# Security Vulnerabilities in LoRaWAN

Xueying Yang*, Evgenios Karampatzakis†, Christian Doerr*, and Fernando Kuipers*
*Delft University of Technology
2628 CD, Delft, The Netherlands
Emails: yangxueying0910@hotmail.com, C.Doerr@tudelft.nl, F.A.Kuipers@tudelft.nl
†Brightsight
2628 XJ, Delft, The Netherlands
Email: karampatzakis@brightsight.com

***Abstract*—LoRaWAN is a MAC-layer protocol for long-range low-power communication. Since its release in 2015, it has experienced a rapid adoption in the field of Internet-of-Things (IoT). However, given that LoRaWAN is fairly novel, its level of security has not been thoroughly analyzed, which is the main objective of this paper. We highlight the security features present in LoRaWAN, namely activation methods, key management, cryptography, counter management, and message acknowledgement. Subsequently, we discover and analyze several vulnerabilities of LoRaWAN. In particular, we design and describe 5 attacks: (1) a replay attack that leads to a selective denial-of-service on individual IoT devices, (2) plaintext recovery, (3) malicious message modification, (4) falsification of delivery reports, and (5) a battery exhaustion attack. As a proof-of-concept, the attacks are implemented and executed in a controlled LoRaWAN environment. Finally, we discuss how these attacks can be mitigated or protected against.**

***Index Terms*—LoRaWAN, security, replay attack, eavesdropping, bit flipping, ACK spoofing.**

## I. Introduction

The fundamental value proposition of the Internet-of-Things (IoT) is to enable new value cases by remotely monitoring and controlling distributed embedded systems, which together with their low cost will result in pervasive deployments. It is predicted that there will be 13.5 billion connected objects in use by 2020 [1].

When new networked devices are monitoring and able to interact with our environment in a pervasive density, security against malicious use becomes paramount. Unfortunately, at present, IoT devices are easily exploited by attackers, because many of these devices are shipped with insecure defaults or insecure, remotely exploitable code [2]. As a result, IoT products have already been used to launch major distributed denial-of-service attacks [2].

To reach a mature level of IoT security, many challenges need to be overcome. For example, there is a lack of standards for secure IoT development. Also, there is no accepted reference architecture among vendors. Moreover, IoT products and services need cooperation of many technologies and protocols, making security of IoT even harder to be guaranteed [3]. Other challenges include IoT product deployment in insecure or exposed environments and resource constraints in embedded systems, which may limit security options [4].

Aside from a lack of standards and reference architectures, also the protocols and methods with which IoT are to be networked to the cloud are still under heavy development. Low-Power Wide-Area Network (LPWAN) technologies are designed to connect IoT devices with low-power requirements, and at long range and low cost. The Long-Range Wide-Area Network (LoRaWAN) is a new MAC-layer protocol in the family of LPWANs. It is based on LoRa radio technology, which is a chirp-spread-spectrum type of wireless modulation. The first LoRaWAN specification was released in Jan. 2015 by the LoRa Alliance and ever since LoRaWAN has seen a steep adoption curve.

The popularity of LoRa has sparked a lot of research, but mostly from a performance perspective, e.g., see [5]. To the best of our knowledge, we have been the first to study the security of the LoRaWAN protocol stack and its vulnerabilities in a systematic way. We provide a vulnerability analysis, outline several possible attacks and describe security solutions for LoRaWAN.

This paper is organized as follows. Section II provides an overview of related work. Section III summarizes the key security features present in a LoRaWAN. Section IV introduces several possible attacks against a LoRaWAN. Moreover, proof-of-concept experiments are conducted to demonstrate those attacks in a controlled environment. Section V presents suggestions to mitigate or prevent the discussed attacks. Section VI concludes our work.

## II. Related work

Although a number of research groups have looked at the performance characteristics of LoRaWAN, to date no work has systematically analyzed the security of the LoRaWAN protocol stack.

Miller [6] provides a brief overview of LoRaWAN security and outlines how to configure the security features in the protocol to set up a LoRaWAN. He describes the location of the key material in a LoRaWAN setup, and alerts that flaws in key management could compromise a backend. The work does however not analyze the protocol nor evaluates the security of message exchanges.

A notorious problem in protocol security is the insufficient use of randomness or nonces (“number used once”). Zulian et al. [7], [8] analyze security threats in LoRaWAN by focusing on the generation of DevNonce, which is used in the join request. They study the randomness of DevNonce and provide alternative generation methods. Also Na et al. [9] focus on vulnerabilities in the join procedure in LoRaWAN. Michorius [10] investigates the issue of privacy leakages if multiple users attempt to access the same application server. He compares encryption algorithms and modes based on time-to-compute and resources used. Naoui et al. [11] compare existing key management protocols for IoT, and propose to add proxy nodes that drive a reputation system to enhance the security mechanisms of LoRaWAN. Aras et al. [12] highlight security issues at the physical layer, and develop practical attacks around selective jamming. Countermeasures and suggestions are given to mitigate these attacks. Lee et al. [13] analyze the potential of a bit-flipping attack.

Although some research has been done on LoRaWAN security, an analysis of the communication and enrollment protocol from a security analysis has not been done. In this paper, we fill this gap. The next section will describe the protocol design of LoRaWAN by the standard; the section after this will present the results from our security analysis.

## III. Security Features of LoRaWAN

This section provides an overview of the security features of the LoRaWAN protocol, specification 1.0.2. Instead of reiterating the standard by function, we will review the specification from a functional perspective. Following the classical definition of security, we care about the confidentiality, integrity and availability of a system. As the communication channel is wireless and thus available to anyone for injection and modification, also the authenticity of communication – in other words do the packets indeed originate from the alleged source – and the protection against originally legitimate but maliciously re-injected traffic become a concern. Finally, in an IoT application context we also need to raise the issue of device and key enrollment. As IoT devices are produced and stored in bulk and shipped to an end-user who would like to connect a device to his or her account, downloading a device/user/application-specific key to the unit becomes necessary. In the following, we will describe the approach LoRaWAN takes on each of these aspects.

### A. *Channel Confidentiality*

LoRaWAN v1.0.2 uses a pair of two distinct keys, the network key NwkSKey, and the application key AppSKey. The reason behind this is that LoRaWAN was designed with the business case of a network or telecom operator in mind, who deploys and operates the LPWAN, while IoT device owners and IoT service providers may use the infrastructure as a black box to establish connectivity between them. As confidentiality and integrity checking is required with different data scopes on the air interface between the IoT device and the network infrastructure (owned by the telco), and between the IoT device and a third-party application provider in the backend, the network key NwkSKey secures the former, while

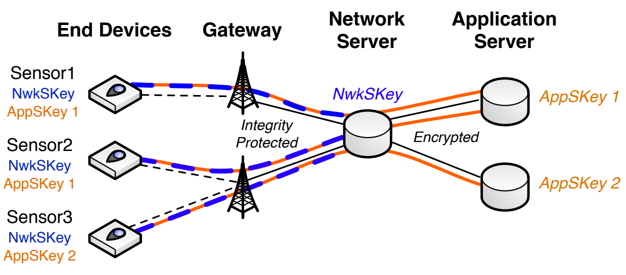


Fig. 1. Key usage in a LoRaWAN.

