Securely connecting IoT devices with LoRaWAN

Alexander Gladisch*, Simon Rietschel**
T-Systems Multimedia Solutions GmbH
Dresden** and Rostock*, Germany

E-mail: *alexander.gladisch@t-systems.com,
**simon.rietschel@t-systems.com

Thomas Mundt§ (corresponding), Johann Bauer‡,
Johannes Goltz¶, Simeon Wiedenmann‖
University of Rostock
Institute of Computer Science
Rostock, Germany
E-mail: §thomas.mundt@uni-rostock.de,
‡johann.bauer@uni-rostock.de
¶johannes.goltz@uni-rostock.de
‖simeon.wiedenmann@uni-rostock.de

***Abstract*—IoT devices are usually connected wirelessly to the Internet. Due to the low energy resources, special new protocols are used. One such protocol is Long Range Wide Area Network (LoRaWAN). It was developed with the aim of sending small amounts of data to the Internet as energy-efficiently and robustly as possible. Transmission security played a major role in the specification of LoRaWAN. Wireless LoRaWAN devices communicate via radio with gateways, which send the data packets to a network server connected to the Internet. The network server has interfaces to be connected to existing IoT platforms and applications. Wireless transmission allows a potential attacker many additional attack vectors. We assess the security mechanisms defined in the LoRaWAN specification and describe own research to show, whether those security mechanisms are sufficient. For this, we explain typical attacks on LoRaWAN radio-based networks. We further show which precautions are necessary not to undermine these measures and whether additional security measures beyond the specification may be necessary. An important contribution of this paper is to present the key features of the LoRaWAN specification in a more generally understandable manner.**

***Index Terms*—LoRaWAN, IoT Security, Network, Protocol**

Acknowledgement.

Simeon Wiedenmann is funded through a grant by the German Federal Ministry of Economics and Energy on the basis of a resolution of the German Bundestag.

I. Introduction.

There are a number of competing protocols and procedures designed specifically to connect IoT devices to the Internet. When deciding on a protocol, various aspects must be taken into account. For an application in the field of building automation, we were looking for a technology that makes it possible to transfer data securely against being spied on by third parties and to prevent attackers from feeding false control commands into the data transmission. LoRaWAN [12] is an evolving protocol and technology specifically designed to connect small devices with the Internet. Typical applications include IoT technology, industrial automation, meter reading transmission and remote control. LoRa is the underlying radio technology. It was developed to achieve reasonable ranges with low error rates and minimal energy consumption even in urban areas. It is able to transmit small data packets over long distances of typically 2 to 10 km with low bandwidth up to 125 kbit/s.

A. Motivation.

Depending on the application, sensitive data can be transmitted or safety-critical processes can be triggered remotely. Every new technology raises the question of how secure it is. We deal in detail with all phases from the development of an application via the registration of devices in a LoRaWAN to the ongoing operation and decommissioning of devices.

In this publication we describe and review the security mechanisms of LoRaWAN during the entire lifetime of a system of IoT devices, network and server-based application. We check whether the security mechanisms defined in the LoRaWAN specification are sufficient, which precautions are necessary not to undermine these measures and whether additional security measures may be necessary.

B. Potential for attackers.

Devices based on LoRa are inexpensive to procure. This applies to both mobile devices and gateways to the Internet. This greatly reduces the costs for a potential attacker who wants to access via the radio interface. In

addition, a complete LoRaWAN system consists of various complex components. Intermediary components between the mobile node and the actual server-based application are often provided by third parties who forward the data in both directions. LoRaWANs are a worthwhile target for attackers for a number of reasons:

- The cost of sending and receiving messages is extremely low. End-devices are freely available. The radio frequencies utilized can be used by anyone without a license.
- Due to the nature of a radio-based protocol and due to the ranges in urban areas, potentially many LoRaWAN subscribers can be reached from a well chosen location. An attacker has potential access to the radio traffic of several stations.
- The risk of being discovered and tracked is negligible for the attacker. The risk becomes almost zero if the attacker just listens.
- LoRaWAN can potentially be used for a number of security-critical applications in which an attacker might be interested.

This list shows that LoRaWANs should not be built and used thoughtlessly.

C. Further structure of this paper.

In Section II we give a short overview of the general structure of a LoRaWAN. We introduce the entities involved in the communication and show which interfaces exist between them. Section III explains the basic security features in all main phases of the communication process as provided in the LoRaWAN standard. The cryptographic methods used in LoRaWAN are described in brief in Section IV. We have outlined a number of possible attacks and planned defense measures in Section V. We present a review of other publication regarding the security of LoRaWAN in Section VI before we conclude our findings in the final section of this paper.

II. General Structure of a LoRaWAN.

Figure 1 depicts the basic minimal architecture of a LoRaWAN backend infrastructure. Figure 2 shows a more complex setup which allows roaming end-devices.

The End-device is a potentially mobile device carrying sensors or actuators. It is connected with the Network Server via the radio gateway. The radio gateway is fully transparent when we focus on security of LoRaWAN communication. The Network server controls access to the network and ensures that data telegrams are sent to the respective correct Application server. The Join Server helps to create non-repeating nonces that are important for cryptographic processes. The

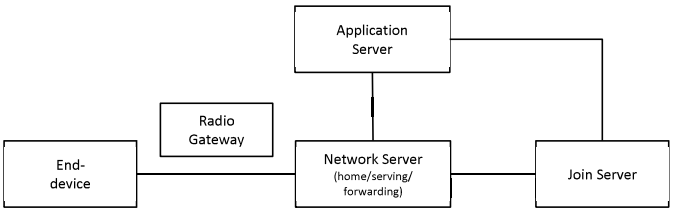

Fig. 1. LoRaWAN Network Reference Model (NRM), End-Device at home - from [2].

