of telegrams that the message is divided into

If IDX not 0

This field is reserved

4.1.2.3 Field PSK

If IDX = 0

0 The RLC and the KEY are not encrypted

1 The RLC and the KEY are encrypted with the pre-shared key. See 4.2

If IDX not 0

This field is reserved

4.1.2.4 Field TYPE

If IDX = 0

Indicates if the application is a PTM or not

1) Non-specific type
2) PTM

If IDX not 0

This field is reserved

4.1.2.5 Field INFO

The interpretation of this field depends on the code in Field (4.1.2.1)

If IDX = 0 and TYPE = 0

0 Unidirectional security teach-in procedure

1 Bidirectional teach-in procedure

If IDX = 0 and TYPE = 1

0 ROCKER A normal Teach In

1 ROCKER B normal Teach In

In other cases this field is reserved

4.1.3 SLF (Security level format)

With the information contained in the teach-in SLF byte the receiver is informed about the details of the secure messages in operation mode: what fields, how long they are, and what are the applied security algorithms.

7	6	5	4	3	2	1	0
RLC_ALGO		**RLC_TX**	**MAC_ALGO**		**DATA_ENC**		
0 – No RLC algorithm 1 – 16-bit x = x+1 2 – 24-bit x= x+1 3 – 32 bit x = x+1		0 - No 1 - Yes	0 – No MAC 1 – AES128 3 byte 2 – AES128 4 byte 3 – N/A		0 – No data encryption 1 – See chapter 6 2 – N/A 3 – VAES – AES128 4 – AES-CBC – AES128		

Table 5. Security Level Format fields and its interpretation. When RLC_ALGO is 1 or 2 the secure teach-in message contains the rolling code whose size corresponds to the one described by that bit field.

4.1.3.1 Field RLC_ALGO

Defines the type of the rolling code algorithm used during secure communication.

0 No RLC used in telegram or internally in memory
1 RLC= 2-byte long. RLC algorithm consists on incrementing in +1 the previous RLC value
2 RLC= 3-byte long. RLC algorithm consists on incrementing in +1 the previous RLC value
3 RLC= 4-byte long. RLC algorithm consists on incrementing in +1 the previous RLC value

4.1.3.2 Field RLC_TX

Indicates if the rolling code is transmitted or not in the secure operation mode

4 Secure operation mode telegrams do not contain RLC. A RLC could still be used internally for CMAC and DATA encryption as indicated by 4.1.3.1. In this case the initial RLC will be sent by the teach-in message (see 4.1.4)
5 Secure operation mode messages contain RLC

4.1.3.3 Field MAC_ALGO

Defines the type of algorithm for the CMAC calculation

0 No MAC included in the secure telegram
1 CMAC is a 3-byte-long code. MAC is calculated using AES128 as described in the 5.8 chapter
2 MAC is a 4-byte-long code. MAC is calculated using AES128 as described in the 5.8 chapter
3 N/A

4.1.3.4 Field DATA_ENC

Defines the type of algorithm for DATA encryption during secure communication in operation mode.

0 DATA not encrypted
1 Reserved. Not defined
2 Reserved. Not defined
3 DATA will be encrypted/decrypted XORing with a string obtained from an AES128. See 5.3
4 DATA will be encrypted/decrypted using the AES128 algorithm. See 5.1. and 5.2 sections

4.1.4 RLC

Contains the rolling code needed for the transmitter-receiver RLC synchronization. Only available if SLF defines it (RLC_ALGO not 0). See chapter 4.1.3.

4.1.5 KEY

Contains the private key used for the encryption of DATA and CMAC generation in operation mode.

4.2 Teach-in with pre-shared key

In case a teach-in is executed with a pre-shared key the fields RLC and KEY are encrypted. For the message structure of a teach-in please refer to the Figure 13.

The pre-shared key of the sender module must have been communicated to the receiver (a gateway) via serial interface in advance. The pre-shared key is typically written on a sticker on the sender module. The pre-shared key is not transmitted through the EnOcean air interface.

When TEACH_IN_INFO.PSK = 1 the fields RLC and KEY will be encrypted using the VAES encryption. See chapter 5.3 for details. The following parameters are used in the VAES algorithm:

- VAES DATA = concatenation of RLC + KEY bytes
- VAES RLC = 0x0000 (Rolling code is not exchanged at this moment, therefore it must be initialized to a default value)
- VAES PRIVATE KEY = PRE-SHARED KEY

4.2.1 PSK checksum

The PSK code (16 bytes) comes together with an extra byte, CRC checksum, which is used to verify that the installer/user writes correctly the 16-byte PSK code into the Device B (see Figure 12). The checksum uses a CRC8 algorithm that is explained in chapter A.4.

5 Security algorithms

This section describes the security algorithms for the EHW that are used to secure the telegram content.

5.1 AES128 encryption

DATA can be encrypted using the standard high-security AES128[(1),(2)] algorithm with cipher-block chaining (CBC)[(5)]. Constant data will result in constant encrypted information.

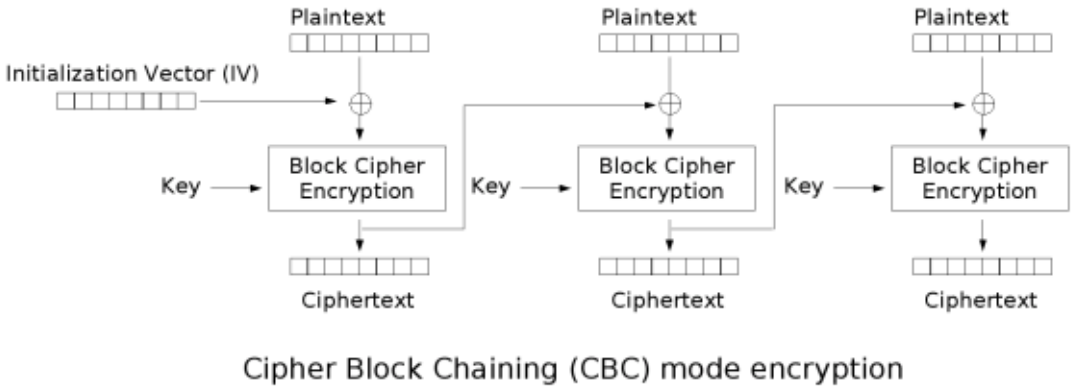


Figure 14. AES128 schematic diagram flux. DATA to be encrypted is called “plaintext” in the figure. The plaintext bytes are divided in chunks of 16 bytes. The first 16-byte array is the plaintext array to the left. If the last piece of plaintext is not 16-byte long the bit sequence 10..0 will be added to complete 16 bytes. The initialization vector has all bytes set to 0. The Block Cipher Encryption block uses the standard AES128 algorithm (1). The 16-byte long key is the same for all AES operation blocks.

AES128 algorithm with cipher-block chaining is only used with chained EnOcean telegrams, because all data blocks transmitted have to be divided in 16-Byte blocks.

Each message sent, consisting of chained telegrams, will start from the initialization vector.

5.2 AES128 decryption

DATA encrypted using the method described in section AES128 encryption can be decrypted using the standard high-security AES128[(1)] algorithm for decryption with CBC[(5)].

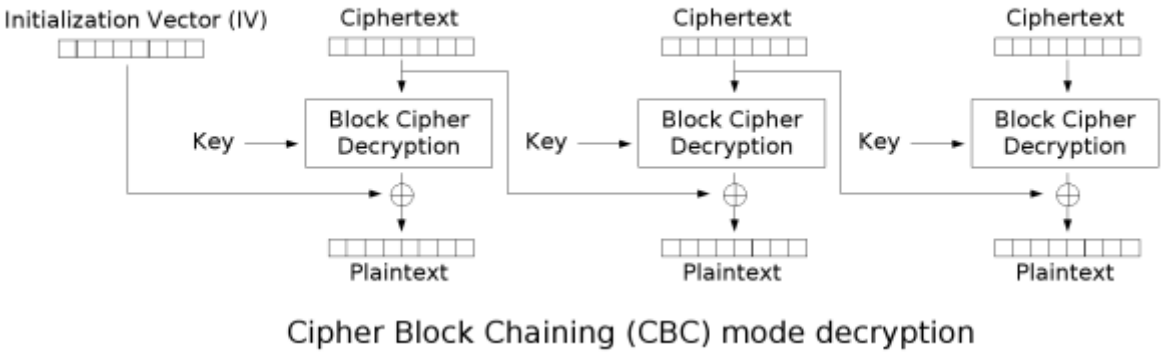


Figure 15. AES128 schematic diagram flux. DATA to be decrypted (ciphertext) is divided in 16-byte long arrays. PRIVATE KEY is 16-byte long. The algorithm returns a 16-byte DATA encrypted (plaintext) operating as indicated in the figure.

The application must eliminate the padding bytes in case these were inserted in the plaintext when encryption took place.

Usually CBC will be used over more messages. In Security of EnOcean networks, the initialization vector will be applied at each new message (which could consist of more EnOcean telegrams).

5.3 VAES (variable AES) encryption

A more secure way to encrypt data is using the 128-bit AES algorithm[(1)] in combination with the rolling code. The mechanism described here allows DATA lengths from 1 to N, where N does not have to be necessarily a multiple of 16. Thanks to the rolling code the generated encrypted code varies, although the DATA input is constant. This feature improves the obscurity of the original information and avoids replay attacks. Encryption is done in 16 bytes chunks, similar to the previous algorithm. But the length of the resulting cipher text is the same as the length of plain text.

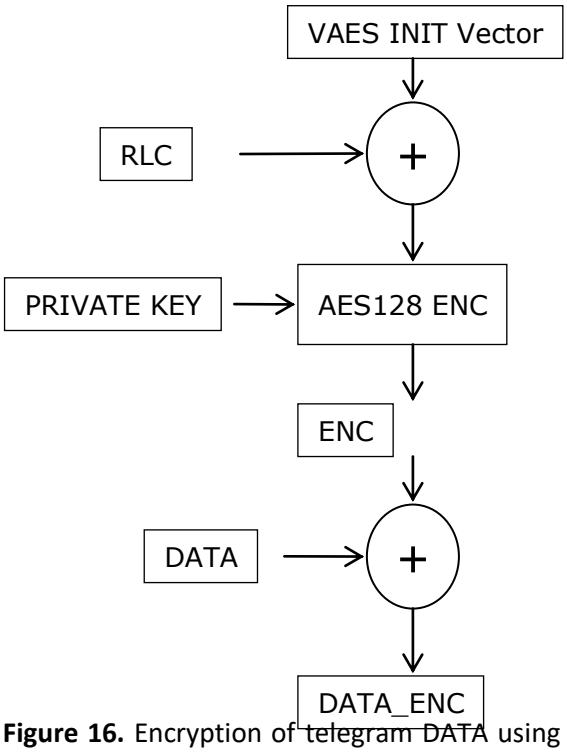


Figure 16. Encryption of telegram DATA using variable XOR AES128 for DATA equal or less than 16 bytes. The AES128 is used to perform a pseudo random generator. The plus symbol within a circle represents a bitwise XOR operation. One DATA chunk can have a maximum of 16 bytes. The length of DATA_ENC is equal to the length of DATA. VAES INIT Vector is a constant 16-byte hexadecimal string array. The RLC is XORed with the VAES INIT Vector before entering the AES128 algorithm. The VAES INIT VECTOR byte array first element, VEAS_INIT_VECTOR [0], is XORed with the RLC most significant byte. VEAS_INIT_VECTOR [1] is XORed with the RLC 2nd most significant byte and so on. The RLC is filled with 00..0 until 16 bytes. In the same way DATA and ENC arrays are XORed to produce the output code.