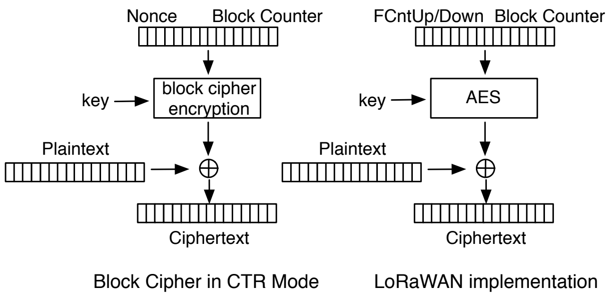


Fig. 2. LoRaWAN uses a AES in counter mode for message encryption.

the end-to-end connection is protected by the application key AppSKey as shown in figure 1.

When a message is sent to the application server, the frame payload is encrypted first by the AppSKey. Data confidentiality is protected by a block cipher operated in counter mode (CTR). This specific construction, shown in figure 2, generates a stream of random bytes using the pseudo-random permutation provided by a block cipher, which is then used as a key stream to encrypt the plaintext by an exclusive OR. As the CTR is thus essentially a stream cipher, the exact key stream may never be used twice. This is realized in counter mode by use of a nonce per connection, and a monotonically increasing block counter to accommodate messages of multiple block lengths. LoRaWAN follows the CTR design to the letter: it chooses AES as a block cipher implementation and uses LoRa's message counter FCntUp or FCntDown as nonce, which is continuously incremented for each message. If the message counter never repeats, then this mode and CTR mode are identical.

### B. *Enrollment Protocol*

As two keys are needed to transmit a frame to the network and application server, this key material needs to be somehow downloaded on each participating IoT device. LoRaWAN specifies two mechanisms for this activation procedure, *Activation by Personalization* (ABP) and *Over-the-Air Activation* (OTAA).

*1) Over-the-Air Activation (OTAA):* An end device will first send a *Join Request*, which contains a 3-byte *DevNonce* – a random number. After the *Join Request* is received by the network server, the network server will check whether the end device can be accepted or not. If the end device is not accepted, there will be no response. If the device is accepted, then the network server will send a *Join Accept* message to the end device. The *Join Accept* contains a 3-byte *AppNonce*, which is generated by the network server. After the *AppNonce* is received by the end device, both sides use the nonces to generate the network and the application keys. As otherwise any eavesdropper would be able to generate NwkSKey and AppSKey, OTAA derives the network and application keys by encrypting the data using the *AppKey*, a 16-byte device-unique key which is assigned by application owners to end devices.

$$\text{NwkSKey} = \text{AES}_E\ (\text{AppKey}, \text{0x01} \,||\, \text{AppNonce} \,||\, \text{NetID} \,||\, \text{DevNonce} \,||\, \text{pad}\ )$$
$$\text{AppSKey} = \text{AES}_E\ (\text{AppKey}, \text{0x02} \,||\, \text{AppNonce} \,||\, \text{NetID} \,||\, \text{DevNonce} \,||\, \text{pad}\ )$$

In order to obtain NwkSKey and AppSKey, the IoT device needs to trade-in the long-term AppKey after commissioning or a volatile reset. OTAA uses unique AppKeys to prevent that after a compromise of one end device the whole network is impaired, and if the system keeps track of past DevNonce replayed join attacks can be deterred. *Join Request* messages are not encrypted, *Join Accept* messages are encrypted after being digitally signed. A signed *Join Accept* message is encrypted using AES in Electronic Codebook (ECB) mode, which means that the plaintext is directly encrypted using the key. This has the disadvantage that identical plaintext messages are encrypted into identical ciphertexts. ECB is generally not recommended as it trivially allows a traffic pattern analysis and breaks the semantic security of a communication system. However, as the join message should never be repeated because of two nonces in the key derivation function of NwkSKey and AppSKey and the strong pseudo-random permutation of AES, the usage of ECB in this instance does not create major complications.

*2) Activation by Personalization (ABP):* ABP skips the exchange of join messages. Before activation, unique parameters – DevAddr, NwkSkey and AppSkey – are assigned to the end device and are stored in the server. When an end device is trying to communicate with the server, it will send messages directly. These messages are encrypted and signed, such that only the corresponding network server can read the message. If devices are setup by ABP, NwkSkey and AppSkey will be used across sessions until updated in the device.

### C. Integrity and Authenticity Validation

A cryptographic message integrity code (MIC)[1] is used in LoRaWAN to provide an integrity check on the MAC header and payload data. The MIC for a data message is calculated using the NwkSkey and AES-CMAC method. When uplink messages arrive at the network server, the server will

[1]In cryptography, we would normally refer to this as a message authentication code or MAC, but due to the confusion with the in networking ubiquitous message access control address or MAC address will refer to them as MICs throughout the paper.

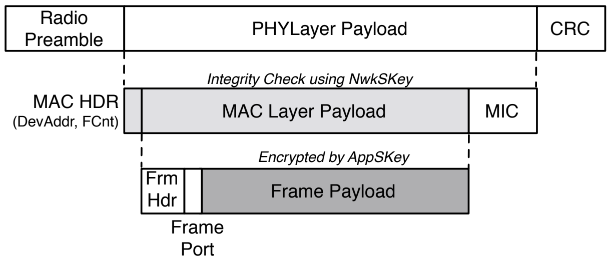


Fig. 3. A message integrity check is computed on the MAC header and payload.

first check the message integrity and, if it passes the check, transfer the message to the application server. For *Join Request* messages, the MIC is generated by using the AppKey instead of NwkSkey. Figure 3 shows the decomposition of the protocol data unit at the physical level into MAC header and payload, and frame contents. The light shaded area is protected by the MIC generated from the NwkSkey, the dark shaded area is encrypted by the AppSKey.

### D. Replay Protection

Counters are an important component in replay protection and, as the message counter is used in LoRaWAN to generate the key stream, are also essential to the confidentiality of the communication channel. For each end device, there are two frame counters named FCntUp and FCntDown. FCntUp is counting uplink messages in the end device, while FCntDown is counting downlink messages in the network server. In order to keep uplink and downlink messages in sync, there is a limit value MAX_FCNT_GAP. If the difference between number of uplink and downlink messages is larger than MAX_FCNT_GAP, subsequent frames will be discarded. Both 16-bit and 32-bit frame counters are allowed in LoRaWAN. If the counter overflows, it will be started from 0 again. According to the LoRaWAN specification, the counter value will be set to zero after resetting.

