SPECIFICATION

Green.Smart.Wireless.
enocean®

Security of EnOcean Radio Networks

V1.3 / 31.10.2012

Content

Published by EnOcean GmbH, Kolpingring 18a, 82041 Oberhaching, Germany
www.enocean.com, info@enocean.com, phone ++49 (89) 6734 6890

Important!

1 Security of EnOcean Radio Networks

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
ADT	Addressed Data Telegram

Table 1. Abbreviations used in this document.

2 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.

3 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.

3.1 Attacker scenarios

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

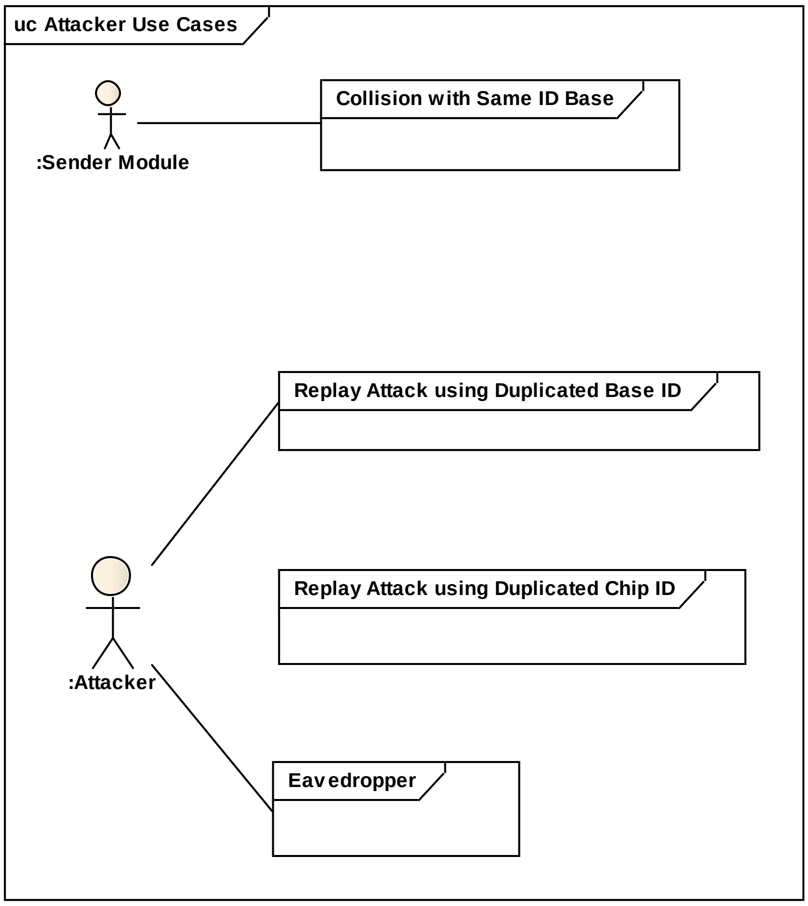

uc Attacker Use Cases
:Sender Module
Collision with Same ID Base
Replay Attack using Duplicated Base ID
Replay Attack using Duplicated Chip ID
:Attacker
Eavedropper

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*.

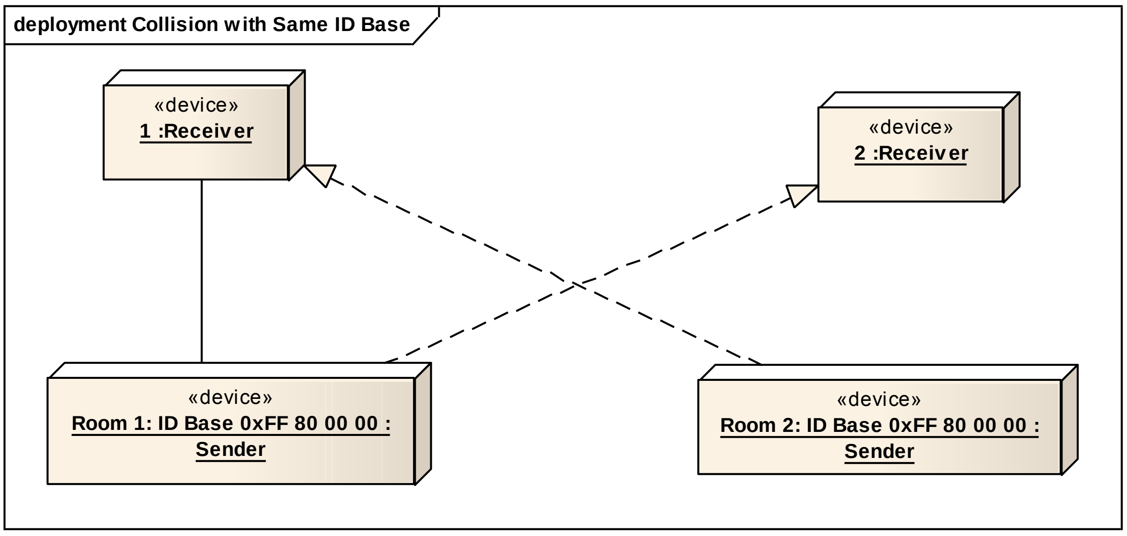

deployment Collision with Same ID Base
«device»
1 :Receiver
«device»
2 :Receiver
«device»
Room 1: ID Base 0xFF 80 00 00 : Sender
«device»
Room 2: ID Base 0xFF 80 00 00 : Sender

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.