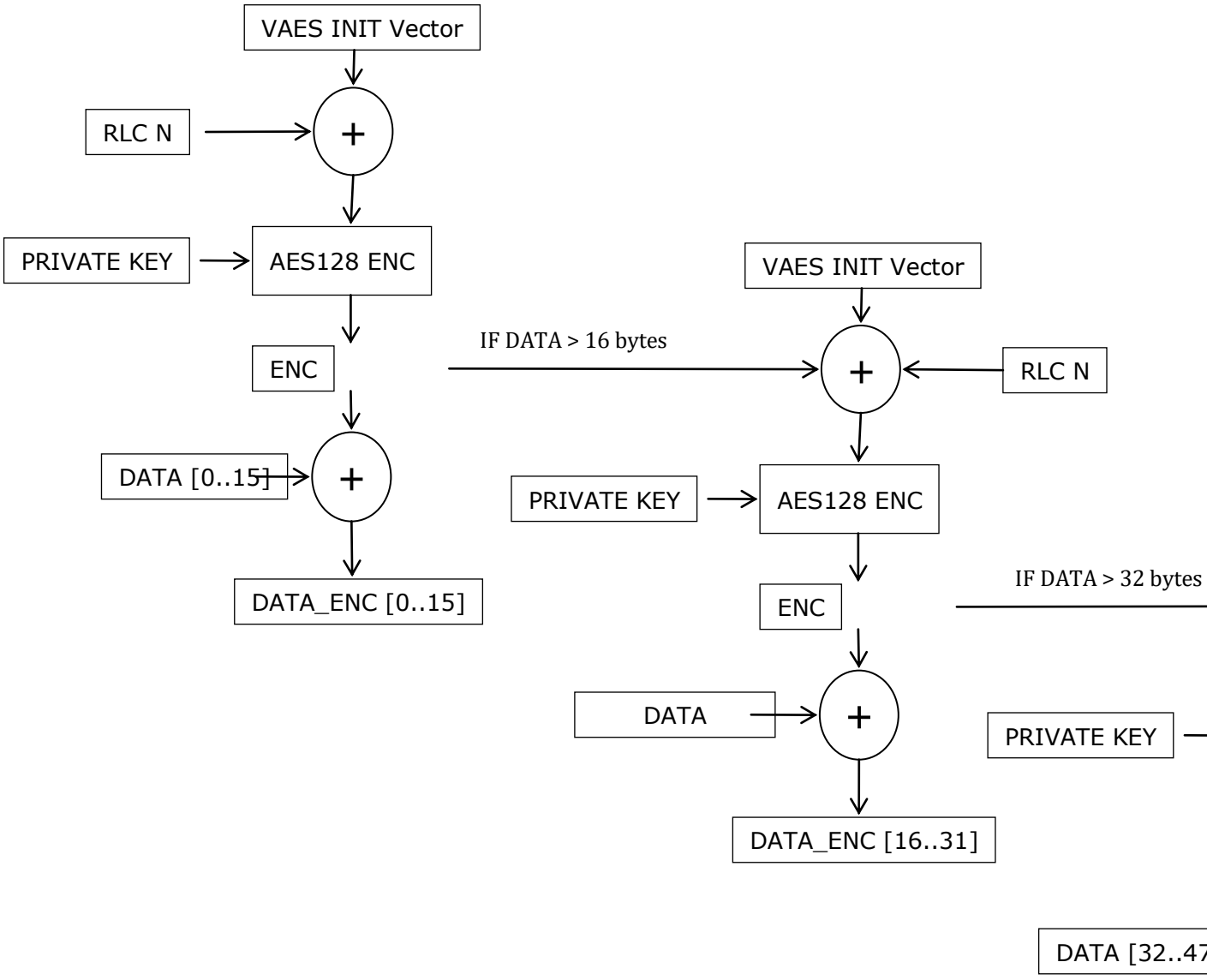


Figure 17. Encryption of telegram DATA using variable XOR AES128 for DATA longer than 16 bytes. Encryption is repeated but in all following cycles the ENC field from previous cycle is XORed with the encryption input for the actual cycle. The Rolling Code is the same for all cycles. By using the ENC field in all following cycles the encryption input will change and so the ENC field for DATA. Otherwise same rules apply as in encryption with less than 16 bytes.

5.4 VAES (variable AES) decryption

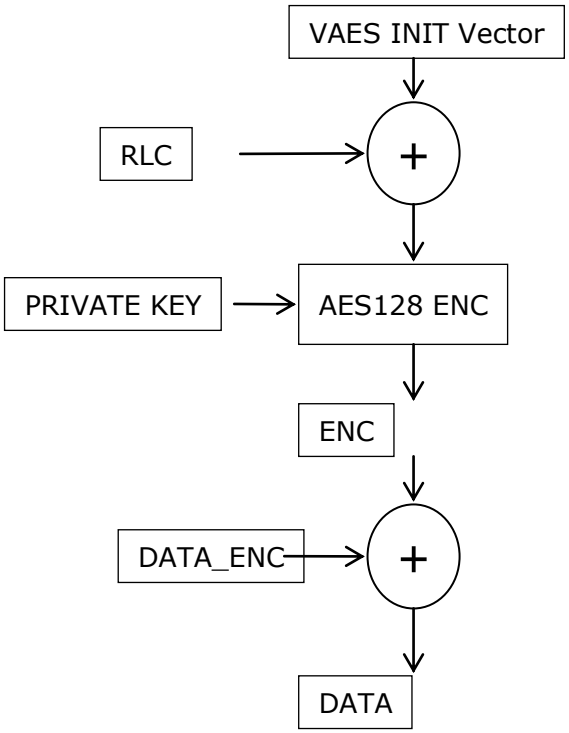


Figure 18. Decryption algorithm of the received telegram. DATA_ENC is the received encrypted DATA. DATA is less than 16 bytes. Note: The AES algorithm required is the same as in the encryption algorithm. The RLC value in the receiver must be the same as in the transmitter module. Otherwise the decrypted DATA will not correspond to the DATA in the transmitter. The DATA has the same length as the DATA_ENC. The XOR bit alignment follows the ideas described in Figure 16.

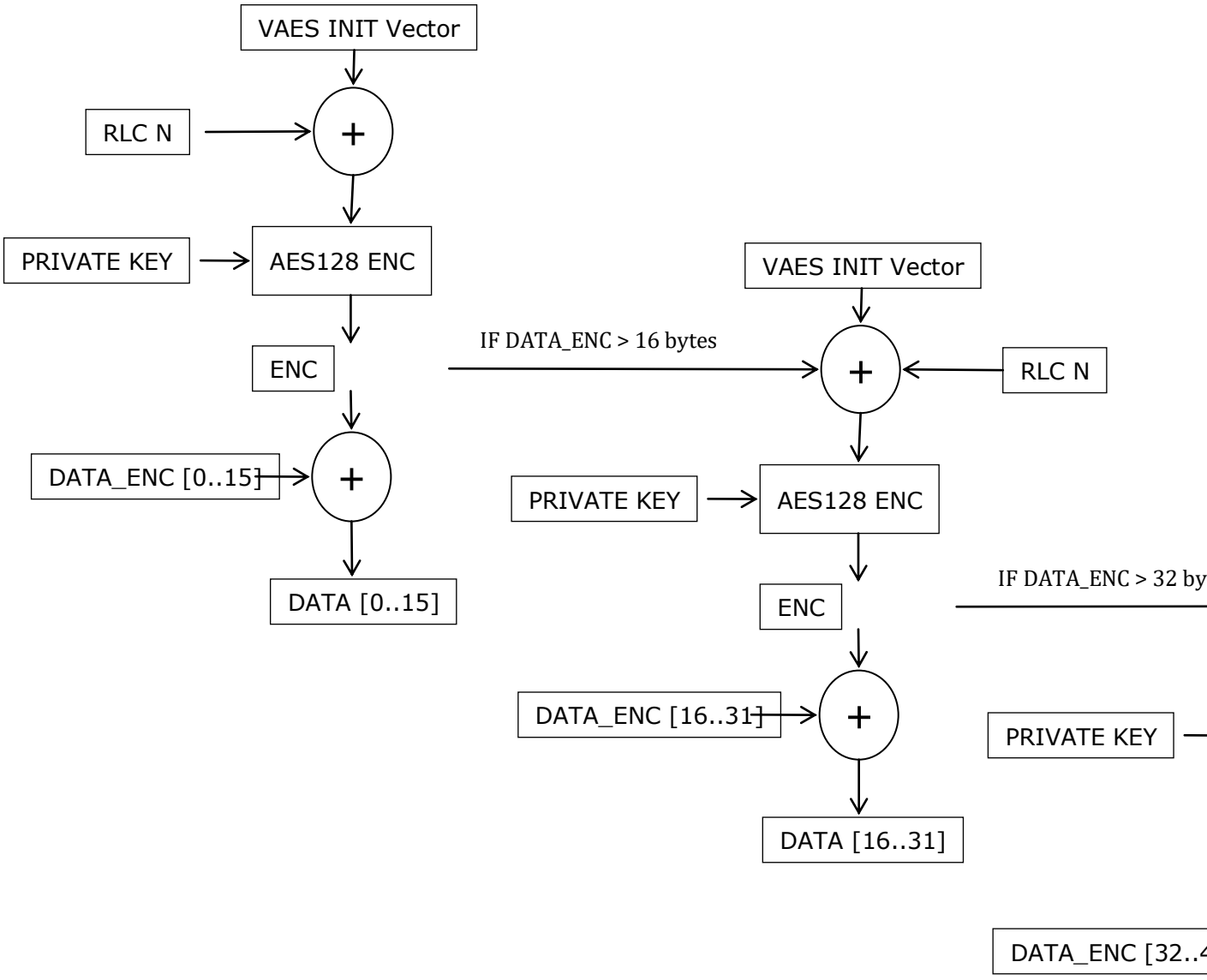


Figure 19. Decryption algorithm of the received telegram. DATA_ENC is the received encrypted DATA from the transmitter. DATA is more than 16 bytes.

For this algorithm to work correctly the receiver must check first that its internal RLC is correct. This can be done by comparing the internal RLC against the telegram RLC or, if there is no transmitted RLC, by comparing the MAC (see 5.8).

5.5 VAES INIT Vector

The VAES INIT Vector is a known constant value used as input to the AES algorithm.