## IV. Attacks toward LoRaWAN

In this section, we present five vulnerabilities in the protocol and describe actual attacks that could exploit them. For verification, we have implemented all attack vectors in a hardware or software proof-of-concept. The attacks target all three main aspects of communication security: first, we demonstrate that it is possible to eavesdrop and decrypt the content of a frame under certain circumstances. Second, we show that the content of a packet may be modified outside of the integrity check provided by the protocol. Third, we highlight that messages could either be replayed, or a node tricked into believing that a message has been received by the gateway when it actually has not, and outline a battery exhaustion attack. This compromises the availability of the network.

In the following, we will discuss each of these five attacks in detail, and present the exploit and necessary preconditions. All attacks are summarized in an attack tree in the conclusion.

### A. Replay attack for ABP-activated nodes

As discussed during the previous section, the ABP-activated end devices are using static keys which are preprogrammed into the device. Moreover, the protocol specification v1.0.2 states:

> "After a JoinReq - JoinAccept message exchange or a reset for a personalized end device, the frame counters on the end device and the frame counters on the network server for that end device are reset to 0."

Therefore, after resetting, an ABP-activated end device will reuse the frame counter value from 0 with the same keys. In this case, an attacker can grab messages in the last session with larger counter values and reuse it in the current session. Besides resetting, another method to restart the counter is a counter overflow. After the counter value reaches its maximum value, the counter will be reset and will restart from 0. With counter values from the last session and the same session keys, an attacker can also replay previous messages to cut off the communication between the end device and the server. This holds both for ABP and OTAA. However, attacking an ABP-activated end device will take less time as both reset and overflow work if the attacker has the ability to reset end devices.

A message replay is trivial to implement for an adversary. First, monitor and store the uplink messages of an ABP-activated node. Second, wait until the device has reset the counter value FCnt, which is sent in clear text. Assume the uplink counter value of the malicious message is $FCnt_m$, the uplink counter value of the end device is $FCnt_{curr}$, and the maximum accepted counter gap is $Gap$. Third, replay any message with $FCnt_m - FCnt_{curr} \leq Gap$ to fit the running window algorithm of LoRaWAN and thus be accepted by the network if replayed. The most harmful attack is to select the counter value $FCnt_m = Gap + FCnt_{curr}$, since it will take the devices the longest time to recover.

Figure 4 shows an example of a replay attack. Here the maximum counter gap is the default protocol value of 16384. The malicious message is the message in the last session with same device address, session keys and larger counter value. As long as the attacker sends this message in this session to the network server, and it is accepted, the messages from the victim with counter value smaller than 70 will be ignored. For the attack, minimal hardware is required: a traffic sniffer as well as a LoRa transmitter to replay messages. While in a small LoRaWAN with only a few end devices, the attacker may need to wait a long time for a counter overflow, the attack can be efficiently conducted for ABP-activated end devices in a large deployment. Once the attacker gets the largest possible counter value for one end device, it can periodically replay the message and block the end device permanently (or until the session keys of the end device are changed, which requires for ABP a separate channel or physical access). This replay vector thus implements a denial-of-service attack on the availability of an LPWAN deployment.

*Proof-of-Concept experiment*

A replay is possible using any LoRa transceiver to achieve the attack as long as the device is able to transmit and receive LoRa wireless messages. For our proof of concept, we had a sensor node activated with a LoRaWAN provider, and operated a malicious local gateway (250 Euro) as well as a malicious LoRa sensor based on the popular RN2483 chip (10 Euro) to inject the traffic, see figure 5. Both are off-the-shelf components; in principle the attack can also be launched without a dedicated gateway at the expense of less convenience as the gateway can capture concurrent transmissions.

As a heartbeat to demonstrate the success of the attack and validate that the DoS outage matches the predicted value, we had the sensor report a field measurement every thirty seconds via LoRaWAN to a backend server. The malicious gateway would monitor all frequencies in use by LoRa, and complete a dictionary. In the top highlighted area of the gateway trace shown in figure 6, the attacker notices a device reset and simply re-injects a previously saved message (bottom highlighted area), in this case with the counter value 10. As the subsequent frames sent by the legitimate sensor are out of sequence, the sensor will need to increment and transmit until back in sync. As devices obey a specific low-volume duty cycle, the sensor is effectively blocked during this time. The reporting backend of the LoRa application service shown in figure 7 confirms the replay and an outage for 5 1/2 minutes. Note that the denial-of-service was accomplished during the entire time by means of a single packet, in contrast to other DoS attacks such as SYN floods, this attack thus leaves no abnormal adversarial traffic such as flooding which is detectable by the network.

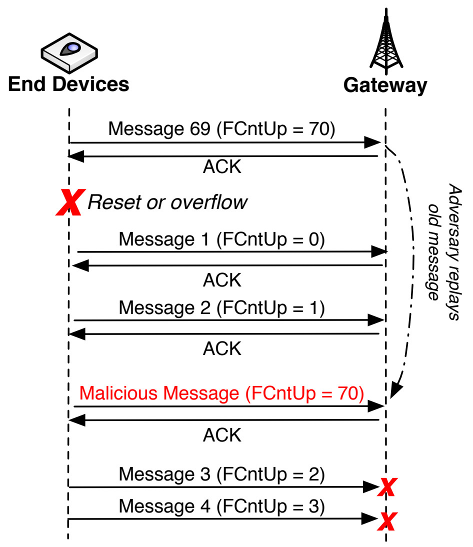


Fig. 4. An example of a replay attack for ABP.