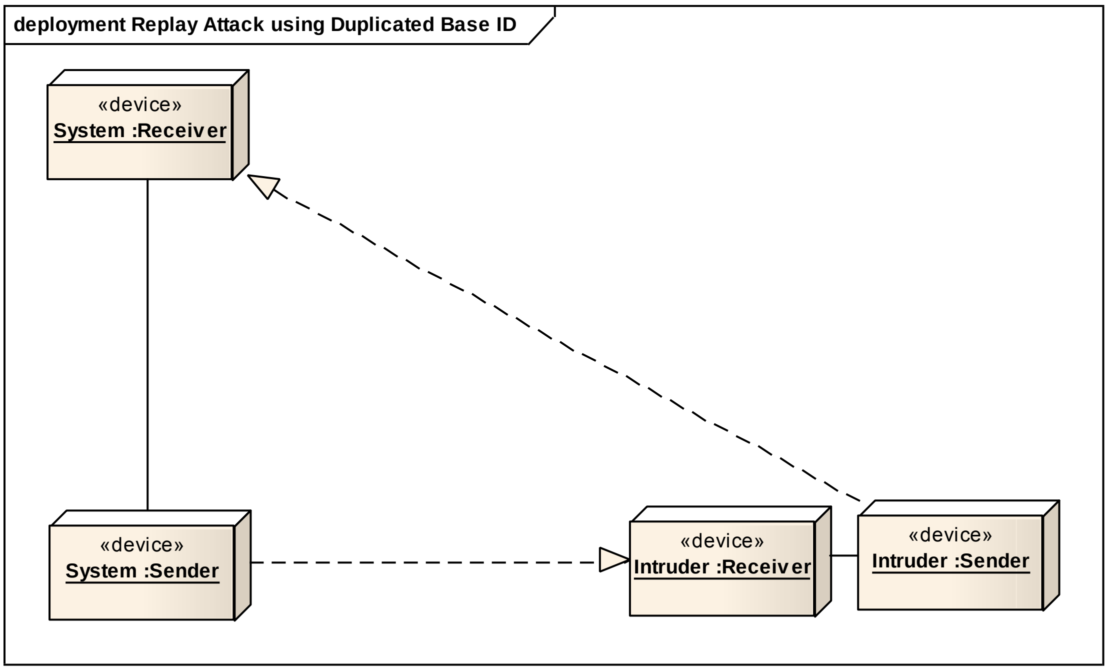


Figure 3. In this scenario an attacker programs the ID base taking intentional control of the system receiver. To do this, the intruder simply listens to the sender ID. It programs then this same Base ID in his sender module, and controls the system receiver. All this is possible using EO tools without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

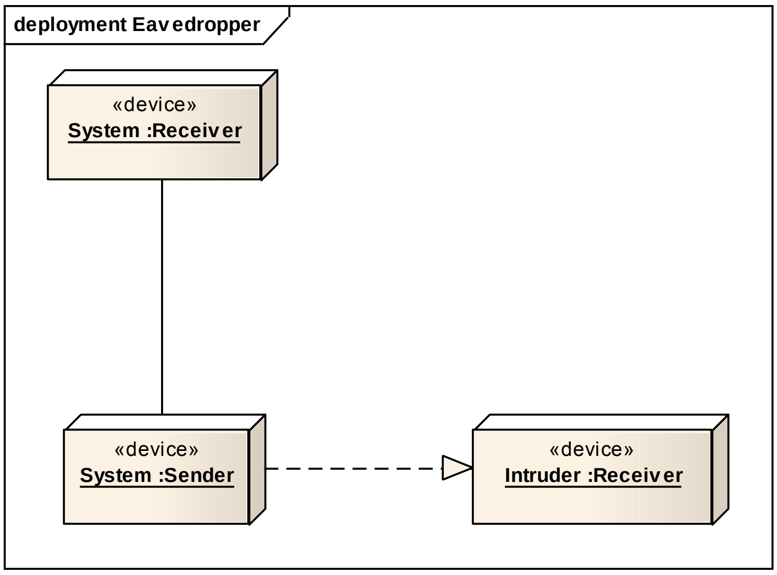


Figure 4. With the only help of an EO receiver it is possible to hear the communication between an EO sender and a receiver. This is undesired if the information being transmitted is private. This is the case of metering reading systems.