Its value is the hex **3410de8f1aba3eff9f5a117172eacabd.**

Seen as a 16-byte array, the VEAS_INIT_VECTOR first element, VEAS_INIT_VECTOR [0] = 0x34.

The array last element is VEAS_INIT_VECTOR [15] = 0xbd.

5.6 Private Key

The private key is a secret constant value used as input to the AES algorithm. Its length is 128 bits (16 bytes).

5.7 Rolling Code

The rolling code is a value stored in the sender and transmitter module independently. The code changes for every sent/received telegram according to a predefined algorithm. The rolling code is used as to obtain different CMAC codes (5.8) and VAES encryption values (5.3) although telegram DATA remains constant.

Explicit RLC strategy - The RLC may be transmitted as part of the telegram information to speed-up the MAC calculation in the receiver, at expense of increasing the energy spent in radio transmission and but improving the application use cases.

Implicit RLC strategy - It is possible for the transmitter module not to send RLC in the telegram. (See 5.8) to decrease to needed energy.

If the RLC is transmitted or not is indicated by the SLF (Security level format)

The rolling code algorithm is the simplest possible: it is a **counter that is incremented by 1** when the transmitter/receiver sends/receives a telegram. When the code rolls-over it starts at 0 again.

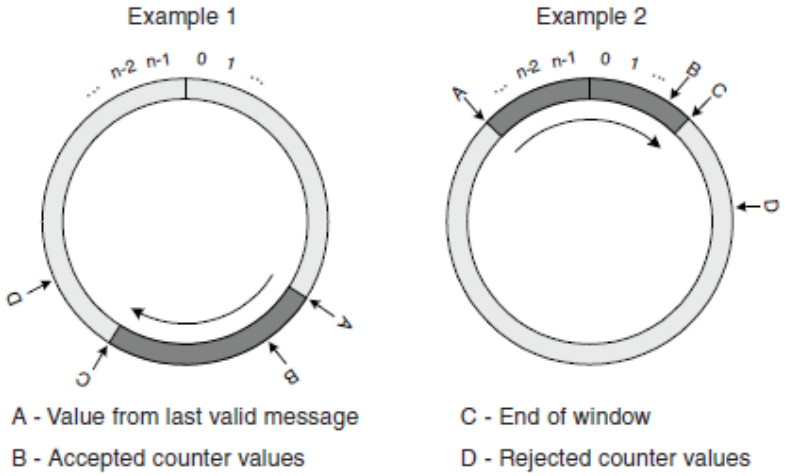


Figure 20. Rolling Window of Acceptance for Counter Values *(AVR411: Secure Rolling Code Algorithm)*.

The RLC window is a mechanism that ensures that even if the transmitter and receiver lost their synchronization (transmitter was operated outside the range of the receiver or telegrams where lost), telegrams will be still accepted. The difference between the rolling code counters in the sender and receiver must not be bigger than the rolling code window size. When the receiver module receives a wrong RLC it tests the next rolling code. If this test fails again, it tests the next rolling code.

These tests continue until the RLC match, or a number of RLC window size tests were performed. In this last case the receiver returns to its original RLC and rejects the telegram.

Once the receiver cannot match the transmitter rolling code in the range of the window size it may block the TXID of the transmitter.

If sender and receiver RLC were desynchronized with a RLC difference bigger than the RLC window size, they can be synchronized again by means of the teach-in procedure (section 4).

The window size for incorrect RLC has a recommended value of 128. Other values are also applicable.

5.8 CMAC algorithm

The general algorithm (see [3]) to calculate the CMAC of a message to send is the following.

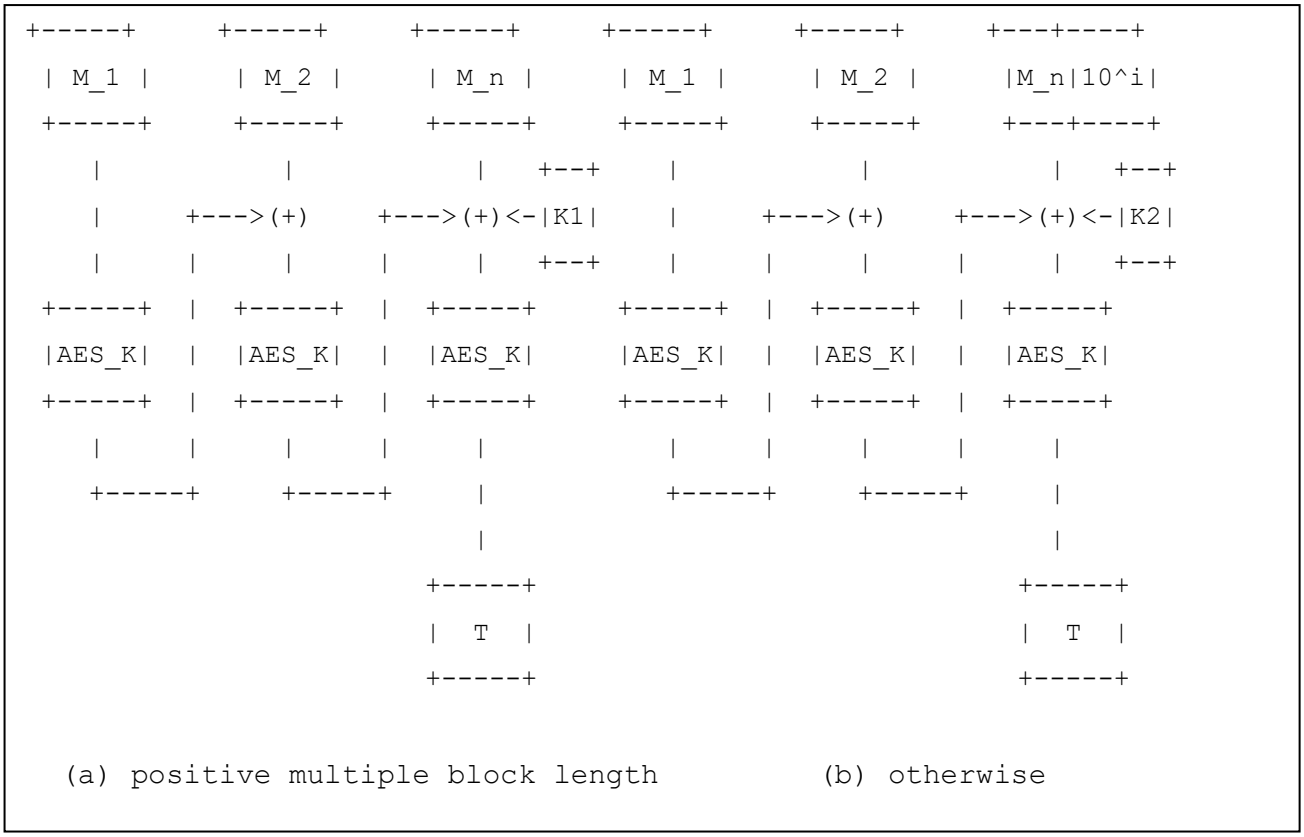


Figure 21. The case (a) applies for messages that are multiple of 16 bytes. Subkey K1 is used in this case. See 5.8.2 for details about the subkey generation. The case (b) applies for messages that are not multiple of 16 bytes. Subkey K2 is used then.

M_i are 16-byte long array of the message bytes whose CMAC is to be calculated. M_1..M_n-1 are 16-byte long arrays. In case of (b) the last byte of the message is concatenated with the bit sequence 10…0, to obtain a 16 byte-long array.

The AES_K uses as input the 16-byte M_i array and the K key.

The last 16-byte M_i array requires the subkey K1 –case (a)- or the subkey K2 -case (b).

The message is composed of the RORG-S field, the DATA field and optionally the RLC (in case it is being used explicitly in the telegram or implicitly).

5.8.1 CMAC calculation for operation mode telegrams

To illustrate the CMAC calculation a message <=16 bytes will be used:

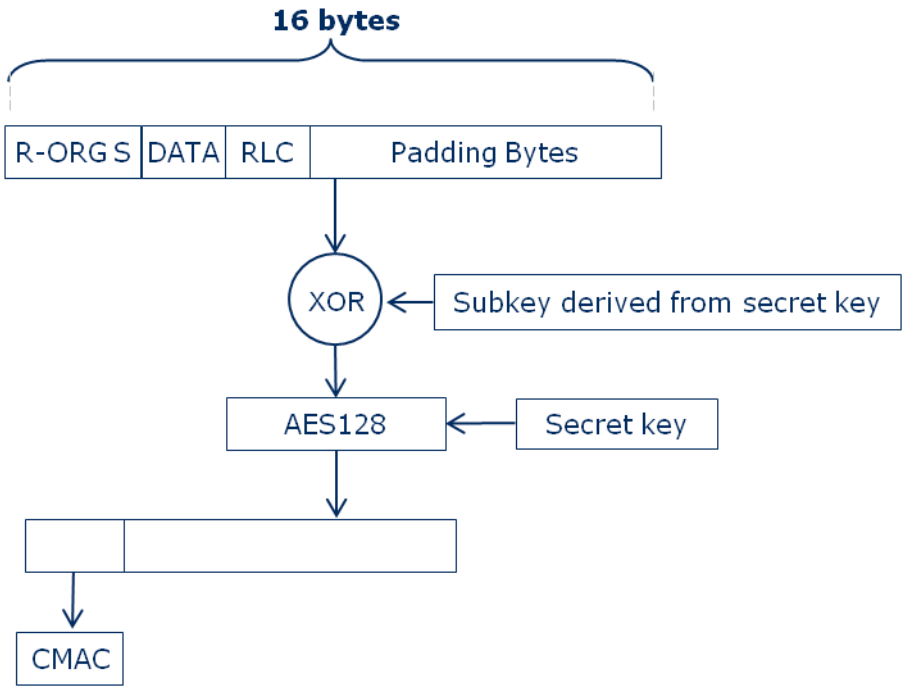


Figure 22. CMAC generation algorithm for a message <= 16 bytes. The R-ORG-S is the secure radio message R-ORG code. DATA is the secure message DATA field. When R-ORG S= 0x31, DATA includes the encrypted non-secure R-ORG as most significant byte. When R-ORG S= 0x30, DATA does not include the encrypted non-secure R-ORG. If no RLC is used (either in the message or internally) the field RLC is not included for the CMAC calculation. IMPORTANT: if the RLC is used but not transmitted with the telegram then the RLC is included in the CMAC calculation. In this way the CMAC changes for each transmitted message. From the AES-generated array the bytes with index 0, 1,… (CMAC_size- 1) are taken as CMAC. The padding bytes are only necessary for the case that the input information is < 16 bytes.

When the message fields (R-ORG S, DATA -and optionally RLC-) does not add 16 bytes for applying the AES128 algorithm, a bit sequence 100…0 (padding bits) is concatenated to form a total bit string of 16 bytes.

The R-ORG S, DATA –optionally RLC and PADDING BITS- are XORed with the 16-byte *Subkey derived from private key*. R-ORG S XORs Subkey_der [0] and so on.