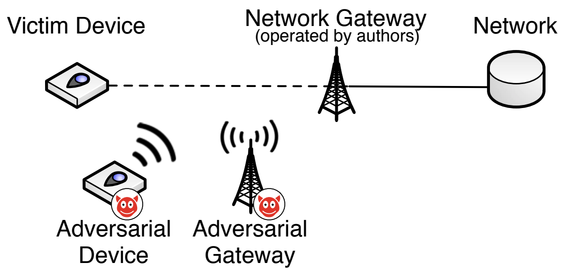
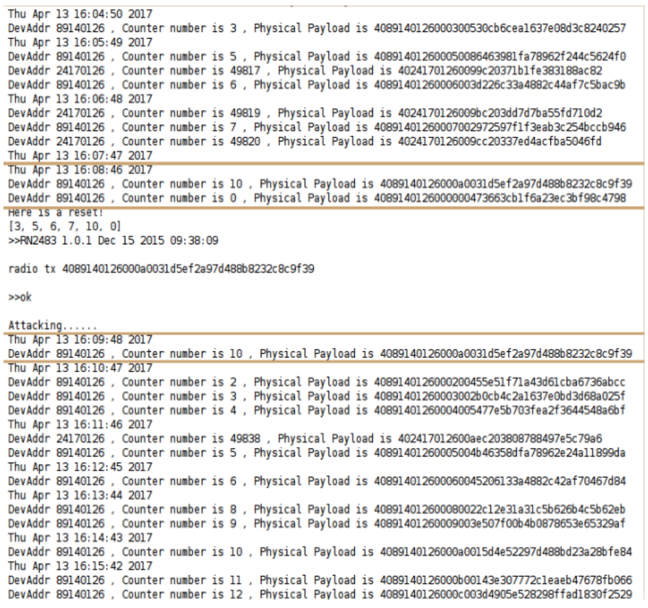


Fig. 5. Setup for LoRaWAN replay attack.



```
Thu Apr 13 16:04:50 2017
DevAddr 89140126 , Counter number is 3 , Physical Payload is 40891401260003005 30cb6cea1637e08d3c8240257
Thu Apr 13 16:05:49 2017
DevAddr 89140126 , Counter number is 5 , Physical Payload is 408914012600050086463981fa78962f244c5624f0
DevAddr 24170126 , Counter number is 49817 , Physical Payload is 40241701260099c20371b1fe383188ac82
DevAddr 89140126 , Counter number is 6 , Physical Payload is 408914012600060 03d226c33a4882c44af7c5bac9b
Thu Apr 13 16:06:48 2017
DevAddr 24170126 , Counter number is 49819 , Physical Payload is 4024170126009bc203dd7d7ba55fd710d2
DevAddr 89140126 , Counter number is 7 , Physical Payload is 408914012600070 02972597f1f3eab3c254bccb946
DevAddr 24170126 , Counter number is 49820 , Physical Payload is 4024170126009cc20337ed4acfba5046fd
Thu Apr 13 16:07:47 2017
Thu Apr 13 16:08:46 2017
DevAddr 89140126 , Counter number is 10 , Physical Payload is 408914012600 0a0031d5ef2a97d488b8232c8c9f39
DevAddr 89140126 , Counter number is 0 , Physical Payload is 40891401260000000473663cb1f6a23ec3bf98c4798
Here is a reset!
[3, 5, 6, 7, 10, 0]
>>RN2483 1.0.1 Dec 15 2015 09:38:09

radio tx 4089140126000a0031d5ef2a97d488b8232c8c9f39

>>ok

Attacking......
Thu Apr 13 16:09:48 2017
DevAddr 89140126 , Counter number is 10 , Physical Payload is 4089140126000a0031d5ef2a97d488b8232c8c9f39
Thu Apr 13 16:10:47 2017
DevAddr 89140126 , Counter number is 2 , Physical Payload is 408914012600020 0455e51f71a43d61cba6736abcc
DevAddr 89140126 , Counter number is 3 , Physical Payload is 408914012600030 02b0cb4c2a1637e0bd3d68a025f
DevAddr 89140126 , Counter number is 4 , Physical Payload is 408914012600040 05477e5b703fea2f3644548a6bf
Thu Apr 13 16:11:46 2017
DevAddr 24170126 , Counter number is 49838 , Physical Payload is 402417012600aec203808788497e5c79a6
DevAddr 89140126 , Counter number is 5 , Physical Payload is 408914012600050 04b46358dfa78962e24a11899da
Thu Apr 13 16:12:45 2017
DevAddr 89140126 , Counter number is 6 , Physical Payload is 408914012600060 045206133a4882c42af70467d84
Thu Apr 13 16:13:44 2017
DevAddr 89140126 , Counter number is 8 , Physical Payload is 408914012600080 022c12e31a31c5b626b4c5b62eb
DevAddr 89140126 , Counter number is 9 , Physical Payload is 408914012600090 03e507f00b4b0878653e65329af
Thu Apr 13 16:14:43 2017
DevAddr 89140126 , Counter number is 10 , Physical Payload is 4089140126000a0015d4e52297d488bd23a28bfe84
Thu Apr 13 16:15:42 2017
DevAddr 89140126 , Counter number is 11 , Physical Payload is 4089140126000b00143e307772c1eaeb47678fb066
DevAddr 89140126 , Counter number is 12 , Physical Payload is 4089140126000c003d4905e528298ffad1830f2529
```

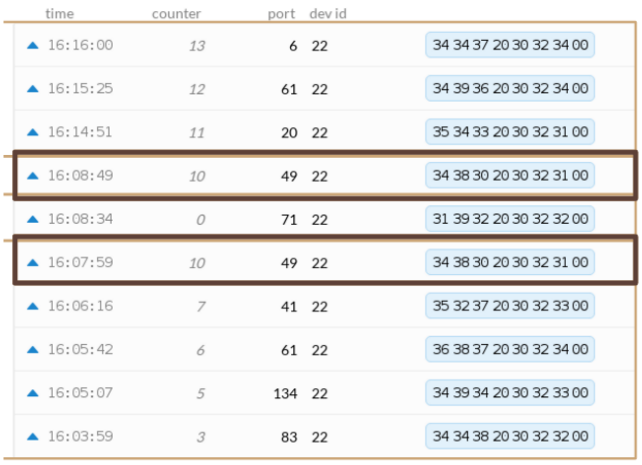
Fig. 6. Log file of malicious gateway.

## B. *Eavesdropping*

As discussed before, LoRaWAN implements channel confidentiality through AES in counter mode. Instead of setting the counter as a nonce, the packet counter value is used as input. As during a reset this counter value is reset according to the specification while the key remains in place, this means that the block cipher will recreate exactly the same key material. This is the classic textbook case of a key stream reuse.