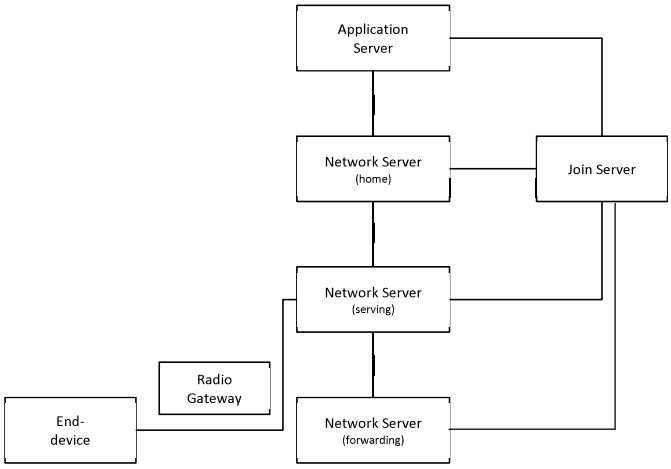

Fig. 2. LoRaWAN Network Reference Model (NRM), roaming End-Device - from [2].

Application Server contains the program logic and is connected to the Internet in IoT scenarios.

Figure 3 shows a typical LoRa-enabled device, in particular a developer board with an ESP32 CPU, WiFi, Bluetooth, several GPIOs, a Micro USB interface and a small OLED display in addition to the LoRa transceiver. Sizes of actual devices may be much smaller.

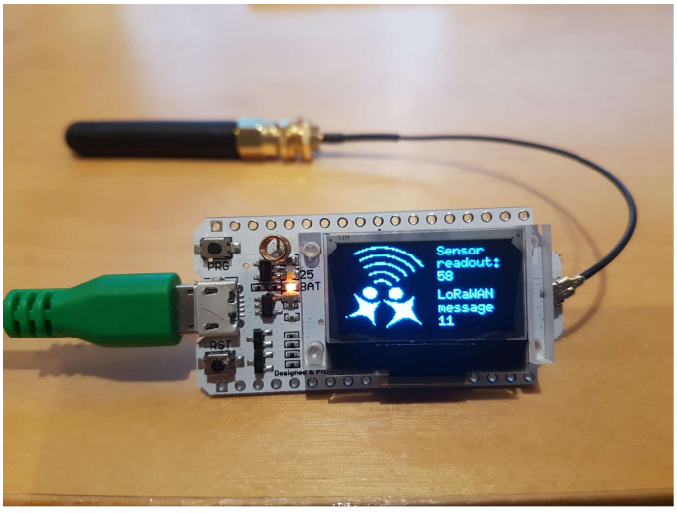

Fig. 3. A typical LoRa-enabled device for development.

LoRaWAN defines three device classes (see [12] section 2.1). These are Bi-directional end-devices **(Class A)** that are not always online and ready to receive messages only after they have transmitted own packets, Bi-directional end-devices with scheduled receive windows **(Class B)** that are ready to receive at scheduled times, and Bi-directional end-devices with maximal receive windows **(Class C)** which are almost always available for receiving transmissions. For Class A devices, communication always starts from the client; for Class B and Class C devices, communication can also start from the network server. This is used, for example, where actuators need to be controlled with commands sent by an Application Server. All end-devices must implement Class A features. The choice of the specific device class has no influence on the safety mechanisms.

III. Security during typical phases of communication.

In a typical scenario the end-device will initiate the communication. Uplink messages are sent by end-devices to the Network Server relayed by one or many gateways. Following each uplink transmission the protocol requires end-devices to open two short receive windows. Each downlink message is sent by the Network Server to only one end-device and is relayed by a single gateway during those receive windows.

LoRaWAN distinguishes between 8 different MAC message types: Join-request, Rejoin-request, Join-accept, unconfirmed data up/down, confirmed data up/down, and proprietary messages (see [12] section 4.2.1).

A. Initial end-device activation.

In order to be able to participate in a LoRaWAN network, each end-device must be personalized and activated. During personalization, two device specific keys are assigned and stored to the end-device ([12] section 6.1.1.3).

The **NwkKey** or actually the session keys derived from it are used to encrypt the communication between the end-device and the Network server and to protect it from changes using the Message Integrity Code (MIC). The NwkKey is device specific and must be stored on the end-device and in the Network server. The uniqueness of the NwkKey for specific end devices can be abandoned by the network operator in favor of simplified network participation (starting with the LoRaWAN 1.1 specification). In this case, the NwkKey will only be used as a participation grant for the network. The integrity and confidentiality of application data is then ensured exclusively via the AppKey. In a community network such as the widespread *The Things Network* [7], potentially many people know the NwkKey.

For this reason, another key was introduced to protect the privacy of payload. The **AppKey** and the session key derived from it are used to encrypt and secure the data between the end device and the application end-to-end. This key is also unique to each end-device.