If the R-ORG S + DATA + RLC fields add 16 bytes then the subkey K1 will be used.

Otherwise the subkey K2 is used.

When performing AES, the byte R-ORG S corresponds to the string array byte with index 0.

A four byte CMAC corresponds to the string array bytes 0 (MSB), 1, 2, 3 (LSB) generated by the AES128 algorithm. A three byte CMAC corresponds to the string array bytes 0(MSB), 1, 2 (LSB) generated by the AES128 algorithm.

Note: When the R-ORG S + DATA + RLC > 16 byte the message has to be spitted like indicated in

Figure 21. The R-ORG will only be present as first byte in the M_1 while the RLC will only be present only within M_n.

5.8.2 Calculation of the subkey (AES-CMAC, RFC4493)

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                    Algorithm Generate_Subkey                         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                      +
+   Input    : K (128-bit key)                                         +
+   Output   : K1 (128-bit first subkey)                               +
+              K2 (128-bit second subkey)                              +
+----------------------------------------------------------------------+
+                                                                      +
+   Constants: const_Zero is 0x00000000000000000000000000000000       +
+              const_Rb   is 0x00000000000000000000000000000087       +
+   Variables: L          for output of AES-128 applied to 0^128      +
+                                                                      +
+   Step 1.  L := AES-128(K, const_Zero);                              +
+   Step 2.  if MSB(L) is equal to 0                                   +
+            then    K1 := L << 1;                                     +
+            else    K1 := (L << 1) XOR const_Rb;                      +
+   Step 3.  if MSB(K1) is equal to 0                                  +
+            then    K2 := K1 << 1;                                    +
+            else    K2 := (K1 << 1) XOR const_Rb;                     +
+   Step 4.  return  K2;                                               +
+                                                                      +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Algorithm Generate_Subkey

- In step 1, AES-128 with key K is applied to an all-zero input block
- In step 2, K1 is derived through the following operation:
  - If the most significant bit of L is equal to 0, K1 is the left-shift of L by 1 bit
  - Otherwise, K1 is the exclusive-OR of const_Rb and the left-shift of L by 1 bit
- In step 3, K2 is derived through the following operation:
  - If the most significant bit of K1 is equal to 0, K2 is the left-shift of K1 by 1 bit
  - Otherwise, K2 is the exclusive-OR of const_Rb and the left-shift of K1 by 1 bit.
- In the case that padding bits where necessary use the K2 as subkey. Otherwise K1

6 EnOcean High Security

EnOcean High security is an extension of the existing security concept. Its main aim is to:

- Enable protection against special man-in-middle attack (relay attack)
- Enable more flexible handling for devices communicating bidirectional with encryption

To handle first requirement is it suitable to apply the challenge & response process with time limited authentication. This implies that both communication devices are bidirectional. All the following definitions are based on that constrain.

6.1 Use Case definition

High security specification is developed for specific designed applications where this kind of security is required (e.g. doorlock, access control, safety applications). It is not the baseline for security application but the highest grade and should be applied only in applications where it is a valid use case.

For high Security these use cases are defined:

1. Bidirectional data flow – both devices are line-powered

   Both devices have enough computing capabilities and energy to perform security relevant tasks. Data telegrams are exchanged in both ways e.g. communication between smart plug and gateway.

2. Bidirectional data flow – one device is energy autarkic and one is line-powered

   One device is doing energy harvesting and the second is line-powered. Data flow is executed in both directions. The autarkic device initializes the communication and then expects a response from the line-powered device i.e. Smart Acknowledge process.

3. Unidirectional data flow (both devices have bidirectional capabilities) – one device is energy autarkic and one is line-powered

   One device is energy harvesting and the second is line-powered. Data flow is executed in one direction – from the autarkic device to the line-powered. Bidirectionality is only required to execute high security features not for actual application use case. In this case Smart Acknowledge will be used too.

6.2 Protection against Relay Attacks

Relay attacks are special man-in-the-middle attack scenarios. The intruder intercepts the communication and blocks the targeted receiver. Then he can replay the communication to a later moment. With this scenario he will even bypass the RLC Counter checking, because the receiver has not updated his counter and so accepts the replayed message.

To prevent this attack scenario the message validity must be time limited and/or the receiver must be aware of transmission process ongoing. Time limitations are ensured by performing an authentication by challenge and response which valid only limited time (500 ms). During challenge and response a Nonce is used as challenge.

6.2.1 Authentication based on CMAC

Authentication can be:

- Unilateral – only one of the communication partners is authenticated and his outgoing communication is protected against replay attacks – one Nonce is exchanged
- Mutual – both communication partners are authenticated and both communication ways are protected against replay attacks – two Nonce are exchanged

For CMAC computing the EnOcean security concept uses the Payload of telegrams. The Nonce is used during CMAC computing too and so ensures that the Nonce is connected with the exchanged message and its data content. So becomes the data content also valid for limited time.

EnOcean Security concept uses the VAES for data encryption. The Nonce can be also used for the initialization vector for the VAES process. This way a random element is added to the VAES process. This is required if Nonce is a random number and no RLC is used.

The Nonce can be:

- Used during CMAC counting
- Used as initialization vector for VAES counting

Nonce represents the challenge and has to be therefore exchanged between the communication partners via air interface. The Challenge and Response does not have to encrypt and can be transferred plaintext. The CMAC algorithm represents the securing element.

During data communication following constrains are applied:

- Bidirectional communication can be only executed after mutual authentication. Both parties can trigger the authentication
- Unidirectional communication is unilateral authenticated. The emitter of the data flow is authenticated and only the emitter can trigger the communication. The challenge is provided by the consumer of the data

For Nonce we can use:

- Random number 32 bit – here is critical that the generator process is not predictable, does not repeat same sequences and is equally spread on the defined range
- Simple incrementing, non-repeating sequence 32-bit number

6.2.2 Communication scenarios

To protect against relay attacks we define three new security communication approaches. They differ in aspect of the Use Case definition made in chapter 6.1:

1. Mutual authentication with RND as NONCE
2. Unilateral authentication with RND as NONCE

In following definitions the DEVICE A is the sensor / autarkic device which initialize the communication. DEVICE B is the line-powered device which is receiving the communication. The definitions also apply for Use Case 1, then DEVICE A is also an line powered device.

The link between A to B is unique, because of used keys. No other device can listen and decrypt the communication. The communication is unicast – and has to be eventually addressed in one or both directions, depending if device A or device B has other communication partners.

Device A

KEY A – key used by device A to encrypt communication, used for communication from A to B

RND A – random number generated by device A - 32 bits

TIMEOUT A – timeout on device A till next message from device B should be received

Device B

KEY B – key used by device B to encrypt communication, used in for communication from B to A

RND B – random number generated by device B - 32 bits

TIMEOUT B – timeout on device B till next message from device A should be received

Fields

In the following description we use these fields:

- ENC: Encryption method - NA / VAES
- VAES IntV: Content of the Initialization Vector of VAES, for Details see 5.3.
- PAYLOAD: Content of the data field used in the telegram
- CMAC: Content which is the CMAC calculated on – complete expiation see 6.3.3.1
- RORG-MS Telegram payload length – in case of MS Telegram is used, we specify the length.

6.2.2.1 Mutual authentication with RND as NONCE

In this scenario the NONCE is represented by random sequences. No RLC is used. This solution might be useable for autarkic devices.

Failures like timeouts are not considered. Only the ideal case is described. On any failure e.g. CMAC not validated or timeouts the whole process is discarded and needs to start from beginning or other measures e.g. resynchronize must be taken. The communication will be executed in these steps.

1. A sends Challenge to B. and starts timeout A.

Message parameters:

- ENC: NA
- RORG: RORG-MS
- DATA: RORG-MS Header (RND), RND A
- RORG-MS telegram payload length = 5 bytes

2. B receives Challenge. Sends Response with next Challenge to A. B starts communication timeout B.

Message parameters:

- ENC: NA
- RORG: RORG-MS
- DATA: RORG-MS Header (DATA / RND / CMAC), RND B
- CMAC: DATA, RND A
- RORG-MS telegram payload length = 9 bytes

3. A receives Challenge and successfully authenticates it with help of both RNDs. A sends Response with payload data to B. Optionally - Starts timeout A.

Message parameters:

- ENC: VAES
- VAES INT: VAES INIT Vector + RND B
- RORG: RORG-SEC
- DATA: application payload
- CMAC: DATA, RND B, RND A

4. B receives Response with payload within timeout B. Validates it with help of both RNDs. Optionally – continues with communication and send another payload message.

Message parameters:

- ENC: VAES
- VAES INT: VAES INIT Vector + RND A
- RORG: RORG-SEC
- DATA: application payload
- CMAC: DATA, RND A, RND B

5. Optionally A receives message from B within timeout A.

No further communication with same RND A and RND B can be executed, because no Sequence number is present and replay attacks could be executed. To transmit bigger data quantities at once use message chaining.

Please see the process description in the sequence diagram below.