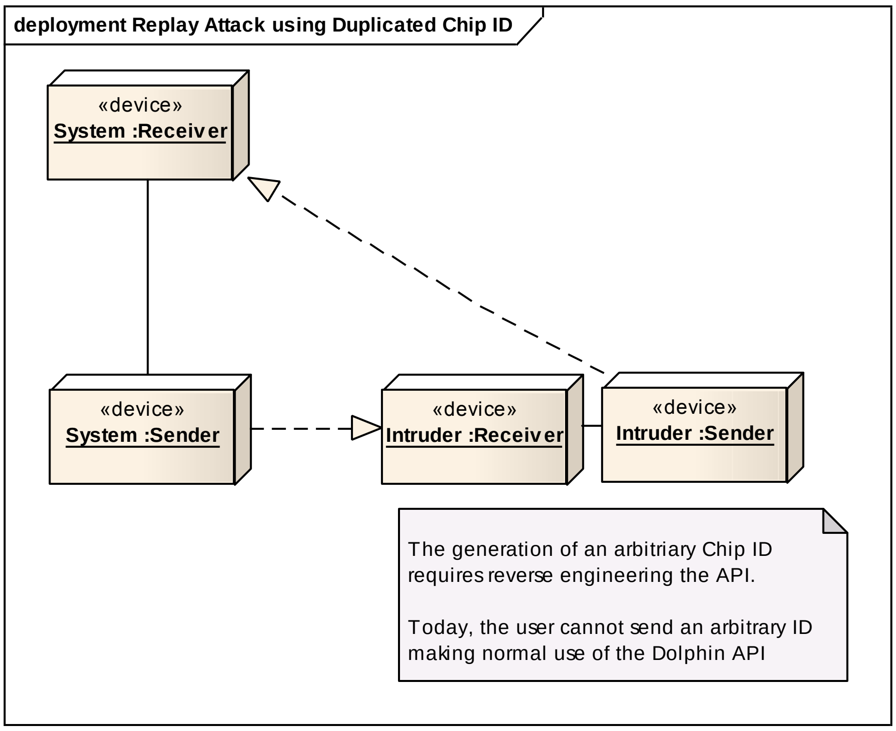

deployment Replay Attack using Duplicated Chip ID
«device»
System :Receiver
«device»
System :Sender
«device»
Intruder :Receiver
«device»
Intruder :Sender
The generation of an arbitriary Chip ID requires reverse engineering the API.
Today, the user cannot send an arbitrary ID making normal use of the Dolphin API

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.

3.2 System Architecture

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

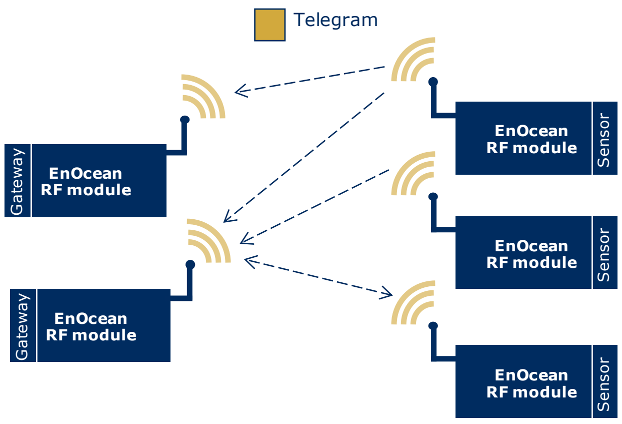


Figure 6 EHW system architecture

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.

4 Specification

For better readiness of the document we removed the Checksum/CRC-Field of each EnOcean-Standard-Telegram from all images used in this document. Please notice, that each telegram is saved by a Checksum/CRC-Field specified in the status byte.

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

RORG	Description
0x30	Secure telegram. Dependend from SLF the data may be encrypted, RLC and/or CMAC is included. There was NO unsecure telegram. A telegram 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	Converting a non secure telegram with given R-ORG to a secure telegram. The original R-ORG will be transmitted by the secure telegram as the first data byte.
0x32	After decryption and authentification of RLC and/or CMAC of a telegram with RORG = 0x30 the unsecure telegram becomes the RORG 0x32. Data is decrypted, RLC and CMAC is removed.
0x35	Secure Teach-In telegrams, used to transmit private key and rolling code as plain text (unsecure) to the communication partner

Table 2 New RORGs for Security handling

4.1 Security telegram structure for operation mode

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

The EO secure telegrams follow a flexible schema made up of different security mechanisms that can be combined.

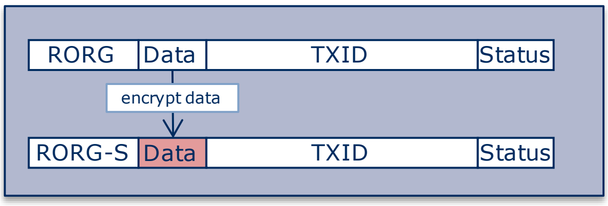
Through the combination of all security strategies the user/module reaches a higher degree of security at the expense of a high amount of energy spent due to higher processing and transmission time.

The following figure depicts secure radio telegram structures, together with examples of application scenarios, attacks they can block, and the qualitative energy factor they represent.

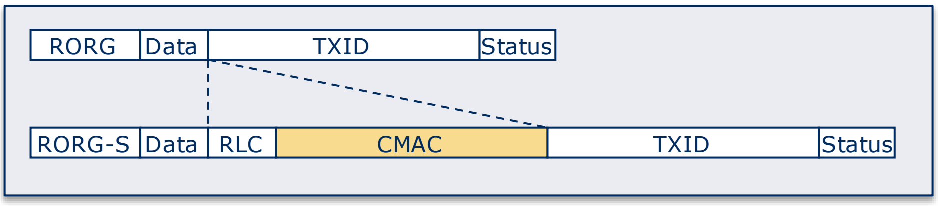
Telegram	Scenario	Reply Attack	Eaves-Dropping	Energy & resource demand
RORG / Data / TXID / Status	unsecured	-	-	Energy harvesting
RORG-S / Data / TXID / Status	Thermostat Vacation mode	-	X	
RORG-S / Data / RLC / CMAC / TXID / Status	Window Control	X		
RORG-S / Data / RLC / CMAC / TXID / Status	Automated Meter Reading	X	X	Line powered