An end-device can be activated in two ways, either via **Over-The-Air Activation (OTAA)** or via **Activation By Personalization (ABP)**. The latter procedure is not considered in detail here, since all necessary session keys are stored directly in the end-device. The subsequent encryption and authentication of the user data is identical for both methods.

When a device joins a network through over-the-air activation, the NwkKey is used to derive the **FNwkSIntKey** (Forwarding Network session integrity key, [12] section 6.1.2.2), **SNwkSIntKey** (Serving Network session integrity key, [12] section 6.1.2.3) and **NwkSEncKey** (Network session encryption key, [12] section 6.1.2.4) session keys, and AppKey is used to derive the **AppSKey** (App Session Key, [12] section 6.1.2.5). Figure 4 shows how session keys are being derived from the two pre-configured keys NwkKey and AppKey.

The **Join-request** and **Join-accept** handshake is depicted in Figure 5. The end-device sends a Join-Request to the network. After this request has been received from the network, it is forwarded to the corresponding network server. The DevEUI is used by the network to determine the corresponding Network Server. The Join-Server is determined using the JoinEUI in the Join-request. The Join-Server supplies a JoinNonce that is used only once in an end-device's lifetime for the particular Joint-request that is currently being processed.

To join a network, an end-device needs the following information beside the two device root keys (AppKey and NwkKey) ([12] section 6.1.1):

- **JoinEUI** The JoinEUI is a global application ID in IEEE EUI-64 address space [4] that uniquely identifies the Join Server for a particular end-device wishing to join the network. The Join Server maintains device specific counters and assists in the derivation of keys.
- **DevEUI** The DevEUI is, as the name suggests, a global end-device ID in IEEE EUI-64 address space that uniquely identifies the end-device. Currently, there is no mechanism to ensure uniqueness. Hence, the network provider has to maintain its own list. Recurring DevEUIs in other networks cause no problems.
- **DevNonce** DevNonce is a counter that starts at 0 when the device is first turned on and incremented on every join-request. A DevNonce value shall never be reused for a particular JoinEUI value. If

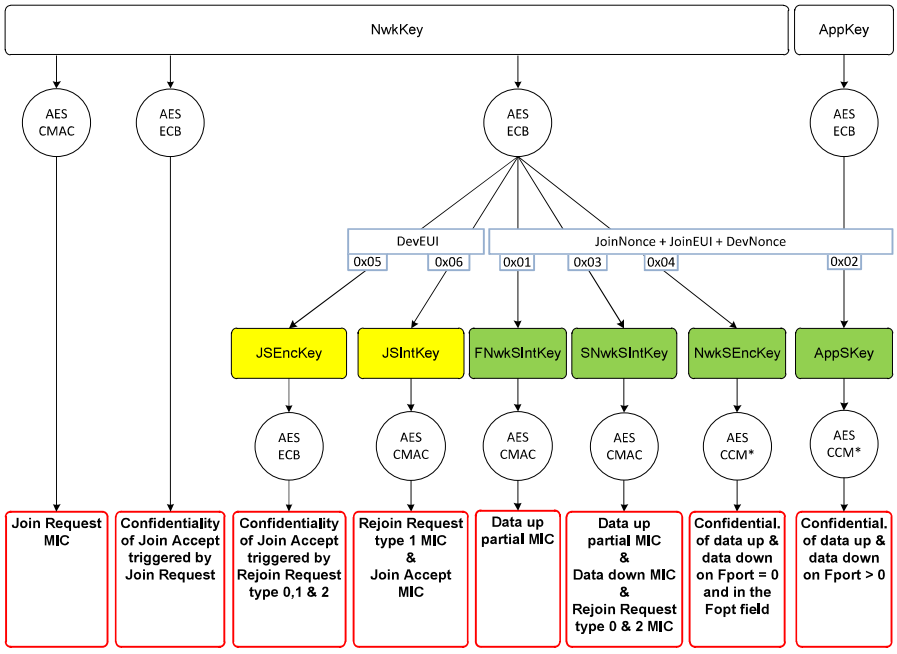

Fig. 4. LoRaWAN1.1 key derivation scheme - from [12].

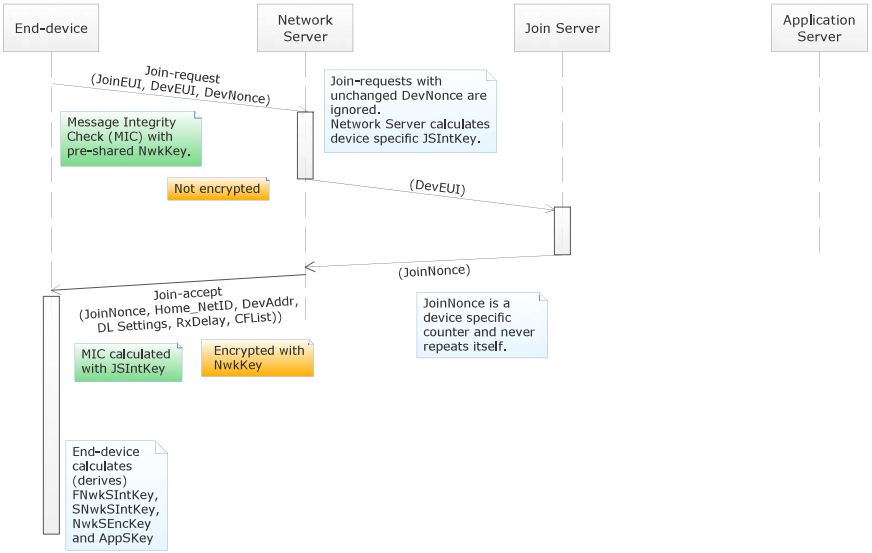

Fig. 5. OTAA join-request and join-accept.

the device can be power-cycled, DevNonce should be persistent (in a non-volatile memory). Resetting DevNonce without changing JoinEUI causes the network server to reject the device's join-requests. For each device, the network server tracks the last DevNonce value used by the device and ignores join-requests if DevNonce is not incremented.