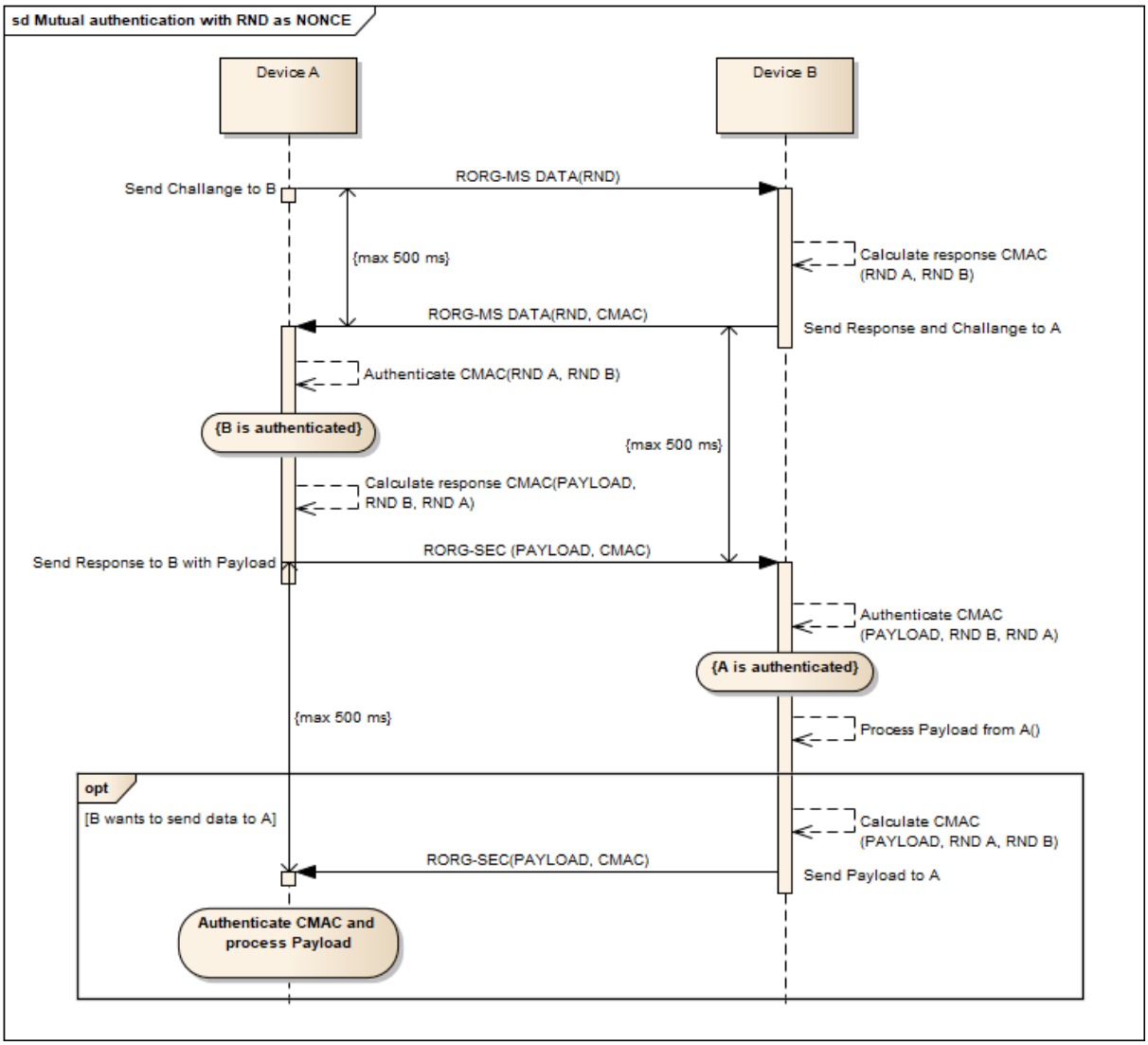


Figure 23. Mutual authentication with RND as NONCE.

SLF:

- DATA ENC – VAES + RND

6.2.2.2 Unilateral authentication with RND as NONCE

In this scenario the HASH is represented by one random sequence. No RLC is used. This is the solution which requires the least amount of resources and ensures relay attack protection.

Failures like timeouts are not considered. Only the ideal case is described. On any failure CMAC not validated or timeouts the whole process is discarded and needs to start from beginning or other measures e.g. resynchronize must be taken. The communication will be executed in these steps.

1. A sends Request for Communication Message to B and start timeout A.

Message parameters:

- ENC: NA

- RORG: RORG-MS
- DATA: RORG-MS Header 0x00
- Payload length = 1 byte

2. B receives Request for Communication. Sends Challenge to A. B starts communication timeout B.

Message parameters:

- ENC: NONE
- RORG: RORG-MS
- DATA: RORG-MS Header (RND) + RND B
- Payload length = 5 bytes

3. A receives Challenge and successfully decodes it. A sends response with payload data to B. Starts timeout A.

Message parameters:

- ENC: VAES
- VAES INT: VAES INIT Vector + RND B
- RORG: RORG-SEC
- DATA: application payload
- CMAC: DATA + RND B

4. B receives message within timeout B. Decodes the message and authenticates with help of RND B.

No further communication with same RND B can be executed, because RLC is not present and replay attacks could be executed. To transmit bigger data quantities at once use message chaining.

Please see process description in the sequence diagram below.

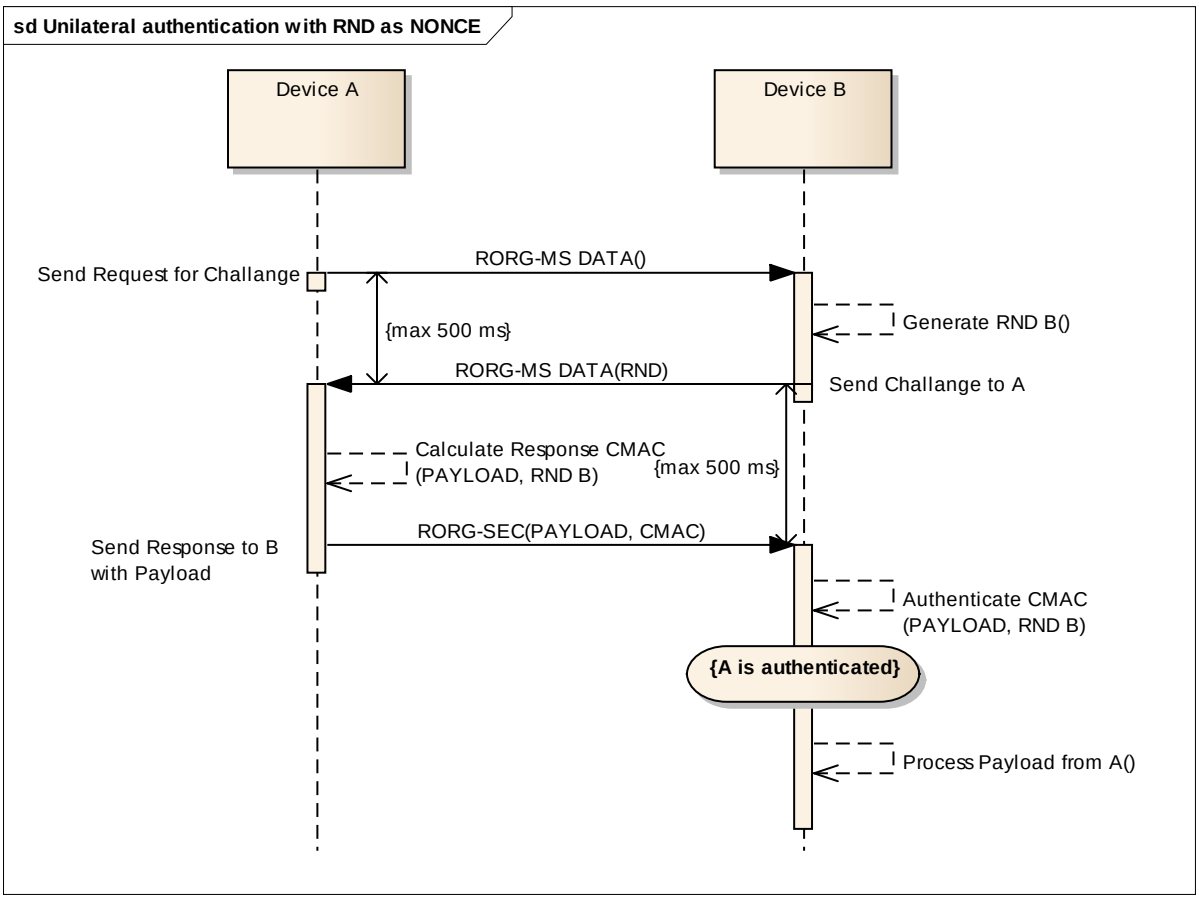


Figure 24. Unilateral authentication with RND as NONCE.

6.2.2.3 Error scenarios

If the message gets lost the process is considered as failed.

Repeated transmissions of request for challenge or challenge messages between identical communication partners shall restart the authentication process and cancel any previous ongoing validation.

6.3 High Security inclusion into existing concept

6.3.1 Security Level Format extension

Which High Security communication concept is used during data communications needs to be communicated at teach in time. For this purpose the defined SLF is extended. Following extension has been made:

7	6	5	4	3	2	1	0
RLC_ALGO		**RLC_TX**	**MAC_ALGO**		**DATA_ENC**		
0 – No RLC algorithm 1 – 16-bit x = (x+1) 2 – 24-bit x= x+1 3 – 32-bit x=x+1		0 - No 1 - Yes	0 – No MAC 1 – AES128 3 byte 2 – AES128 4 byte 3 –		0 – No data encryption 1 – VEAS – AES128 - RND 2 – N/A 3 – VAES – AES128 - RLC 4 – AES-CBC – AES128		

Table 6. Security Level Format extension fields and its interpretation.

DATA_ENC – Value: 0x1

VAES – AES128 – RND is used as initialization vector instead of RLC i.e. Mutual authentication with RND as NONCE, Unilateral authentication with RND as NONCE.

If secure teach in is bidirectional or not is already defined in INFO field – see chapter 4.1.2.

6.3.2 Meta Security Telegram

The Meta Security Telegram is required to support additional task required in high security. Following message structure is defined.

Figure 25. Meta Security Telegram structure.

RORG-MS

Meta Security telegrams are identified by the code 0x36.

Header

The RORG-MS telegram has a header which expresses what content of the meta-security information is carried. This header is always present at the first place.

7	6	5	4	3	2	1	0

NA	NA	NA	NA	NA	NA	RND Included	CMAC Included
						0: RND not included 1: RND is included	0: CMAC not included 1: CMAC is included

Table 7. Meta Security Telegram header fields and its interpretation.

RND - Random number used as challenge – if included.

CMAC - CMAC is included, used as response for authentication – if included.

6.3.3 Algorithm extension

6.3.3.1 CMAC

The CMAC is altered for data communication and for meta security telegram.

Following fields are added:

- SOURCE ID - Also the LSB 32-bit of the communication ID of the originating device
- RND 1 – 32 bit, first included random number
- RND 2 – 32 bit, second included random number (optional, depending on scenario i.e. Mutual or unilateral)

RND 1, RND 2 – can be represented by RND A, or RND B. The order is specified and is important to consider. Please see communication scenarios chapter 6.2.2 for details on order and content.

All fields are included only if required. Please find meta security header definition chapter 6.3.2 and communication scenarios chapter 6.2.2 for details and contend of the telegrams.

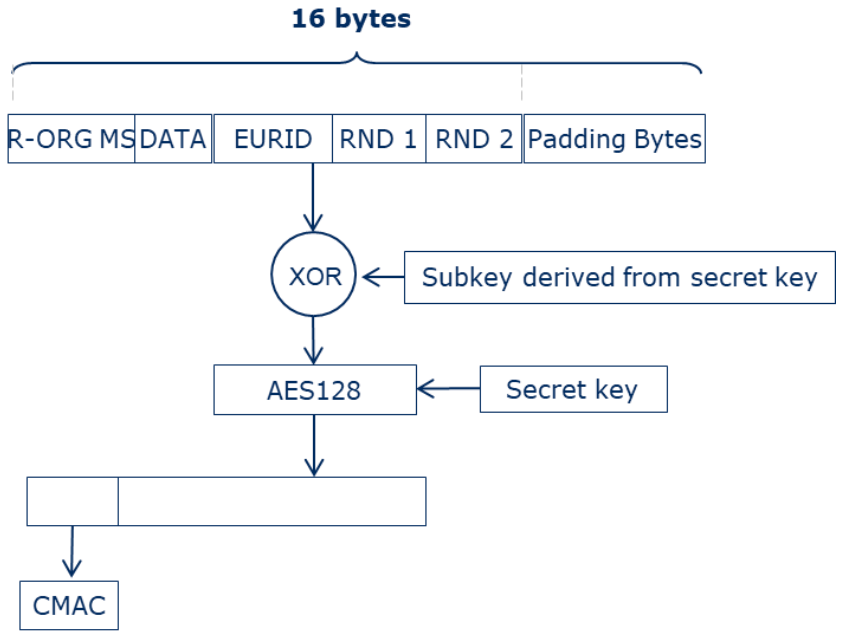


Figure 26. Graphical representation of altered CMAC for Meta Security Telegram

6.3.3.2 VAES

The VAES process has to be extended to use RND during processing of VEAS INIT Vector. The communication scenarios in chapter 6.2.2 are defined to RND as initialisation vector.