Table 3 EnOcean secure telegrams can be built up with a higher or lower degree of security, which allows adapting the telegram structure to the security vs. energy application requirements. The table shows some secure telegrams structures for typical applications requiring security. Telegram structure fields in green represent a security step compared to the unsecure telegram structure. Different telegram structures offer different degree of protection against a certain type of attack. **Any combination of DATA encrypted/not encrypted, RLC present/not present, and CMAC present/not present is possible.**

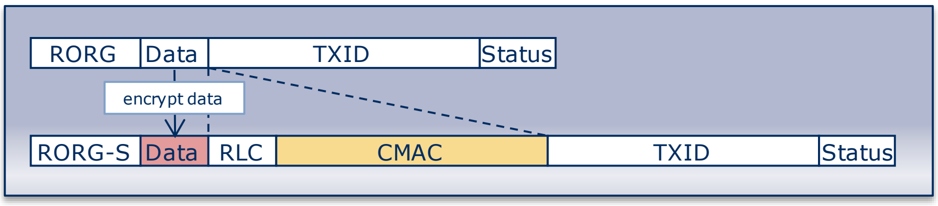
The structure of the secure telegram is negotiated between the devices within the teach-in telegram procedure. See 4.2.



Encrypting Data



Inserting Rolling Code



Encrypting Data & Inserting Rolling Code

Figure 7. Various methods for securing telegrams

4.1.1 Field RORG-S

There are two RORG-S codes for secure telegrams. The code 0x31 indicates that the non-secure telegram RORG is included as part of the secure telegram. RORG-S 0x30 indicates that the secure telegram does not contain any information about the non-secure RORG

4.1.1.1 Encapsulation of non-secure RORG

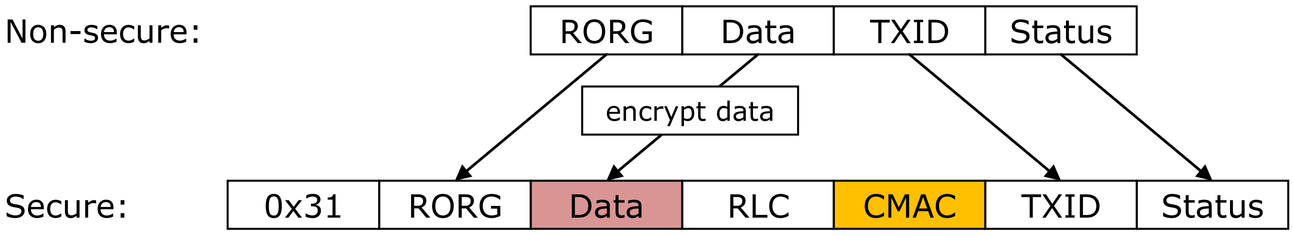


Figure 8. Inclusion of the non-secure RORG in the secure telegram is indicated by the RORG-S = 0x31

In this case, when the receiver gets a secure message with RORG-s 0x31, the non-secure telegram (after encrypting data and authentification of RLC and CMAC) is like this:

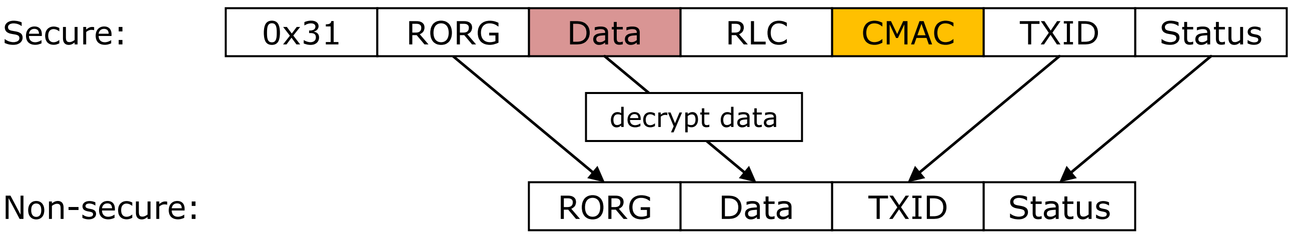


Figure 9. Decapsulaton of a telegram with RORG-S 0x31 and exclusion of the encapsulated RORG

4.1.1.2 Secure telegram without non-secure RORG

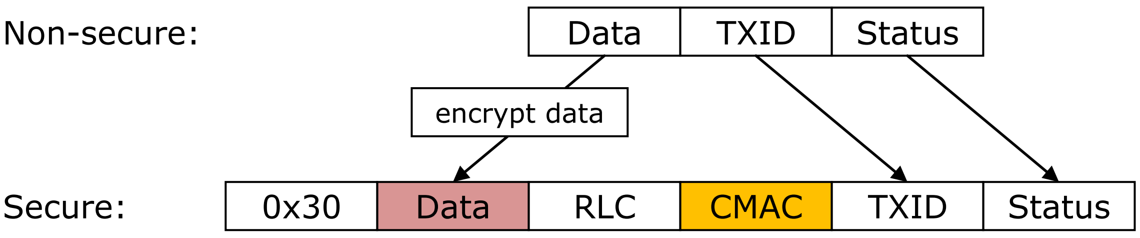


Figure 10. Inclusion of the non-secure RORG in the secure telegram is indicated by the RORG = 0x30