The Network Server derives a **JSIntKey** from the NwkKey and the DevEUI. JSIntKey is used to calculate the Message Integrity Check (MIC) of Join-accept messages[1].

The Join-accept message contains the following information ([12] section 6.2.3):

- **JoinNonce** The JoinNonce is a device-specific counter value (which never repeats itself) provided by the Join Server and used by the end device to derive the session keys FNwkSIntKey, SNwkSIn-

[1]Please note that there are differences in the encryption and MIC of Join-accept messages, depending on whether they were sent in response to Join-requests or Rejoin-requests.

tKey, NwkSEncKey and AppSKey. JoinNonce is increased with every Join-accept message.
- **home_NetID** The field home_NetID of the Join-accept message corresponds to the NetId of the device's Home network.
- **DevAddr** The DevAddr consists of 32 bits and identifies the device in the current network. The DevAddr is assigned by the Network server of the end-device.

The following session keys are derived from pre-shared keys and other information as stated below and are used while communicating after activation ([12] section 6.1.2):

- **FNwkSIntKey** The FNwkSIntKey is a network session key specific to the end-device. It is used by the end-device to calculate the MIC or part of the MIC of all uplink data messages to ensure data integrity. The FNwkSIntKey is derived from NwkKey, JoinNonce, NetID, and DevNonce.
- **SNwkSIntKey** Serving Network session integrity key. The SNwkSIntKey is a network session key specific to the end-device. It is used by the end-device to verify the MIC of all downlink data messages to ensure data integrity and to calculate half of the uplink messages MIC. The SNwkSIntKey is derived from NwkKey, JoinNonce, NetID, and DevNonce.
- **NwkSEncKey** The NwkSEncKey is a network session key specific to the end-device. It is used to encrypt and decrypt uplink and downlink MAC commands that are transmitted as payloads. In LoRaWAN 1.0 which does not define a separate AppKey the NwkSEncKey was also used to encrypt data payload for uplink and downlink. The NwkSEncKey is derived from NwkKey, JoinNonce, NetID, and DevNonce.
- **AppSKey** The AppSKey is an application and device-specific key for the end-device. It is used by both the Application server and the end-device to encrypt and decrypt the payload field of application-specific data messages. Application payloads are encrypted between the end-device and the Application server, but they are protected against alteration (integrity) only in a hop-by-hop manner: one hop between the end-device and the Network server and the other hop between the Network server and the Application server. The AppSKey is derived from AppKey, JoinNonce, NetID, and DevNonce.

Secure provision, storage and use of root keys NwkKey and AppKey and all session keys derived from them on the end device and the back end are an essential component of the overall security of the solution. The corresponding implementation is not specified by the LoRaWAN specification. However, external, tamper-proof keystores are proposed by the specification.

B. Rejoin.

A device can periodically send a rejoin-request message. This enables the back end to periodically initialize a new session context for the end-device. For this purpose, the network responds with a join-accept message.

There are three different rejoin-request messages defined by the standard ([12] section 6.2.4):

- **Type 0** Resets all radio parameters including DevAddr and all session keys. The rejoin-request contains NetID and DevEUI. Additionally a counter RJcount0 is included in the request.
- **Type 1** Restores a lost session context similar to an initial join-request. The rejoin-request contains JoinEUI and DevEUI. Additionally a counter RJcount1 is included in the request.
- **Type 2** Resets all session keys but no radio parameters. The rejoin-request contains NetID and DevEUI. Additionally a counter RJcount0 is included in the request.

The following Figures 6 and 7 illustrate the message handshakes after rejoin-requests for all three types of rejoins. Please note that RJCount0 or RJCount1 replace the DevNonce in all requests and key derivation. Type 0 and Type 2 requests are handled entirely by the network server. Type 1 requests require the Join Server to provide a new JoinNonce while the JoinNonce remains unchanged for Type 0 and Type 2. On its own discretion the network server can either answer with a join-accept message or a regular downlink frame.

C. Data up and downlink.

All end-devices (Class A, B, C, see section II) can send uplink frames at any time, but need to respect a very low duty cycle required by the standard. End-devices are open to receive downlink frames shortly afterwards. Frames can contain data, Media Access Control (MAC[2]) commands, or both. If a data frame carries a payload, the payload itself (field FRMPayload) must be encrypted. The encryption scheme used is based on the generic algorithm described in IEEE 802.15.4/2006 Annex B [1].

The **AppSKey** will be used for encryption of payload as mentioned in section III-A and shown in Figure 8.