In a stream cipher, a plaintext $P$ is combined through an exclusive OR with a key stream to obtain the ciphertext $C$. When given two messages $P_1$ and $P_2$ encrypted under the same keystream, $P_1 \oplus K = C_1, P_2 \oplus K = C_2$, an adversary could eliminate the secret key as

$$\begin{aligned} C_1 \oplus C_2 &= (P_1 \oplus K) \oplus (P_2 \oplus K) \\ &= P_1 \oplus P_2 \oplus \underbrace{(K \oplus K)}_{\text{cancels out}} \\ &= P_1 \oplus P_2 . \end{aligned}$$



| time | counter | port | dev id | |
|---|---|---|---|---|
| ▲ 16:16:00 | 13 | 6 | 22 | 34 34 37 20 30 32 34 00 |
| ▲ 16:15:25 | 12 | 61 | 22 | 34 39 36 20 30 32 34 00 |
| ▲ 16:14:51 | 11 | 20 | 22 | 35 34 33 20 30 32 31 00 |
| ▲ 16:08:49 | 10 | 49 | 22 | 34 38 30 20 30 32 31 00 |
| ▲ 16:08:34 | 0 | 71 | 22 | 31 39 32 20 30 32 32 00 |
| ▲ 16:07:59 | 10 | 49 | 22 | 34 38 30 20 30 32 31 00 |
| ▲ 16:06:16 | 7 | 41 | 22 | 35 32 37 20 30 32 33 00 |
| ▲ 16:05:42 | 6 | 61 | 22 | 36 38 37 20 30 32 34 00 |
| ▲ 16:05:07 | 5 | 134 | 22 | 34 39 34 20 30 32 33 00 |
| ▲ 16:03:59 | 3 | 83 | 22 | 34 34 38 20 30 32 32 00 |

Fig. 7. Log file of the victim's server.

Since the ciphertexts are transmitted over the air and known, in order to get the plaintexts, we first guess a part of the content in $P_1$, then derive the part of $P_2$ at the corresponding position. If all the plaintexts are readable, the guess is possibly correct. The more resets, the more likely it becomes to recover the messages. This method is also called *crib dragging* [14].

*Proof-of-Concept experiment*

Executing the attack requires only a LoRa node configured as a pass-through receiver or an off-the-shelf gateway to capture traffic from all sensors in range. The attack requires only the build-up of a dictionary per IoT device as identified by the DevAddr and FCnt value. As the payload size is highly limited and due to the air-duty-cycle limitation, developers are conditioned to minimize the amount of transmitted information, and thus data content in LoRaWAN can be expected to be highly condensed and structured. As IoT devices will be produced in high volume and devices can be assumed to be readily available to an adversary for purchase, the data format must thus be assumed to be publicly known.

That a key stream reuse will lead to a problem is obvious and does not require any formal demonstration. After the implementation of the proof-of-concept, we thus evaluated the effectiveness of a data recovery attack of sensor value information.

In our experiment, a sensor is configured to send data messages periodically, the default frame payload for data messages in these devices is 16 bytes, consisting of one light measure value and one temperature value. A malicious gateway keeps tracking the uplink messages from the sensor. The attack methods differ only slightly for when numbers or alphanumeric strings are transmitted, so we confine to only presenting the case for numbers. Suppose that from an inspection of the device the adversary knows that the length of the plaintext is 8 digits, which means that there are only 12 possibilities for one digit, numbers from 0 to 9, a space,

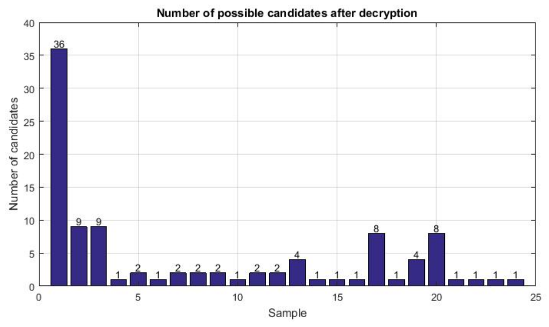


Fig. 8. The number of possible candidates.

or a placeholder. A message is divided into 2 parts: light and temperature, with one space between their values. The first part of the message gives a light value of at least 2 digits. The second part of the message gives a temperature value of 3 digits.

Choosing $P_1$ as the victim and $P_2$ as the reference object with both plaintexts unknown, we can traverse 12 options for each digit for $P_1$, and check the corresponding $P_2$ to see whether it is readable and consistent. Using the regular patterns as constraints to decrease the number of possible options for plaintexts, the victim's plaintext can be derived. Figure 8 shows a histogram of the number of possible candidates left after decryption as a function of the obtained sensor values. We see that the reconstruction is effective of structured values as soon as a handful of messages have been obtained. After 3 resets, the number of possible candidates is calculated. In the 24 valid samples, 45.8% have only one candidate, identical to the original data. For the others, the number of possible candidates is highly decreased from $12^8$ to 4, 8, 9, 36, etc. This attack is based on the assumption that the attacker is able to perform a reset and affect the light sensor value. If the attacker is not able to affect the sensor readings, the number of resets needed will be larger.

If the sensor data are not numerical values but for example bitmasks – imagine a bit value indicates whether movement was detected or light switched on in a particular room or corridor –, the traffic analysis would not need to recover the original values but only indicate differences between frames. This result may be obtained given two LoRa messages with different values.

### C. Bit-Flipping Attack

While LoRaWAN messages are both encrypted and equipped with a message integrity check, the two features are not applied at the same scope. Recall from the description of the protocol that the cryptographic message integrity code on the payload data and header information is checked and terminated by the infrastructure provider, while the payload encryption using the AppSKey is undone by the application provider. This means that in between the infrastructure operator's network server and the IoT solution provider's application server, the content cannot be checked for integrity and authenticity.

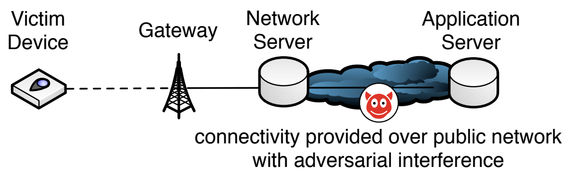

Victim Device
Gateway
Network Server
Application Server
connectivity provided over public network with adversarial interference

Fig. 9. The setup of a bit-flipping attack.

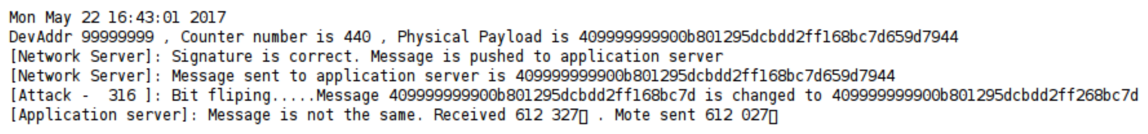
While good practices in network design would place the internal infrastructure of an infrastructure operator into a separate private network compartment, the link between the LPWAN operator and a third party application provider would typically run over a public network such as the Internet. Unless other precautions are taken such as a tunnel with validation or pinning of certificates, the messages between the two parties could be altered in content or rerouted.

While a block cipher normally reacts very sensitively to bit modifications, a principle called the avalanche effect which would normally render a bit-flipped message unreadable, as AES is only used as a key stream generator and a stream cipher is easily malleable unless an integrity check is combined with the decryption. LoRaWAN, however, deviates from the recommended practices of using authenticated encryption, and terminates the integrity check too early.