In this case, when the receiver gets a secure message with RORG-S 0x30, the non-secure telegram (after encrypting data and authentification of RLC and CMAC) is like this:

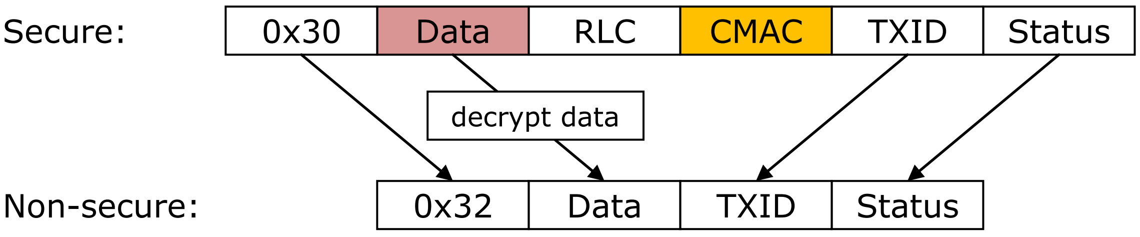


Figure 11. The secure telegram with RORG-S 0x30 does not include the non-secure RORG information. In the conversion from secure to non-secure telegram the latter becomes the RORG = 0x32. The interpretation of Data is done by the application.

4.1.2 Field Data

The data is encrypted using any of the encryption algorithms described in section 4.3.

The kind of data encryption algorithm is indicated to the receiver during the teach-in procedure.

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

4.1.3 Field RLC (ROLLING CODE)

This field contains a code that changes according to a predefined algorithm (see 4.4.3) that both sender and receiver know. If a telegram RLC does not coincide with the expected RLC value the receiver rejects the telegram.

The code changes every time a telegram is sent by the transmitter.
RLC is MSB first.

The first rolling code is sent during the teach-in procedure. This first code synchronizes the RLC in the transmitter and the receiver.

The RLC telegram 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 MAC code and DATA field encryption. See 4.4.4

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

Calculation of the RLC algorithm is explained in section 4.4.3

4.1.4 Field CMAC

The CMAC (cipher-based message authentication code) field–with MSB first- is included just before the ID to authenticate a telegram.

The MAC is dependent on the private key (see 4.2.5), a public vector, the telegram bytes, and if it exists, the rolling code (see 4.1.3).

The MAC field code should be used in combination with a RLC. The MAC field code changes then for every sent telegram in an unpredictable way unless the key is known.

The algorithm to calculate CMAC is explained in the chapter 4.4.4

4.1.5 Field TXID

4 byte long transmitter module identifier.

4.1.6 Field Status

1 byte long status field according to Standard EnOcean telegram.

4.2 Message structure for teach-in procedure

To configure the details of the secure communication in operation mode a teach-in procedure mode must be started in both receiver and sender module.

A preferred method for the activation of the teach-in mode in the modules is via a special key activation or via serial interface. In contrast, the activation of the teach-in via wireless represents a high risk of attack.

The teach-in method is limited typically to 30 seconds. After this time-out the modules leaves their teach-in mode. Teach-in messages are not accepted until the next activation of the teach-in mode.

By means of the teach-in message the transmitter communicates the receiver the private key as well as other details of the secure protocol that will be used during the operation mode.

These details are transmitted in plain text (no encryption in the information is performed).

The teach-in message has following structure:

8	8	8	16/24	128	32
RORG-TS	TEACH_IN_INFO	SLF	RLC	KEY	ID

Figure 12 Secure teach-in message. Field are measured in bits. The private KEY is always a multiple of 8 bits. The private KEY length is 128 bits (16 bytes).

This secure teach-in message does only transfer the security specific data. This message does not contain the information how the operational data has to be interpreted (except for field type PTM). To enable interpretation by any profile a further profile-teach-in message (EEP or GP) has to be transmitted, also in plain text.

4.2.1 Field RORG-TS

Secure teach-in message Choice or message RORG. TRLChe code is 0x35.

4.2.2 Field Teach-in Info

7	6	5	4	3	2	1	0
RESERVED				**TYPE**		**INFO**	
				0 – Non-specific 1 – PTM 2 – N/A 3 – N/A		Interpretation depends on Type. See 4.2.2.1	

Table 4. Teach-in info bits

4.2.2.1 Field TYPE

Indicates the type of application that sends the message

0- Non-specific type

1- PTM

2- N/A

3- N/A

4.2.2.2 Field INFO

The interpretation of this field depends on the code in Field TYPE (4.2.2.1)

If TYPE = 1

0	ROCKER A
1	ROCKER B
2	N/A
3	N/A

N/A for other TYPE values

4.2.3 Field SLF (Security level format)

With the information contained in the SLF the receiver is informed about:

- The next bytes that the teach-in contains.
- What is the format of the secure telegrams 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 – N/A		0 - No 1 - Yes	0 – No MAC 1 – AES128 3 byte 2 – AES128 4 byte 3 – N/A		0 – No data encryption 1 – N/A 2 – N/A 3 – VAES – AES128		

			4 – AES-CBC – AES128

Table 5. Security Level Format fields and its interpretation.