[2]Please note, that MAC does not mean the cryptographic term *Message Authentication Code* here. The LoRaWAN standard [12]uses the term Message Integrity Code for cryptographic purposes. Actually MIC values are the first 4 bytes of a CMAC. See section IV.

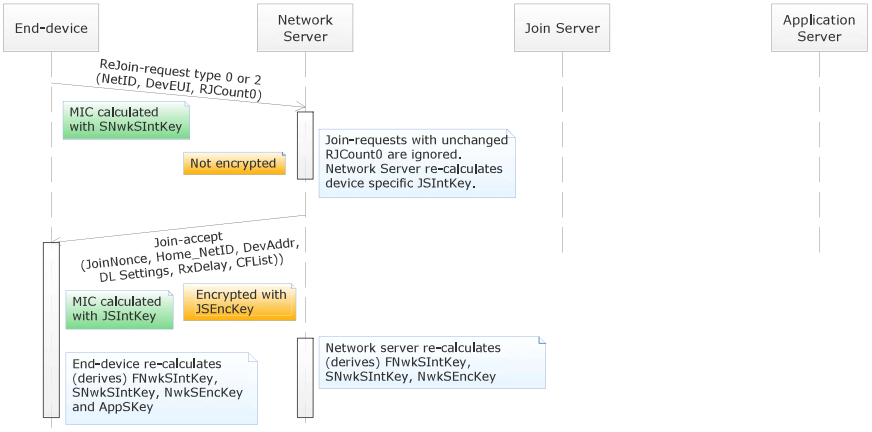

Fig. 6. OTAA rejoin-request type 0 or type 2 and join-accept.

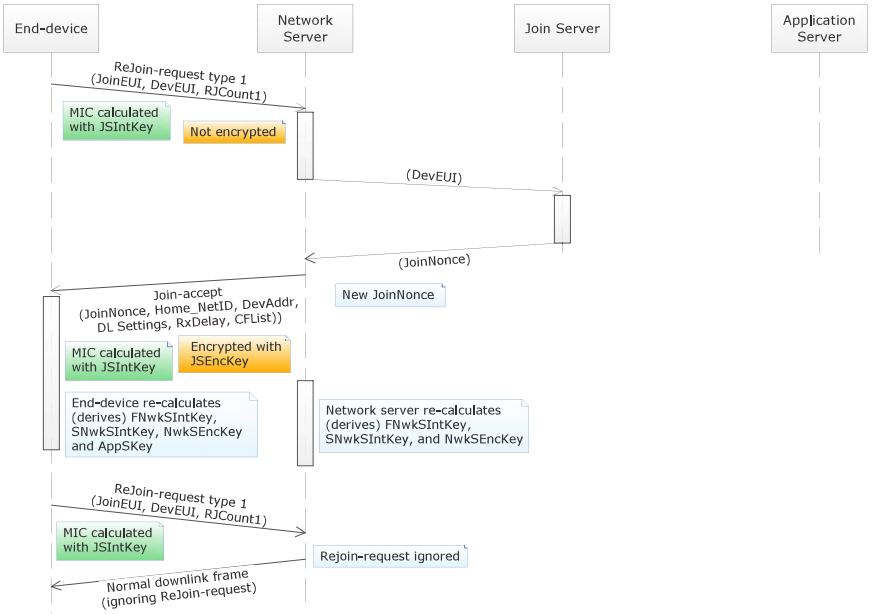

Fig. 7. OTAA rejoin-request type 1 and join-accept.

Message integrity is not guaranteed on an end-to-end basis, only in two steps, the first MIC step between end-device and network server, The second step between network server and application server is considered to be out of scope for this paper as it relies on standardized network security principles and implementations such as SSL/TLS for example.

The MIC (see [12] section 4.4) of downlink frames is calculated with the AES-128 CMAC algorithm as described in RFC4493 [13]. The SNwkSIntKey is used to sign a message block containing some constants, two counters ConfFCnt and (AFCntDwn or NFCntDown), the DevAddr, and the length of this message.

The fields in this message block have the following meaning:

- **ConfFCnt** For downlink frames in reply to confirmed uplink frames, ConfFCnt is the frame counter value modulo 2^{16} of the "confirmed" uplink frame that is being acknowledged. In all other cases ConfFCnt = 0x0000.
- **DevAddr** The DevAddr is assigned by the Network server of the end-device at the time of activation over the air or pre-assigned during Activation by Personalization.
- **AFCntD(o)wn**[3] Frame counter for downlink direction (towards the end-device). This counter is

[3]Spelled with and without 'o' in [12].

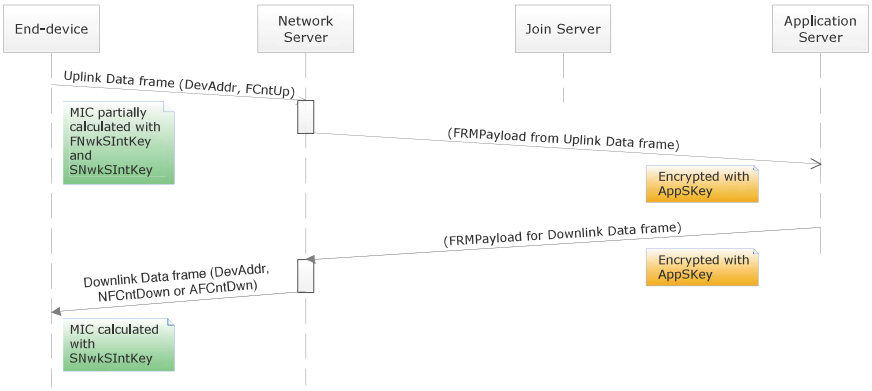

Fig. 8. End-to-end encryption between End-device and Application Server.

used when the end-device works in LoRaWAN1.1 mode for all FPort values except 0. (FPort enables application-specific handling of received frames. An FPort value of 0 indicates that this frame contains MAC commands only. Frames with an FPort value of 0 shall be processed by the LoRaWAN implementation, not the application).