The vector is possible if an attacker can insert himself anywhere between the LPWAN operator and the IoT solution provider, see figure 9. There exist a variety of techniques for this, ranging from routing-based approaches (insufficiently secured routing protocols, BGP prefix hijacking, IP source routing option, etc.) to physical and link-layer based attacks (a compromised device on the path, a malicious actor in a shared data center, etc.), which are beyond the scope of this paper. As in a stream cipher bit modifications in the ciphertext affect the exact bit position in the plaintext in a predictable manner, it is thus possible for the adversary to arbitrarily modify the content of the sensor readings (and in case of key reuse, pretend that the sensor value had originated from a different device). In addition, header, routing information, and commands could also be modified in this way.

### *Proof-of-Concept experiment*

While from a cryptographic point of view the feasibility of a bit-flipping attack is evident, we also implemented a proof-of-concept for completeness sake, see figure 10. Indeed, the experiments show that given a man-in-the-middle attack between a network operator and an application server we registered, we were able to (1) modify the FrmPayload and change its room temperature data from "027" to "327", (2) increase the FCnt value until it was larger than the FCnt at the server, after which the message dropped by the application server, and (3) flip bits of DevAddr, so that the application considered the message to be from another end device.



```
Mon May 22 16:43:01 2017
DevAddr 99999999 , Counter number is 440 , Physical Payload is 409999999900b801295dcbdd2ff168bc7d659d7944
[Network Server]: Signature is correct. Message is pushed to application server
[Network Server]: Message sent to application server is 409999999900b801295dcbdd2ff168bc7d659d7944
[Attack -  316 ]: Bit fliping.....Message 409999999900b801295dcbdd2ff168bc7d is changed to 409999999900b801295dcbdd2ff268bc7d
[Application server]: Message is not the same. Received 612 327▯ . Mote sent 612 027▯
```

Fig. 10. An example result of a bit-flipping attack.

TABLE I
PHYSICAL PAYLOAD FORMAT OF AN ACK MESSAGE.

| MHDR | DevAddr | FCtrl | FCnt | MIC |
|---|---|---|---|---|
| 60 | 88889999 | 20 | 0B00 | BAE1557A |

### D. ACK spoofing

To maximize battery life, the data-acknowledgement mechanism in LoRaWAN is made optional to reduce the time the radio needs to be powered up.

Table I shows an acknowledgement message (FCtrl code 20) sent over the air, addressed to device 88889999. As can be observed, the ACK message does not state which message it is confirming. While there is a cryptographic integrity value that lets the IoT device confirm the authenticity of the ACK, the frame counter of the ACK is the sequential number of all downstream messages. A captured ACK could therefore be delayed and used to selectively acknowledge the successful receipt of another unrelated message, even when it has not arrived at backend provider.

As LoRaWAN relies on spread-spectrum techniques with a high spreading factor to realize a fault-tolerant link under minimal power requirements, the transmission of a LoRaWAN packet takes comparatively long. A versatile attacker without access to a gateway could thus prevent the receipt of the acknowledgement through selective jamming of the IoT device, and would be later able to knock out any uplink message while replaying the previously cached ACK.

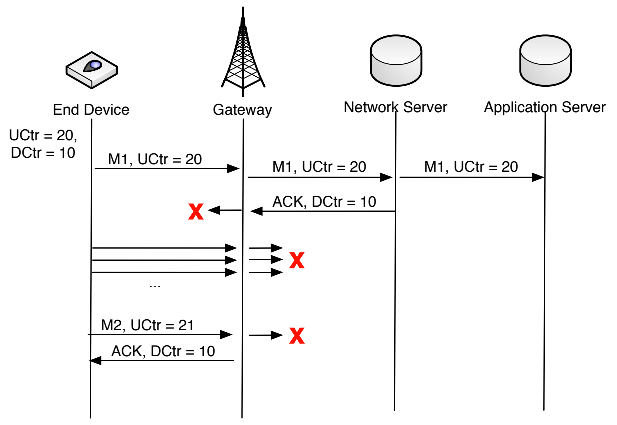


Fig. 11. ACK messages may be repurposed to acknowledge other frames than originally received by the application provider.

*Proof-of-Concept experiment*

For the demonstration of selective ACK spoofing, we assume that the gateway is malicious and may selectively suppress certain frames from transmission. Such a selective packet loss may be implemented by not sending the frame at all, or in a more subtle way by transmitting the packet to a wrong downlink port number so that the frame will be ignored by the actual recipient.

Figure 11 shows the procedure of the ACK spoofing experiment in effect. The end device sends a confirmed message M1 to gateway with $UCtr = 20$. The message is accepted by the server and the application server will confirm the successful message delivery. The network server would then create a cryptographic MIC with a confirmation of the delivery with a $DNCtr = 11$ and forward it to the gateway. Since downlink transmission is disabled at the gateway, no messages are sent to the end device. Since there is no ACK received by the end device, the device retransmits M1 for 7 times, and then considers M1 to be lost or rejected. The LoRa chip informs the IoT device's CPU of this situation through a "mac_err". A while later, the attacker would like to drop a status message from the IoT device but let the sender believe it was confirmed by the network. When message M2 is received, the attacker enables the gateway downlink transmission and discards the frame. As no frames were sent in between, the expected $DNCtr$ frame counter of the device is still 10. By sending out the previously signed and cached ACK, the end device will believe the frame to be received and processed by the network and the chip reports a "mac tx ok", while the frame has not been passed along.

### E. LoRa class B attacks

LoRaWAN defines three modes of operation to accommodate the diverse use cases of IoT deployment scenarios. In the most basic case, field devices would like to wirelessly upload status messages to an application server, and wait for an immediate reply by the application server in response to the measured data. If no download message is received within a short timeframe, the transmitter again powers down until the next time the device sends out a message. This so-called class A network thus realizes the most energy-efficient battery operation, but comes at the disadvantage that instructions to sensors can only be relayed when they report in by themselves based on their preconfigured duty cycle. In class B networks, the field devices in addition periodically wake up to wait for any incoming messages during extra receive windows. These durations are specified by the beacons broadcast by the gateway, but the receiver again powers down in between these windows. In a class C network, the LoRa transceiver is powered up all the time, which is ideal for applications requiring constant availability of a downlink channel at the expense of a significantly higher energy consumption.

The attacks described above were applicable to all LoRa modes. In the following, we will outline specific class B vulnerabilities, which allow a malicious actor to drain the battery of the field devices. As stated above, class B was created to balance power consumption and the possibility to periodically relay downlink instructions. In order to open receiving windows at fixed times, gateways should synchronously broadcast a beacon to provide a time reference to the end devices [15].