The RND is 32 bit long so the XOR operation needs to be extended to uses all 32 bits.

The VAES INIT VECTOR byte array first element, VEAS_INIT_VECTOR[0], is XORed with the RND most significant byte. VEAS_INIT_VECTOR[1] is XORed with the RND 2nd most significant byte and so on.

Please see graphical representation bellow.

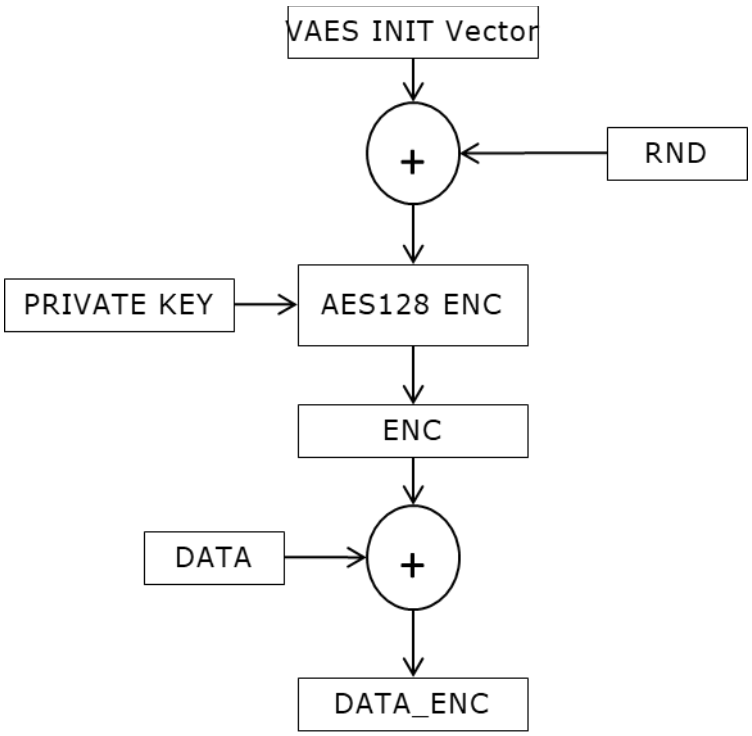
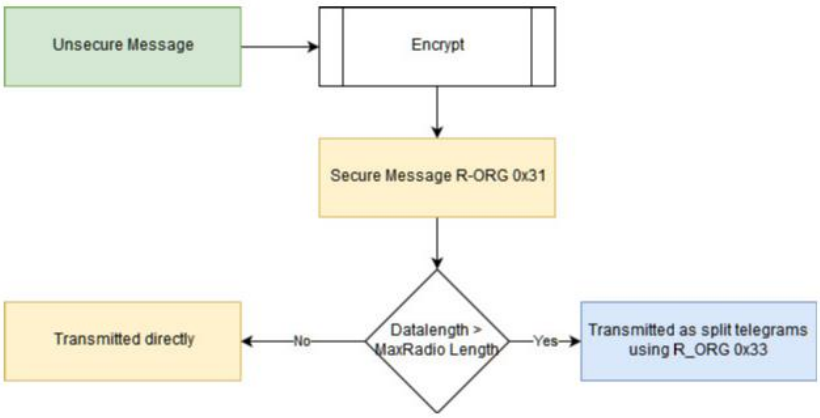


Figure 27. Graphical representation of VAES extended process.

7 SEC_CDM – 0x33 chained secure messages

Secure chaining is an add-on for the ERP chaining mechanism. The core functionality is the same as for Chained messages and the operation key and SLF will be used for transmitting secure chained messages. The message will be first encrypted like a normal telegram with encapsulated R-ORG and the SLF of the device. The encryption function uses the 0x31 R-ORG internally for the CMAC calculation. After the encryption using the 0x31 R-RORG was successful and the data length is longer than the maximum transmittable length. The message will be chained using the SEC_CDM.
This has the advanced that only once the overhead of transmitting the CMAC and RLC is added to the transmission of the split telegrams and only one RORG encapsulation is happening.



As most devices are embedded devices with limited memory capacity, it is highly recommended that only one chained message is send at a time. (To or from a device).

SEC_CDM	SEQ	IDX	data field	EURID	status	checksum
1 byte	2 bit	6 bit	N bytes. Depend if addressed or not And protocol	4 bytes	1 byte	1 byte
	1 byte					

Table 8: Radio telegram structure of SECURE chained telegram – NOT ADDRESSED.

Reminder from CDM:

A message consists of a number of chained telegrams. Each telegram in a chain has a sequence number and an index. The sequence number is the same for each message in the same chain and indicates that the messages belong to the same chain. It is increased with every new CDM and cannot be equal to 0.

The index starts from 0 and is incremented with each message in a chain. The purpose of the index is to indicate the correct order of the messages. The first message of a chain contains the total length of the payload of the message.

A full working example can be found in the appendix A.5.3

Example for the IDX 0 and ERP1: the following data is in the data field:

SEC_ CDM	SEQ	IDX	data length	RORG_EN	Payload	EURID	status	checksum
1 byte	2 bit	6 bit	2 Bytes	1 byte	10 bytes	4 bytes	1 byte	1 byte
	1 byte							
0x33	IDX= 0		Unencrypted, clear text. Data length = RORG+ Payload + RLC+CMAC length	Encapsulated RORG	Payload			

For the following index which are not the last telegram

SEC_ CDM	SEQ	IDX	Payload	EURID	status	checksum
1 byte	2 bit	6 bit	<= 13bytes	4 bytes	1 byte	1 byte
	1 byte					
0x33	IDX= x					

For the last telegram:

SEC_ CDM	SEQ	IDX	data field	RLC	CMAC	EURID	status	checksum
1 byte	2 bit	6 bit	Last part of the payload data	0,2 or 3 bytes	0, 3,4 bytes	4 bytes	1 byte	1 byte
	1 byte							
0x33	IDX=last			Depend on SLF	Depend on SLF			

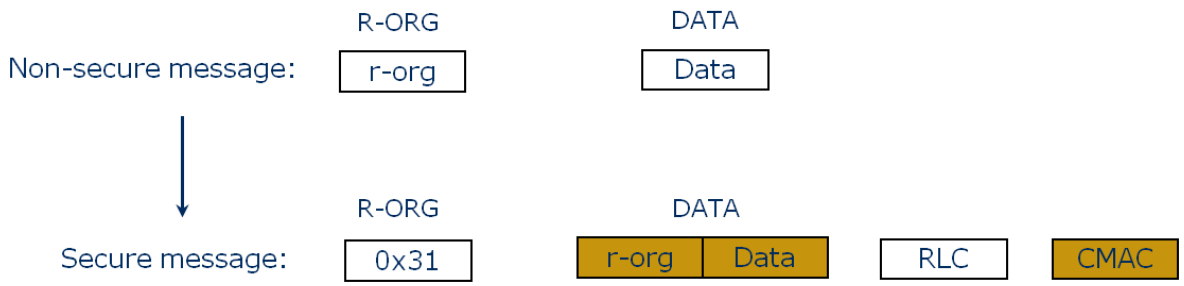


Figure 28. Reminding of “Transforming an unsecure message into a secure message with R-ORG encapsulation” Section 3.2.1. The secured message “0x31” is than transformed while sending into the 0x33 message

8 Referenced documents

(1) NIST,FIPS 197, *Advanced encryption standard (AES)*, 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

(2) - Zabala, *Rijndael Cipher, 128-bit version (data block and key)*
http://www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf

(3) - JH. Song, R. Poovendran, J. Lee, T. Iwata, 2006, *The AES-CMAC Algorithm*
http://www.rfc-editor.org/rfc/rfc4493.txt

(4) - *AVR411: Secure Rolling Code Algorithm*.
http://www.atmel.com/Images/doc2600.pdf

(5) – Wikipedia, *Block cipher modes of operation*.
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Annex A Appendix

A.1 Application recommendations

This chapter shall provide best practice definitions based on field experience reflecting most applicable combination of security parameters (e.g. SLF) for final applications.

A.1.1 Rolling code

Rolling code is the key element in data protection and validation. Following definitions shall be adapted:

The RLC as plain information (not encrypted) can be transmitted explicitly inside the telegram without corrupting the data encryption / authentication or increasing the risk of success of potential intruder attacks.

The RLC shall be 32-bit sequence number anywhere possible. Size of 24-bit is not recommended and shall be used only in applications transmitting low amount of telegrams. Size of 16-bit is considered as depreciated and shall not be used in any applications.

A.1.2 Best case definition

Best case scenario applies mostly for applications which are:

- Line powered
- Battery powered
- Solar powered
- Thermal powered
- Partly kinetic powered

Best practice shall use this SLF format:

7	6	5	4	3	2	1	0
RLC_ALGO		**RLC_TX**	**MAC_ALGO**		**DATA_ENC**		
3 – 32-bit x=x+1		1 - Yes	2 – AES128 4 byte		1 – VAES – AES128 - RND or 3 – VAES – AES128 - RLC		

RLC_TX shall be explicit included inside the telegram and the concept of RLC window limit shall not be applied. This results in accepting any incoming RLC from the communication partner as long as it has a higher value as the previously stored RLC. Roll over is not supported.

If RLC counter reaches its maximum a new teach-in with a new AES Security code shall be executed. With new AES code the RLC counter can be set to 0 again.

This removes the need of RLC resynchronization and simplifies the use case scenario by not compromising any security protection.

A.1.2.1 Secure Teach-in with Standardized labels

Plain text AES key transmission over air at teach-in moment is a weak point of the current security concept. To address this issue the PSK Protection was defined. With respecting the above recommendation to use explicit RLC transfer there is another scenario to address this weak point.

The PSK scenario includes scanning the a devices PSK and entering it into the receiver. By keeping these steps following scenario is possible:

1. Scan the 128bit AES communication code from label of the communicating device. (similar like PSK)
2. Enter the 128bit AES code into the receiving device.
3. The receiving device will use the RLC from the first data telegram for initialization.

NOTE: Since the RLC is part of the CMAC computing the RLC in every data telegrams is signed. Therefore there is no additional risk in taking the RLC from first radio telegram. It is recommended that first data telegram must be triggered by the user at defined moment similar to teach-in telegrams.

A.1.3 Exceptions for energy limited applications

We always recommend following above definition if the application energy budget allows it. In applications which are absolutely energy constrained (e.g. kinetic applications), following exceptions are considered as acceptable:

7	6	5	4	3	2	1	0
RLC_ALGO		**RLC_TX**	**MAC_ALGO**		**DATA_ENC**		
2 – 24 bit x=x+1 or 3 – 32-bit x=x+1		0- No or 1 - Yes	1 - AES128 3 byte or 2 – AES128 4 byte		1 – VEAS – AES128 - RND or 3 – VAES – AES128 - RLC		

A.2 ERP1 secure telegrams

The following figure indicates the concretization of the secure message into ERP1 protocol.