4.2.3.1 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 Not defined algorithm.

4.2.3.2 Field RLC_TX

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

0 Secure operation mode telegrams do not contain RLC. A RLC could still be used internally for MAC and DATA encryption as indicated by 4.2.3.1. In this case the initial RLC will be sent by the teach-in message (see 4.2.4).

1 Secure operation mode messages contain RLC.

4.2.3.3 Field MAC_ALGO

Defines the type of algorithm for the MAC calculation

0 No MAC included in the secure telegram.

1 MAC is a 3-byte-long code. MAC is calculated using AES128 as described in the 4.4.4 chapter

2 MAC is a 4-byte-long code. MAC is calculated using AES128 as described in the

4.4.4 chapter

3 N/A

4.2.3.4 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 a AES128. See 4.3.3.

4 DATA will be encrypted/decrypted using the AES128 algorithm. See 4.3.1. and 4.3.2 sections

4.2.4 Field RLC

Contains the initial rolling code needed for the transmitter-receiver RLC synchronization.

4.2.5 Field KEY

Contains the private key used for the encryption of DATA and MAC generation.

4.2.6 Field TXID

4 byte long transmitter module identifier.

4.2.7 Secure teach-in message telegram chaining

EnOcean ASIC tx/rx buffer poses a limit of 21 bytes to the length of telegrams. Moreover, many EnOcean applications have a very limited amount of energy available.

For these reason is possible to fragment the teach-in message (described in 4.2) in several telegrams.

8	2	2	2	2	8	16-24		32	8
RORG-TS	IDX	CNT	TYPE	INFO	SLF	RLC	KEY	ID	STATUS

8	2	6		32	8
RORG-TS	IDX	RESERVED	KEY	ID	STATUS

Figure 13. The teach-in information is encapsulated in this case in two telegrams. Field lengths are expressed in bits. The KEY may have a variable length depending from the encryption algorithm used.

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.

4.2.7.1 Field RORG-TS

Contains the secure teach-in identifier fix code 0x35.

4.2.7.2 Field IDX

Indicate the order of the telegram in the telegram chain. Taking the Figure 13 as example, the first telegram has IDX=0 and CNT=2.The second telegram contains IDX=1.

4.2.7.3 Field CNT

Indicates how many telegrams comprise the telegram chain. This field exists only in the telegram with IDX = 0 (the 1st telegram). In the example of Figure 13 the field contains the value 2, since the information is sent in two telegrams.

4.2.7.4 Field KEY

Private key for the encryption/decryption in operation mode. Length is 128 bit. The most significant bits are located in the first telegram till the maximum telegram length is reached. All remaining bytes are located in the second telegram.

4.2.7.5 Field ID

The transmitter ID field of chained telegrams fields must be the same.

4.2.7.6 Field STATUS

The STATUS field of both telegrams fields must be the same. See 4.1.6.

4.3 Security algorithms

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

4.3.1 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.

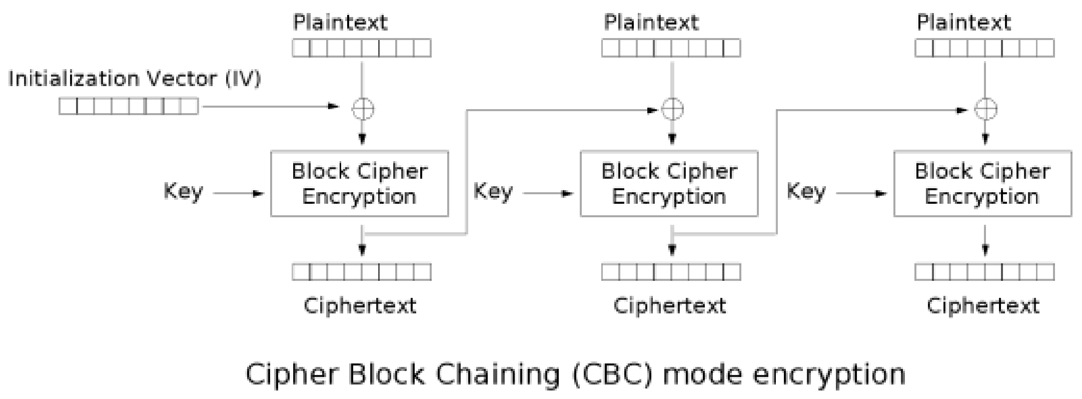


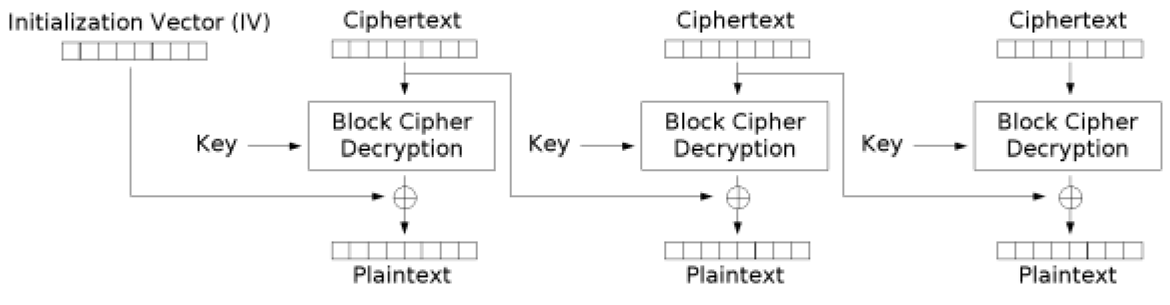
Figure 14. AES128 schematic diagram flux. DATA to be encrypted is called "plaintext" in the figure. The plaintext bytes are divided in chunks of 16 bytes. The first 16-byte array is the plaintext array to the left. If the last piece of plaintext is not 16-byte long the bit sequence 10..0 will be added 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.