- **NFCntDown** Frame counter used similar to AFCntDown, when FPort value is 0 or FPort is not present (according to LoRaWAN1.0 specification).

The MIC of uplink frames is calculated with a slightly more complicated process. This MIC contains of two separately calculated blocks. The first block (B0) contains some constants, the DevAddr, and the frame counter FCntUp. The second block (B1) contains some constants, the ConfFCnt, TxDr, TxCh, the DevAddr, the frame counter FCntUp and a value representing the actual length of the message.

The fields in these two message block have the following meaning:

- **FCntUp** (B0 and B1) Frame counter for uplink frames.
- **DevAddr** (B0 and B1) see section III-C.
- **ConfFCnt** (B1) see section III-C.
- **TxDr** (B1) TxDr is the data rate used for this transmission.
- **TxCh** (B1) TxCh is the channel used for this transmission.

Block B0 and the message are integrity protected with FNwkSIntKey as follows: cmacF = aes128_cmac(FNwkSIntKey, B0 | msg). Block B1 and the message are integrity protected with SNwkSIntKey as follows: cmacS = aes128_cmac(SNwkSIntKey, B1 | msg) (see section III-A for a description of how these session keys are derived).

The 4 byte MIC is calculated by concatenating the first two bytes of cmacS and the first two bytes of cmacF.

MICs are calculated after payload encryption. This allows the network server to check the integrity without the help of any other entity, and hence, without knowing the plaintext of the payload.

IV. Cryptography in LoRaWAN.

We assume that the basic cryptography used in LoRaWAN is sufficiently secure. These include the AES128 Cipher-based Message Authentication Code (CMAC), AES128 Counter with CBC-MAC (CCM), and AES128 Electronic Codebook (ECB). However, without thorough examination we cannot presume that the preconditions of these cryptographic procedures, which are necessary for a secure working mode, are fulfilled.

The simplest of the encryption modes is the Electronic Codebook **(ECB)** mode. The message is divided into blocks and each block is encrypted separately. Identical message blocks are also encrypted in the same way. An exchange of blocks in the ciphertext leads to the same exchange of the blocks in the decrypted message. An error in a block only affects the decryption of this block. The use of ECB mode is therefore not recommended, unless a single message block is to be encrypted only once [10]. Figure 9 shows ECB schematically.

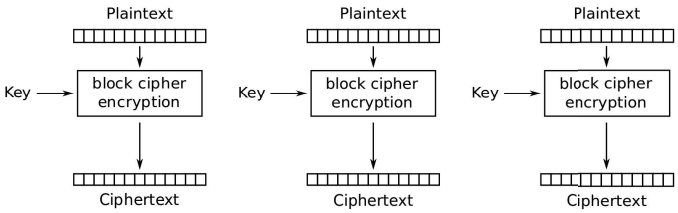

Fig. 9. Encryption in ECB mode.

The Counter with CBC-MAC mode **(CCM)** is an operating mode for block ciphers developed by Russ

Housley, Doug Whiting and Niels Ferguson. CCM turns a block cipher into an authenticated encryption process designed to guarantee both confidentiality and integrity. RFC3610 [15] specifies CCM only for block ciphers with a block length of 128 bits, such as AES. With CCM, an initialization vector (IV) must not be used twice with the same key. This is because CCM is derived from Counter Mode (CTR), and CTR is a stream cipher. Figure 10 shows CTR schematically. A proof of security under certain circumstances can be found in [9]. LoRaWAN uses a slightly modified CCM* as defined in [1].

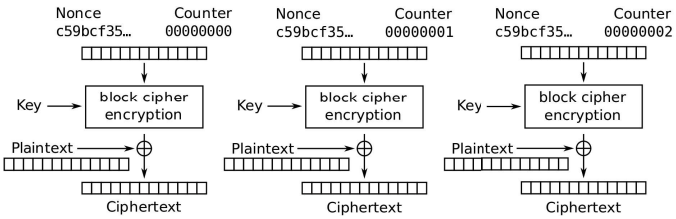

Fig. 10. Counter mode encryption.

The special feature of Counter Mode in comparison to other operating modes is the fact that the initialization vector consists of a new random number (Nonce) to be selected for each cipher frame, linked to a counter, which is incremented with each further block. The link can be made, for example, by concatenation (append), addition or XOR.

Figure 11 shows the Cipher Block Chaining (CBC) Mode. For integrity checks (CBC MAC) the initialization vector is set to zero and the last block encrypted with CBC is considered the Message Authentication Code (MAC) [5].