A.2.1 Operation mode with EPR1

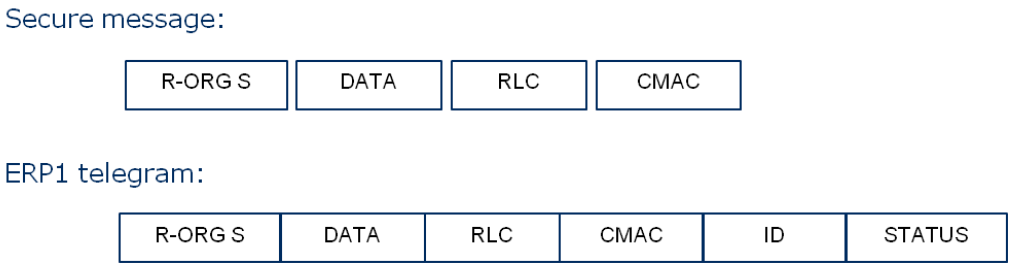


Figure 25. In the most general version of the secure message the RLC and CMAC fields are present. DATA can be encrypted. The fields R-ORG S, DATA, RLC and CMAC are integrated without modification in the ERP1 telegram. If DATA is encrypted in the message it is also encrypted in the ERP1 telegram. The ID and STATUS are specific to the telegram.

A.2.2 Secure teach-in chaining with ERP1

The ERP1 limits the amount of bytes transmitted to 21 per telegram. Therefore, the secure teach-in message must be partitioned in at least two telegrams (chaining).

Figure 26. The general teach-in message security members

Dividing the message into two telegrams results in the following:

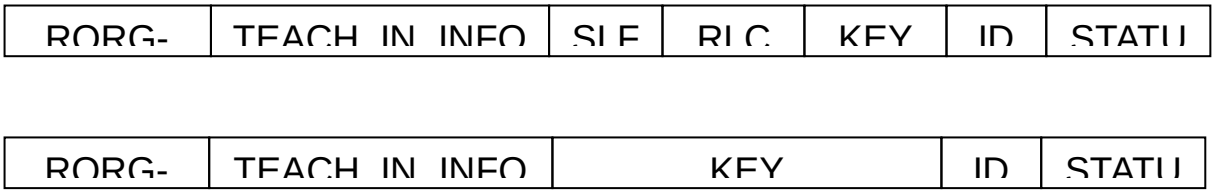


Figure 27. Teach-in message information divided in two ERP1 telegrams. The telegram with TEACH_IN_INFO_0 byte is sent first. This telegram contains the teach-in RLC, SLC and the first part of the KEY of the teach-in message. The second telegram transports the second part of the KEY.

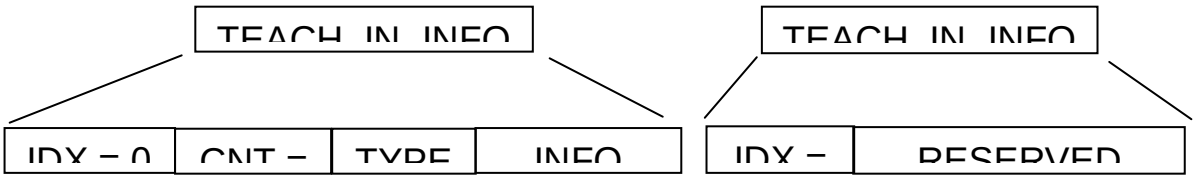


Figure 28. The IDX bit field of TEACH_IN_INFO_0 = 0 indicates that this is the first telegram. The CNT=2 indicates that the message is divided in 2 telegrams. TEACH_IN_INFO_1 IDX field = 1 says that this telegram is the second one. See chapter 4.1.2 to learn more about the TEACH-IN INFO bit fields.

It is specified, that there is no direct timeout between the chained telegrams of the teach-in-message. Each newer received telegram overwrites older received telegrams. At the end of the learn mode, all partly received messages will be deleted.

A.3 ERP2 secure telegrams

The following figure indicates the concretization of the secure message into ERP2 protocol.

A.3.1 Operation mode with EPR2

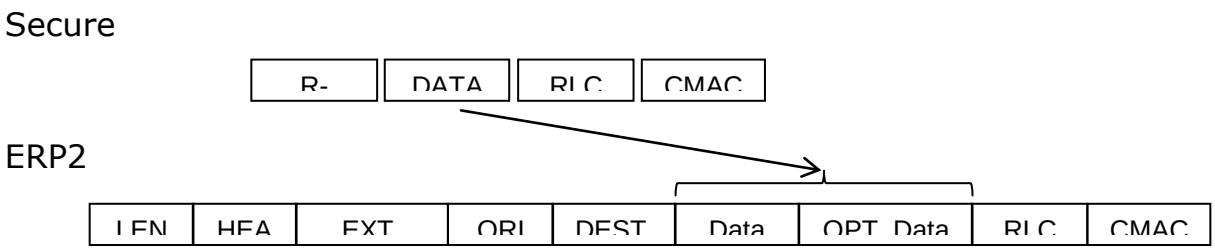


Figure 29. The secure message DATA field used throughout this documentation contains the information of the concatenated ERP2 telegram Data and Optional Data fields. If the message DATA field is encrypted is also encrypted in the telegram. The RLC and CMAC are placed after the Optional Data. The information of the R-ORG S is contained within the HEADER field. The EXTENDED HEADER depends on the telegram. See ERP2 Specification

A.3.2 Secure teach-in with ERP2

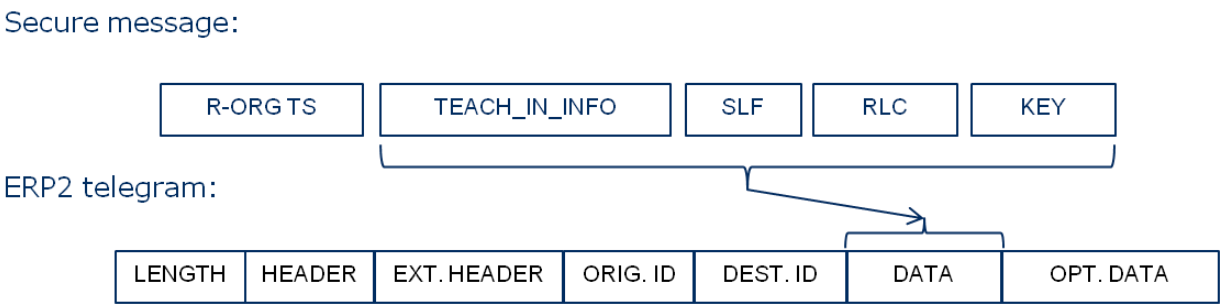


Figure 30. The message fields TEACH_IN_INFO, SLF, RLC and KEY are concatenated in the order indicated and placed in the DATA field of the ERP2 telegram. The information of R-ORG TS is contained within the HEADER. The EXTENDED HEADER depends on the telegram.

A.3.2.1 Secure teach-in chaining with ERP2

Many EnOcean applications have a very limited amount of energy available. For this reason, if needed, is possible to fragment the teach-in message (described in 4) in several telegrams.

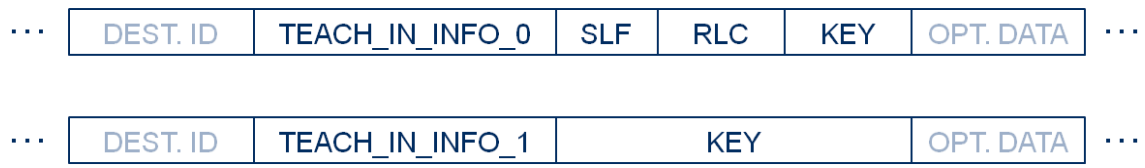


Figure 31. The secure teach-in message is transmitted here by two ERP2 telegrams. The information within the field DATA, between the telegram's Destination ID and Optional Data, is shown. SLF, RLC and the first part of the KEY are transmitted in the first telegram. The second telegram contains the second part of the KEY. The TEACH_IN_INFO_X bytes are interpreted like in the Figure 28.

It is specified, that there is no direct timeout between the chained telegrams of the teach-in-message. Each newer received telegram overwrites older received telegrams. At the end of the learn mode, all partly received messages will be deleted.

A.4 PSK CRC8 checksum algorithm

The polynomial is $P(x) = x^8 + x^2 + x^1 + x^0$

code uint8 u8CRC8Table[256] = {

0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15,
0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d,
0x70, 0x77, 0x7e, 0x79, 0x6c, 0x6b, 0x62, 0x65,
0x48, 0x4f, 0x46, 0x41, 0x54, 0x53, 0x5a, 0x5d,
0xe0, 0xe7, 0xee, 0xe9, 0xfc, 0xfb, 0xf2, 0xf5,
0xd8, 0xdf, 0xd6, 0xd1, 0xc4, 0xc3, 0xca, 0xcd,
0x90, 0x97, 0x9e, 0x99, 0x8c, 0x8b, 0x82, 0x85,
0xa8, 0xaf, 0xa6, 0xa1, 0xb4, 0xb3, 0xba, 0xbd,
0xc7, 0xc0, 0xc9, 0xce, 0xdb, 0xdc, 0xd5, 0xd2,
0xff, 0xf8, 0xf1, 0xf6, 0xe3, 0xe4, 0xed, 0xea,
0xb7, 0xb0, 0xb9, 0xbe, 0xab, 0xac, 0xa5, 0xa2,
0x8f, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9d, 0x9a,
0x27, 0x20, 0x29, 0x2e, 0x3b, 0x3c, 0x35, 0x32,
0x1f, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0d, 0x0a,
0x57, 0x50, 0x59, 0x5e, 0x4b, 0x4c, 0x45, 0x42,
0x6f, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7d, 0x7a,
0x89, 0x8e, 0x87, 0x80, 0x95, 0x92, 0x9b, 0x9c,
0xb1, 0xb6, 0xbf, 0xb8, 0xad, 0xaa, 0xa3, 0xa4,
0xf9, 0xfe, 0xf7, 0xf0, 0xe5, 0xe2, 0xeb, 0xec,
0xc1, 0xc6, 0xcf, 0xc8, 0xdd, 0xda, 0xd3, 0xd4,
0x69, 0x6e, 0x67, 0x60, 0x75, 0x72, 0x7b, 0x7c,
0x51, 0x56, 0x5f, 0x58, 0x4d, 0x4a, 0x43, 0x44,
0x19, 0x1e, 0x17, 0x10, 0x05, 0x02, 0x0b, 0x0c,
0x21, 0x26, 0x2f, 0x28, 0x3d, 0x3a, 0x33, 0x34,
0x4e, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5c, 0x5b,
0x76, 0x71, 0x78, 0x7f, 0x6A, 0x6d, 0x64, 0x63,
0x3e, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2c, 0x2b,
0x06, 0x01, 0x08, 0x0f, 0x1a, 0x1d, 0x14, 0x13,
0xae, 0xa9, 0xa0, 0xa7, 0xb2, 0xb5, 0xbc, 0xbb,

0x96, 0x91, 0x98, 0x9f, 0x8a, 0x8D, 0x84, 0x83,
0xde, 0xd9, 0xd0, 0xd7, 0xc2, 0xc5, 0xcc, 0xcb,
0xe6, 0xe1, 0xe8, 0xef, 0xfa, 0xfd, 0xf4, 0xf3

```
crc=0;
for (i=0; i<sizeof(key); i++)
{
   crc  = u8CRC8Table[crc ^ key[i]];
}
```

Example

key = 0x3410de8f1aba3eff9f5a117172eacabd

crc8 = 0x07

A.5 Security Test vectors

The following test vectors are useful to check the user implementation for secure applications.