TABLE II
BCNPAYLOAD FORMAT [15].

| BCNPayload | NetID | Time | CRC | GwSpecific | CRC |
|---|---|---|---|---|---|
| Size (bytes) | 3 | 4 | 1 | 7 | 2 |

The beacon frame is basically a PHY layer header followed by the beacon payload. The BCNPayload for the EU 863-870MHz ISM band is shown in Table II. Beacons are not encrypted, nor otherwise protected against malicious modification. While the beacon payload includes a CRC to protect the integrity of the beacon's common part (Time and NetID), a cyclic redundancy check is only suitable to help in the detection of random bit flips as a result of a noisy channel and not a suitable mechanism to guard against an adversarial use case scenario. As a CRC is non-keyed and linear, it may be trivially updated by an attacker modifying an existing frame, as well as independently calculated in case the attacker would like to inject a forged frame. As the content of the BCNPayload is public knowledge and existing beacons list the configuration values in plaintext, an attacker can send out a beacon with malicious parameters, and that beacon would be received and processed by the end devices.

The fields in the LoRaWAN beacon frame allow for two fundamental attack vectors:

- *Finding or spoofing the location of a LoRa gateway:* The 7-byte value in the GwSpecific field in the BCNPayload is a composite of a 1-byte InfoDesc key followed by a 6-byte value. When InfoDesc = 0, 1, or 2, the 6-byte value contains the GPS coordinates of the antenna that is broadcasting the beacon (3 bytes for the latitude and 3 bytes for the longitude). Given the lack of encryption or an integrity check, the attacker can use this information to locate the gateway, but also maliciously alter the location of the gateway. As low-power devices will likely not run a power-hungry GPS receiver, any location-based service based on this information is thus vulnerable. Since GwSpecific is used for end devices to notify the network server when they are moving to another cell, faking the GwSpecific will cause the end devices to send wrong notices and cause the network server to have incorrect or inconsistent information of the end device location.
- *Battery exhaustion:* If the attacker is able to create his own beacons, it is possible that the attacker can send beacons with a random or extreme wakeup time values. In the first case, the adversary may disturb the downlink operation as the device would wake up at different times than expected by legitimate gateways. In the second case, the device can be triggered to wake up frequently, thereby increasing the power consumption of the sensor. The ability to drain the batteries will let an IoT deployment fail prematurely, resulting in an early expensive write-off or a labor-intensive replenishment in the field.

## V. Attack mitigation and security suggestions

In this section, we will discuss possible mitigation and security suggestions with respect to the previously described attacks. Some of the counter measures will require minor modifications on the firmware or the way LoRaWAN transceivers are integrated into an IoT device, while others require modifications to the standard to mitigate the attack vector at the root of the problem.

The work described in this article was performed when LoRaWAN specification 1.0.2 was the most recent version. Between the completion of our research and the camera-ready version, a new version of the LoRaWAN protocol, specification 1.1, was released which included several changes that address some of the vulnerabilities we have discussed above. As both specifications are supported, all of the results presented above remain valid. In the following, we will hence describe, in addition to our proposed controls, also the design changes introduced with the 1.1 version.

### A. Replay attack for ABP-activated nodes

The replay attack is based on the observation that the NwkSKey and AppSKey are actually used as long-term key material that remains unchanged after a counter reset, instead of being restricted to a single session. In order to prevent this attack from happening, the following measures could be taken:

- The use of activation by personalization should be minimized and if used, new keys should be downloaded periodically. Device enrollment by OTAA does however not mean the end device is secure, because counters can overflow. Yet, the renegotiation on power-up means that in over-the-air activation an attacker would wait for a longer time to perform a replay attack.
- End devices should be physically protected to prevent a malicious party to initiate a system reset. While this is difficult to accomplish in a variety of IoT deployment contexts, design changes such as non-volatile memory may preserve the counter value in between resets. If the attacker cannot reset the counter by resetting the end devices, the only way to achieve the attack is to wait for a counter overflow. This change significantly reduces the exposure, but requires a change in the LoRaWAN specification.

  Fortunately, this modification is included as such in version 1.1. Under the new specification, ABP devices must use non-volatile memory to store the frame counters, and re-initialization of an ABP end-device's frame counters is explicitly forbidden. An ABP device thus uses the same session keys throughout their lifetime, and the specification recommends that no rekeying is possible. If this mechanism of non-volatile storage is correctly implemented and not subject to for example glitching attacks, the attack surface is significantly reduced as the adversary must wait for a counter overflow. While this will still occur, especially in busy networks, it will take longer to be exploited by an adversary.

- To prevent the replay attack, the end device should be required to rekey every time the counter reaches its maximum value. If the end device is using OTAA, it should go through the OTAA activation procedure again to obtain new session keys. If the end device is using ABP, it should be re-configured, and session keys should be changed.

### B. *Eavesdropping*

The eavesdropping attack exploits that a block cipher in counter mode is not secure if the counter value is allowed to repeat. Specifically the case of a monotonically increasing counter value is the classic example of how CTR mode can fail in practice given volatile memory. Also here the problem may be addressed through a variety of measures:

- Replace the counter value by a nonce (a number used just once), which is derived from a cryptographically-secure pseudo random number generator or a true random number generator available on the sensor platform. While this would immediately reduce the chance for a collision to the unavoidable theoretical minimum of the birthday paradox collision probability of one in $2^{\frac{\text{block size}}{2}}$, extra care must be taken to prevent the IoT device from starting up with the same seed value across boot ups and installations. Harvesting unique entropy on embedded systems is challenging, but resilient methods have been developed to accomplish this.
- Rekey on reset in addition to any counter overflow. Since this attack needs to collect several messages with the same counter value and session keys, changing session keys periodically can prevent the attacker from collecting enough messages. If every time the counter value in the end device reaches its maximum value, the end device re-activates, there will not be enough messages for the attacker to perform decryption. As discussed with the replay attack, this is easier to achieve with OTAA than with ABP.

The decryption success rate is related to the number of messages that use the same key stream and same counter value. As explained for the replay attack, resetting a device under v1.1 will not help the attacker, so the decryption success rate will decrease, as it now solely relies on counter overflows.

### C. *Bit-flipping attack*

A malicious bit flipping of the sensor values in between the infrastructure operator and application provider is possible due to the too-early termination of the message integrity code in the system architecture. This early termination is also present in v1.1. While it is good secure practice to design networks based on the defense-in-depth principle and for example expect an SSL tunnel in between network and application server, the LPWAN architecture and protocol should be resilient to negligent deployment scenarios where such precautions are omitted.