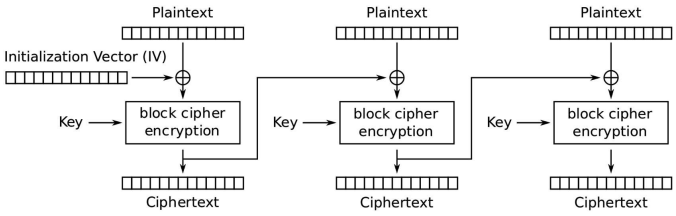

Fig. 11. Encryption in CBC mode. If CBC is used for integrity protection, the initialization vector is set to zero and the last block encrypted with CBC is appended as MAC (the so-called CBC-MAC or CBC residual value) to the original unencrypted message and sent together with this MAC [5].

The core of the **CMAC** algorithm is a variant of CBC-MAC proposed and analyzed by Black and Rogaway under the name XCBC [6][8].

V. Attacks

For our investigation we assume the following attacker profiles:

- Attacker A is able to eavesdrop the entire LoRaWAN radio traffic by placing it's own receiver between end-device and gateway.
- Attacker B has the capabilities of attacker A and can additionally send any frame at any time.
- Attacker C is able to analyze individual end-devices (not all of them) and read the keys stored in them.
- Attacker D knows the NwkKey, but not the AppKey, which is typical for a global network such as The Things Network.
- Attacker E is able to freely manipulate the entire backend infrastructure (network server, join server, radio gateway), but not the application server.

We do not consider attackers that are able to manipulate the application server. That would be out of scope for this paper, which looks at LoRaWAN. On the other hand, it would simply be impossible to protect against this attacker with the means of LoRaWAN. Furthermore, a denial of service attack is trivially easy to perform for most radio procedures and is also ignored in this paper.

We assume the following aims of attackers:

- Read encrypted payload between end-device and application server.
- Sending meaningful or meaningless fake messages to an application server.
- Participate in the network without permission. For example, if the network is a paid network, a potential attacker wants to gain free access to the network and the connected application servers.
- Profiling of end-devices. An attacker wants to find out when certain end-devices are active and where they are at that time.
- Misplacing end-devices. An attacker wants end-devices to appear at a wrong location.

A. General attack vectors and countermeasures.

In this section we look at attack vectors that we have collected in a brainstorming process. We show which countermeasures are planned according to the standard.

1) Attacks on key material.: An obvious possibility is to spy out secret key material which is stored in end-devices. NwkKey and AppKey are generated and stored in each end-device during manufacturing or on provisioning. Both keys are specific to a particular end-device, but the NwkKey is often sacrificed to allow easy access to a network. In these cases, NwkKey is shared by all end devices and the AppKey provides eavesdropping security between the end-device and the application. In the event that the session keys were generated in advance as part of an Activation By

Personalization (ABP) process, the very same applies to the session keys (see section III-A).

2) Attacks on counters and nonces.: If the session keys were generated within the OTAA, the session keys are generated from NwkKey and AppKey and a series of counters and nonces. As shown in Section IV, it is important for the cryptographic methods AES-CCM and AES-CMAC that counter and initialization vectors are not repeated. The Join Server guarantees that the JoinNonce is not repeated, the end-device guarantees that the DevNonce or the RejoinCounters RJCount0 and RJCount1 are not repeated.

Due to the length of the counters and the slowness of the transmissions, it is practically hopeless for an attacker to wait for a repetition of a counter value. In addition, at least according to the protocol definition, it is forbidden for counters to be repeated during the lifetime of end-devices. Hence, attackers would have to provoke a repetition of a counter. This attack can only be carried out by the attackers we refer to as Attacker B in Section V. Since the counters themselves cannot be determined by radio traffic due to strong encryption, this attack also requires the ability to read or manipulate the corresponding counters either on the end-device or the network server. An attacker who can do this, however, can save himself an attack on the radio procedure.

Especially problematic would be the repeated use of the CCM* encryption method with the same key, nonce and counter (see Sections III-A and IV). The values of the frame counter may only be used once in all transmission with the CCM* operating mode. CCM* is being used for data up and downlink messages. Therefore the reinitialization of an ABP end-device frame counter is prohibited. ABP devices must use a non-volatile memory to store the frame counters, since ABP devices use the same session keys throughout their lifetime. Therefore, it is recommended in the standard to use OTAA devices for higher security applications, which re-generates the session keys at every network join or rejoin.

3) Replay of frames.: Attackers of category B (referred to as Attacker B in Section V) can send any number of messages. In view of the fact that transmissions are cryptographically secured, the question arises whether a repetition of a data packet can nevertheless have negative effects on authenticity and integrity. Data packets are protected against replay attacks by FrameCounter. (Re)JoinRequests are protected by either JoinNonce or RJCount0 respectively RJCount1.

A special form of this attack vector would be if an attacker records a data frame in one location and plays it back in another location in the reception area of another network server.

4) Causing rejoin.: With the help of rejoin requests, corrupted or fake end-devices could require a renegotiation of the session keys. Rejoin requests are sent unencrypted, but are protected against manipulation by SNwkSIntKey or JSIntKey.

5) Shortening, lengthening, and bitwise manipulation of message frames.: Unlike a replay attack or a classic man-in-the-middle attack, this attack is about manipulating data frames during transmission. For example, a transmission can be extended by a few bits. The modulation method used in LoRa and the serial transmission of the bits would not prevent this. It would also be possible to change individual bits by means of targeted, stronger emissions or to suppress them by noise. However, due to the signature procedures used, the attacker would have to be in possession of the signing key or session key used for the respective message type.