A.5.1 Secure STM with ID (01 9E B6 3B)

The first example uses variable AES128 as encryption algorithm, 4 byte CMAC and 3 byte rolling code as security parameters.

- + Teach-In Message:Transmitted packets over the air:
  1) 35 20 93 C0 FF EE 45 6E 4F 63 65 61 6E 01 9E B6 3B 00
  2) 35 40 20 47 6D 62 48 2E 31 33 00 01 9E B6 3B 00
- Teach-Info : 0x00
- Security Layer format: 0x93

  "SLF_DATA_ENC_VAES128 | SLF_MAC_4BYTE | SLF_RLC_ALGO_24BIT"
- Security Key:

  0x45, 0x6E, 0x4F, 0x63, 0x65, 0x61, 0x6E, 0x20, 0x47, 0x6D, 0x62, 0x48, 0x2E,

  0x31, 0x33, 0x00
- Rolling code: 0xC0FFEE

+ Unsecure Data message:

- Transmitted Packet over the Air:

  1) A5 08 27 FF 80 01 9E B6 3B 00
- RORG: 0xA5
- DATA: 0x08, 0x27, 0xFF, 0x80

+ Secure Data Message:

Data encryption:

```
VAES init vector    0x3410de8f1aba3eff9f5a117172eacabd
XOR
RLC padded          0xc0ffee00000000000000000000000000
-------------------------------------------------
Input               0xf4ef308f1aba3eff9f5a117172eacabd
AES
Key                 0x456E4F6365616E20476D62482E313300
-------------------------------------------------
                    0x9be2e35d5fe7858645200587dd3b515c
XOR
Data padded         0xa50827ff8000000000000000000000
-------------------------------------------------
Data encrypted      0x3eeac4a2dfe7858645200587dd3b515c
```

Subkey generation for CMAC calculation

```
Key          0x456E4F6365616E20476D62482E313300
AES128
Input        0x00000000000000000000000000000000
-------------------------------------------------
        0xdd21aa892ddf3cb967c314369a272338
<<

-------------------------------------------------
             0xba4355125bbe7972cf86286d344e4670
XOR
             0x00000000000000000000000000000087
-------------------------------------------------
K1=      0xba4355125bbe7972cf86286d344e46f7
<<

-------------------------------------------------
```

```
             0x7486aa24b77cf2e59f0c50da689c8dee
XOR
             0x00000000000000000000000000000087
-----------------------------------------------
K2=          0x7486aa24b77cf2e59f0c50da689c8d69


Message padded 0x313eeac4a2dfc0ffee80000000000000
XOR
K2             0x7486aa24b77cf2e59f0c50da689c8d69
-------------------------------------------------
Input          0x45b840e015a3321a718c50da689c8d69
AES128
Key            0x456E4F6365616E20476D62482E313300
-------------------------------------------------
CMAC(128bits)  0xeaf20eed28679f641c15b10b9308d0d4
```

- Transmitted Packet over the Air:

1) 31 3E EA C4 A2 DF EA F2 0E ED 01 9E B6 3B 00

- RORG: 0x31 (secure ROG with encrypted original ROG)
- DATA: 0x3E, 0xEA, 0xC4, 0xA2, 0xDF
- CMAC: 0xEA, 0xF2, 0x0E, 0xED
- RLC (not transmitted): 0xC0FFEE

1. The second example uses variable AES128 as encryption algorithm, 3 byte Rolling code and no CMAC as security parameters.

+ Teach-In Message

- Transmitted packets over the air

  1) 35 20 83 C0 FF EE 45 6E 4F 63 65 61 6E 01 9E B6 3B 00

  2) 35 40 20 47 6D 62 48 2E 31 33 00 01 9E B6 3B 00

- Teach-Info : 0x00
- Security Layer format: 0x83

  "SLF_DATA_ENC_VAES128 | SLF_MAC_NO | SLF_RLC_ALGO_24BIT"

- Security Key:

  0x45, 0x6E, 0x4F, 0x63, 0x65, 0x61, 0x6E, 0x20, 0x47, 0x6D, 0x62, 0x48,

  0x2E, 0x31, 0x33, 0x00

- Rolling code: 0xC0FFEE

+ Unsecure Data Message:

- Transmitted Packet over the Air:

  1) A5 08 27 FF 80 01 9E B6 3B 00

- RORG: 0xA5

- DATA: 0x08 0x27 0xFF 0x80

+ Secure Data Message:

- Transmitted packet over the air:

  31 3E EA C4 A2 DF 01 9E B6 3B 00

- RORG: 0x31
- DATA: 0x3E 0xEA 0xC4 0xA2 0xDF. Same byte sequence as in example 1.
- RLC: 0xC0FFEE

A.5.2 Secure PTM (with ID 01 85 E1 77)

1. This example uses variable AES128 encryption algorithm, 3 byte CMAC and 2 byte rolling code.

+ Teach-In Message:

- Transmitted packets over the air:

  1) 35 24 4B 3E 2D 45 6E 4F 63 65 61 6E 01 85 E1 77 00
  2) 35 40 20 47 6D 62 48 2E 31 33 00 01 85 E1 77 00

- Teach-In info: 0x04 // TEACH_INFO_PTM | TEACH_INFO_PTM_ROCKERA
- Security Layer format = 0x4B

  SLF_DATA_ENC_VAES128 | SLF_MAC_3BYTE | SLF_RLC_ALGO_16BIT

- Security Key:

  0x45, 0x6E, 0x4F, 0x63, 0x65, 0x61, 0x6E, 0x20, 0x47, 0x6D, 0x62, 0x48,

0x2E, 0x31, 0x33, 0x00

- Rolling code: 0x3E2D

+ Unsecure Data message:

- Transmitted Data over the air:

  D2 09 01 85 E1 77 00

- RORG: 0xD2
- DATA: 0x09

+ Secure Data message:

Data encryption:

```
VAES init vector    0x3410de8f1aba3eff9f5a117172eacabd
XOR
RLC padded         0x3E2D00000000000000000000000000
-------------------------------------------------
Input               0x0a3dde8f1aba3eff9f5a117172eacabd
AES
Key                 0x456E4F6365616E20476D62482E313300
-------------------------------------------------
                    0xc77f3257ca9f626ad8f6ed44db04d26c
XOR
Data padded         0x09
-------------------------------------------------
                    0xce
AND
Mask(*)           0x0f
-------------------------------------------------
```

Data encrypted 0x0e

(*) For PTM secure telegrams the secured data most significant byte is chopped.

CMAC generation (like in example 1)	
Key	0x456E4F6365616E20476D62482E313300
Subkey K2	0x7486aa24b77cf2e59f0c50da689c8d69
Message padded	0x300E3E2D8000000000000000000000000
XOR	
K2	0x7486aa24b77cf2e59f0c50da689c8d69
Input	0x44889409377cf2e59f0c50da689c8d69
AES128	
Key	0x456E4F6365616E20476D62482E313300
CMAC(128bits)	0x**ebdcc4**7285ab750a21c06a4ed2627ad0

- Transmitted Data over the air:

  30 0E EB DC C4 01 85 E1 77 00

- RORG: 0x30
- DATA: 0x0E
- CMAC: 0xEB, 0xDC, 0xC4

+ Decrypted Data Message by the receiver:

- Transmitted Data over the air:

32 09 01 85 E1 77 00

- RORG: 0x32
- DATA: 0x09

A.5.3 Secure MSC Data

Expected decrypted data:

RORG: 0xD1
0x54 0x68 0x69 0x73 0x20 0x61 0x20 0x76 0x65 0x72 0x79 0x20 0x6C 0x6F 0x6E 0x67 0x20 0x73 0x74 0x72 0x69 0x6E 0x67 0x20 0x74 0x6F 0x20 0x67 0x65 0x6E 0x65 0x72 0x61 0x74 0x65 0x20 0x6E 0x6F 0x69 0x73 0x65 0x21 0x00

Used encryption KEY:
0x45 0x4f 0x54 0x45 0x53 0x54 0x4b 0x45 0x59 0x59 0x45 0x41 0x48 0x21 0x5c 0x30

Used SLF: SLF_RLC_TX_ON | SLF_RLC_32_BIT | SLF_MAC_3_AES128 | SLF_DATA_ENC_VAES128

RLC: **0x01 0x02 0x03 0x04**

Generated Encrypted Message:

RORG: 0x31
Payload with encrypted RORG + Payload +RLC+ CMAC:

0x1D 0x70 0xD9 0x54 0x01 0x0F 0x84 0x78 0x2B 0x24 0xF6 0x32 0xE6 0x65 0xAD 0x7D 0x65 0x21 0xAE 0x4F 0x34 0x84 0xB6 0x39 0x08 0xF7 0x12 0xFC 0x0A 0xAC 0x56 0x76 0x49 0xC4 0xC9 0xF2 0xA8 0x29 0x7D 0xF3 0x61 0x99 0x51 0x0D 0x01 0x02 0x03 0x04 0x2F 0x71 0x9F

Transmitted Telegrams over air:

IDX = 0

RORG	SEQ	IDX	Date length	Payload
SEC_CDM	**2 bit**	**6 bit**	**2 byte**	**11 byte**
33	**2**	**0**	**00 33**	**1D 70 D9 54 01 0F 84 78 2B 24 F6**

IDX = 1

SEC_CDM	SEQ	IDX	Payload
SEC_CDM	2 bit	6 bit	12 byte
33	2	1	32 E6 65 AD 7D 65 21 AE 4F 34 84 B6 39

IDX = 2

RORG	SEQ	IDX	Payload
SEC_CDM	2 bit	6 bit	12 byte
33	2	2	08 F7 12 FC 0A AC 56 76 49 C4 C9 F2 A8

IDX = 3

RORG	SEQ	IDX	Payload
SEC_CDM	2 bit	6 bit	12 byte
33	2	3	29 7D F3 61 99 51 0D 01 02 03 04 2F 71

IDX = 4

RORG	SEQ	IDX	Payload
SEC_CDM	2 bit	6 bit	1 byte
33	3	4	9F