4.3.2 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].



Cipher Block Chaining (CBC) mode decryption

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).

4.3.3 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 16 bytes to be encrypted. The generated encrypted code varies although the DATA input is constant. This feature improves the obscurity of the original information and avoids replay attacks.

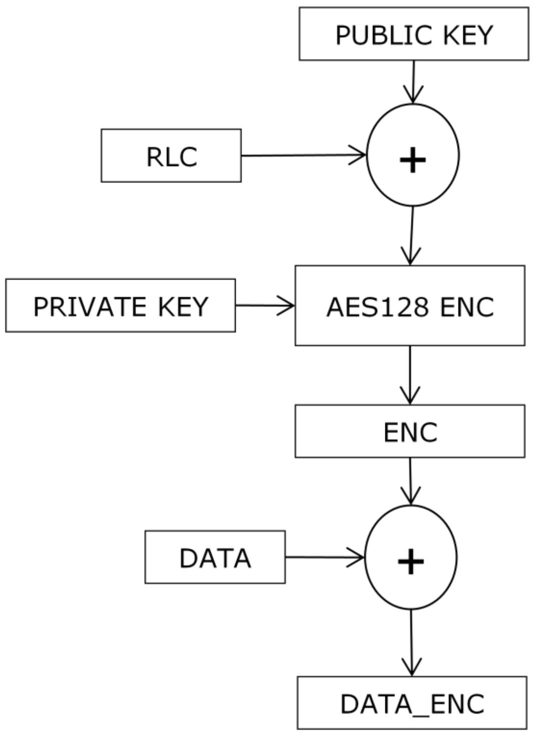


Figure 16. Encryption of telegram DATA using variable XOR AES128. The AES128 is used to perform a pseudo random generator. The plus symbol within a circle represents a bitwise XOR operation. DATA can have a maximum of 16 bytes. The length of DATA_ENC is equal to the length of DATA. PUBLIC KEY is a constant 16-byte hexadecimal string array. The RLC, is XORed with the PUBLIC key before entering the AES128 algorithm. The PUBLiC KEY byte array first element, PUBLIC_KEY[0], is XORed with the RLC most significant byte. PUBLIC_KEY[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

4.3.4 VAES (variable AES) decryption

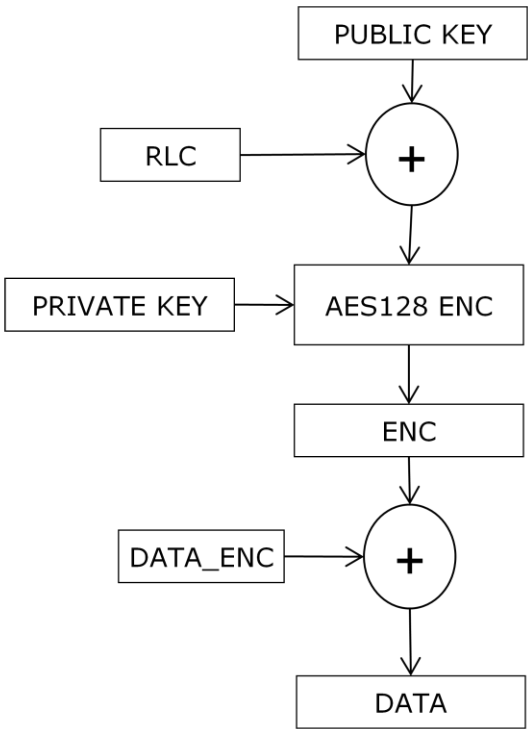


Figure 17. Decryption algorithm of the received telegram. DATA_ENC is the received encrypted DATA from the transmitter. Note: The AES algorithm required is the encryption algorithm, not the AES decryption algorithm. The RLC value in the receiver must be the same as in the transmitter module. Otherwise 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

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 4.4.4).

4.4 Security contents

4.4.1 Public key

The public key 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 PUBLIC_KEY first element, PUBLIC_KEY[0] = 0x34
The array last element is PUBLIC_KEY[15] = 0xbd

4.4.2 Private key

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

4.4.3 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 MAC codes (4.4.4) and VAES encryption values(4.3.3) although telegram DATA remains constant.

Explicit RLC strategy - The RLC may be transmitted as part of the telegram information (4.1) to speed-up the MAC calculation in the receiver, at expense of increasing the energy spent in radio transmission and decreasing the degree of security.

Implicit RLC strategy - It is possible for the transmitter module not to send RLC in the telegram. This strategy increases the system security at the expense of increasing the processing time for CMAC verifications (see 4.4.4).

If the RLC is transmitted or not is indicated by the Field 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.