6) Profiling: It is in the nature of things that LoRaWAN end-devices send data at least occasionally. The location of the devices can be determined during transmission. An attacker can use three receivers with known location and trilateration to estimate the location based on the distance-dependent field strength (specified as RSSI for Received Signal Strength Indicator).

Especially in LoRaWAN installations with many radio gateways this works very well. Uplink messages are sent by end-devices to the Network Server relayed by one or many gateways. The operator of an application server receives the field strength measurements virtually free of charge. In the global LoRaWAN The Things Network, which has already been mentioned several times, such a report looks like this:


```
{
  "time": "2018-06-03T08:13:32.680499764Z",
  "frequency": 867.9,
  "modulation": "LORA",
  "data_rate": "SF7BW125",
  "coding_rate": "4/5",
  "gateways": [
    {
      "gtw_id": "eui-b827ebfffe2a509f",
      "timestamp": 2996109116,
      "time": "2018-06-03T08:13:32.656102Z",
      "channel": 7,
      "rssi": -79,
      "snr": 8.2,
      "latitude": 54.085,
      "longitude": 12.13314,
      "altitude": 40
```

```
    },
    {
      "gtw_id": "eui-b827ebfffe37dc3b",
      "timestamp": 2322081132,
      "time": "2018-06-03T08:13:32.65646Z",
      "channel": 7,
      "rssi": -37,
      "snr": 8.8,
      "latitude": 54.09296,
      "longitude": 12.09826,
      "altitude": 30
    }
  ]
}
```


The same transmission was received by two radio gateways. Both gateways have inserted a precise time stamp in addition to the aforementioned RSSI value. If the time stamp comes from a high-precision clock, such as a GPS receiver, position data can also be derived from transit time measurements. Many LoRa gateways already contain a GPS receiver to deliver these time stamps.

VI. Critical consideration of other publications regarding the security of LoRaWAN.

In this section we give an overview of other publications on the topic "Security of LoRaWAN" and review some of them.

The authors of the well cited [11] claim that there is a security hole during the join process. The authors assume wrong assumptions and obviously have not read the standard. For example, they claim that join accept messages are encrypted with the AppKey in their Figure 1. In their section II-B they completely hide the role of the join server to prevent replay attacks.

A better overview of the security of LoRaWAN can be found in [3]. The authors also work on the basis of version 1.1 of the standard. They also explain the importance of secure storage of the NwkKey and AppKey keys and explain replay and wormhole attacks. Like us, the authors were not able to show a concrete gap either.

A detailed statistical analysis of the probability of collisions of nonces and counters during the Join procedure can be found in [14]. The problems shown are not due to the standard, but to the implementation of random number generators in the chipsets frequently used for LoRa.

VII. Conclusion.

After a detailed examination of the LoRaWAN standard and other publications on this topic, we found no obvious security gap. Of course, this does not mean that they do not exist. It could just be an inability of ourselves. Nevertheless, we do not currently see any concerns when using LoRaWAN, as long as the safety measures already mentioned in the standard are observed and implementations are carefully "crafted". We consider LoRaWAN to be suitable for our needs in an IoT environment.

References

[1] "IEEE Standard for Information technology – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, Sept 2006.

[2] Alper Yegin (Actility), Editor, *LoRaWAN Backend Interfaces Specification - Version 1.0*, LoRa Alliance, Inc., 3855 SW 153rd Drive, Beaverton, OR 97003, USA, Oct 2017.

[3] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, "Exploring The Security Vulnerabilities of LoRa," in *Cybernetics (CYBCONF), 2017 3rd IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[4] I. S. Association *et al.*, "Guidelines for 64-bit global identifier (eui-64 (tm))," *Registration Authority Tutorials, April*, 2010.

[5] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[6] J. Black and P. Rogaway, "A suggestion for handling arbitrary-length messages with the cbc mac," in *NIST Second Modes of Operation Workshop, August*, 2001.

[7] N. Blenn and F. Kuipers, "LoRaWAN in the wild: Measurements from the things network," *arXiv preprint arXiv:1706.03086*, 2017.

[8] M. J. Dworkin, "Recommendation for block cipher modes of operation: The cmac mode for authentication," Tech. Rep., 2016.

[9] J. Jonsson, "On the security of ctr+ cbc-mac," in *International Workshop on Selected Areas in Cryptography*. Springer, 2002, pp. 76–93.

[10] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[11] S. Na, D. Hwang, W. Shin, and K.-H. Kim, "Scenario and countermeasure for replay attack using join request messages in LoRaWAN," in *Information Networking (ICOIN), 2017 International Conference on*. IEEE, 2017, pp. 718–720.

[12] Nicolas Sornin (Semtech), Editor, *LoRaWAN Specification - Version 1.1*, LoRa Alliance, Inc., 3855 SW 153rd Drive, Beaverton, OR 97003, USA, Oct 2017.

[13] J. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm," Internet Requests for Comments, RFC Editor, RFC 4493, June 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4493.txt

[14] S. Tomasin, S. Zulian, and L. Vangelista, "Security analysis of lorawan join procedure for internet of things networks," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2017 IEEE*. IEEE, 2017, pp. 1–6.

[15] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," Internet Requests for Comments, RFC Editor, RFC 3610, September 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3610.txt