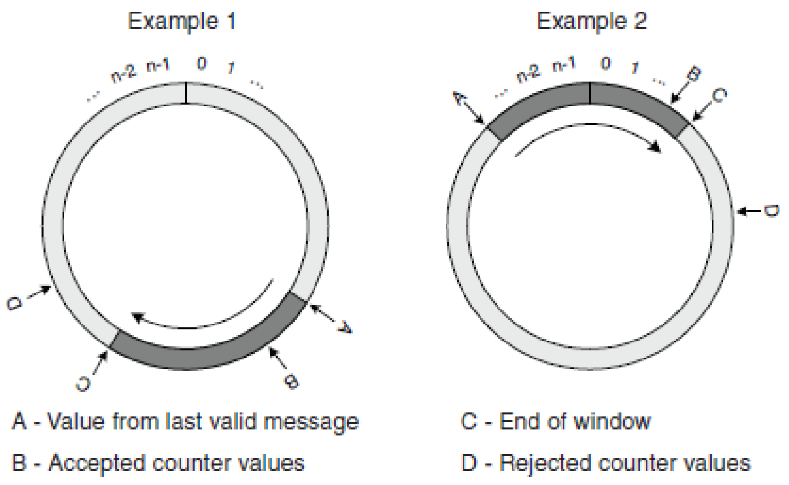

Example 1
Example 2
... n-2 n-1 0 1 ...
A
B
C
D
A - Value from last valid message
B - Accepted counter values
C - End of window
D - Rejected counter values

Figure 18 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 roling 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 4.2).

The window size for incorrect RLC has a value of 128.

4.4.4 CMAC algorithm

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

```
+------+     +------+     +------+     +------+     +------+     +---+----+
| M_1  |     | M_2  |     | M_n  |     | M_1  |     | M_2  |     |M_n|10^i|
+------+     +------+     +------+     +------+     +------+     +---+----+
   |            |            |   +--+     |            |            |   +--+
   |      +--->(+)     +--->(+)<-|K1|     |      +--->(+)     +--->(+)<-|K2|
   |      |     |      |     |   +--+     |      |     |      |     |   +--+
+------+  |  +------+  |  +------+     +------+  |  +------+  |  +------+
|AES_K |  |  |AES_K |  |  |AES_K |     |AES_K |  |  |AES_K |  |  |AES_K |
+------+  |  +------+  |  +------+     +------+  |  +------+  |  +------+
   |      |     |      |     |            |      |     |      |     |
   +------+     +------+     |            +------+     +------+     |
                             |                                      |
                          +------+                                +------+
                          |   T  |                                |   T  |
                          +------+                                +------+

 (a) positive multiple block length             (b) otherwise
```

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).

For simplicity, if the information to authenticate is <=16 bytes:

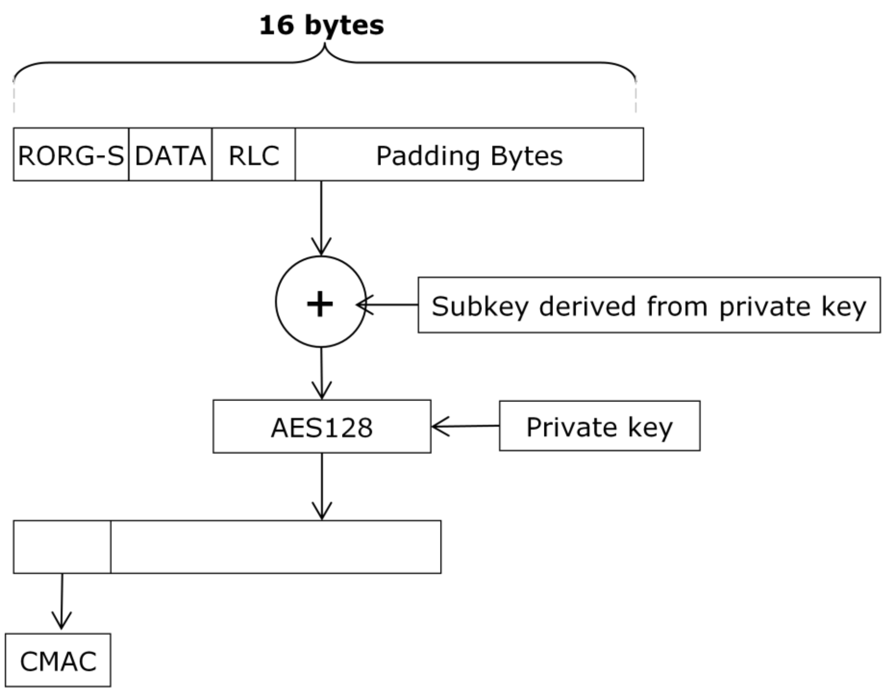


Figure 19 CMAC generation algorithm (3]. The RORG-S is the secure radio telegram choice and DATA the telegram payload. If no RLC is used (either in the telegram or internally) the field RLC is not included for the CMAC calculation. From the AES-generated array the bytes with index 0,1,… (CMAC_size- 1) are taken as CMAC.

When the current telegram fields (RORG-S, DATA -optionally RLC-) does not provide 16 bytes for applying AES128, the values are extended with the bit sequence 100…0 (padding bits). The padding bits are concatenated to the RORG DATA and RLC to form a total bit string of 16 bytes.

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

When performing AES, the byte RORG-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.

CMAC LSB is place next to the ID, as indicated in the secure telegram structure 4.1.

4.4.4.1 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.

5 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