- The obvious solution to avoid an attack featuring a malicious modification of the payload content is to run the integrity check value at the application server and not the network server. Ideally, a modern protocol design should implement *authenticated encryption* instead of mere encryption, a lesson that has been repeatedly reinforced in the long chain of attacks on protocols such as Wi-Fi or TLS. This would however cannibalize some payload bytes in the MAC payload to accommodate a MIC checked by the application server. Equipping the network server with the AppSKey is clearly undesired with respect to business and deployment model, and contra-productive with respect to a stringent security design.
- A more radical but better approach would be to repurpose protocol fields, which would require some changes to the standard and firmware updates. Note that in the LoRaWAN specification there are actually two checks included. The MIC that runs a cryptographically strong integrity check value over the MAC-layer payload, and a CRC appended to the physical-layer payload, which checks for bit flips at the physical-layer protocol data unit. The additional coverage provided by the CRC is only the inclusion of the data bits in front of the MAC header, such as the radio preamble, while at the same time providing a much weaker security service to the MAC-layer payload. In other words, while any *payload* modification detectable by the CRC can also be detected by the MIC, there is an entire class of attacks that can only be detected by the MIC, but never by a CRC. While cyclic redundancy checks are trivial to compute, the computational overhead of a MIC can also be designed minimal, for example if relying on constructions such as a Carter-Wegman message authentication code, and RFC4418 defines with the UMAC algorithm an open-standard high-speed MIC based on this principle.
  While there is value in ensuring the integrity of the frame when received by the network, a much better solution would be to replace the CRC by a MIC as shown in figure 12. Although CRCs are trivial to compute and were usually favored for this reason over cryptographically-secure checksums, specialized hardware acceleration for AES has melted this argument away over the last decade. Such a design would guard against *any* kind of modification to the frame, and the freed-up bytes at the MAC layer could be repurposed for a MIC for the application provider, thus realizing authenticated encryption without a single byte of payload capacity being lost. If such a solution

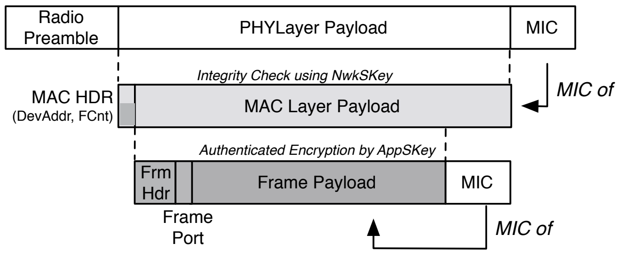


Fig. 12. A repurposing of the fields will lead to better coverage against stronger threat models, while not changing the amount of available payload bandwidth in any way.

is realized, the channel between infrastructure operator and application provider is protected against subversions of confidentiality, integrity and authenticity regardless of additional security measures taken in the network (which should regardlessly be there).

## *D. ACK spoofing*

The fundamental problem that enabled the ACK spoofing and malicious repurposing of previously stored acknowledgement frames is that the ACK message does not indicate which message it actually confirms. This is also still the case in v1.1. While the intentional frame loss by a compromised gateway cannot be prevented as it is outside of the scope of both the IoT device owner and the application server, there exist several measures that could be taken to prevent the malicious replay of an ACK in another context and detect excessive frame losses:

- With the change towards a MIC for both the connection to the network server and the application server, it is possible to add a cryptographic checksum with the returned acknowledgement. The return value should include the entire packet as sent by the field device. Costing no additional bytes, it will enable the IoT device to confirm that (a) the ACK belongs to this message, and (b) the value of the message was not changed in transit. As the ACK is bound to one specific message, it is not possible to cache and resend ACKs in another context.
- As an additional precaution, we further suggest that the authenticated encryption (AE) of the MAC-layer payload should be modified towards an AE with associated data (AEAD). This extends the cryptographic integrity check to other fields, specifically the device address, frame counter, as well as select frame header fields. If the network will act maliciously and discard frames, evidence will remain in the gaps of the frame counter value, which the adversary cannot modify due to the AEAD coverage, alerting the application server of a potential problem. Furthermore, attacks in the network (for example between the gateway and network server, or network server and application server) of rerouting the frame to a wrong frame port and thus prevent correct reception will be futile.

## *E. LoRa class B attacks*

Root cause for the mode-specific attacks, such as the location spoofing or battery exhaustion, was the insufficient protection of the beacon frames, specifically the lack of an integrity check value (which is still the case in v1.1). As the LoRaWAN beacon is a MAC payload claimed by the network operator for management purposes, the proposed modification of transitioning the PHY CRC to a MIC, as shown in figure 12, would immediately solve the problem of any malicious beacon modifications, including all of the specific class B attacks outlined in the last section.

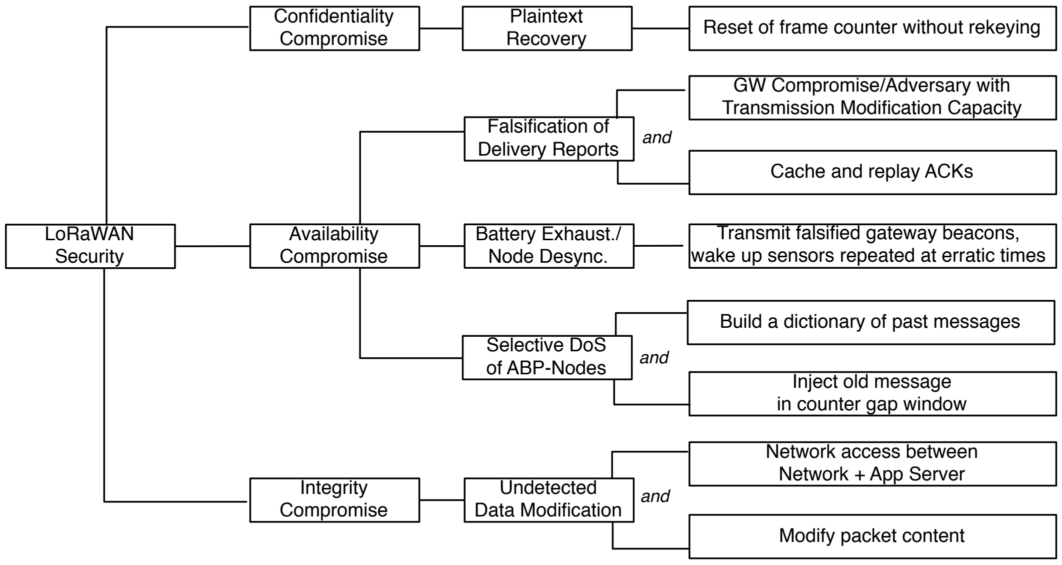
It has to be remarked though that a MIC is actually a trade-off in this model. We assume here that the NwkSKey will not be extracted by a capable adversary with physical access from the IoT devices, as otherwise the attacker could create and broadcast falsified beacons using this key himself. If the NwkSKey is shared across a large number of devices, the exposure to a beacon attack increases, while per-device NwkSKeys would reduce efficiency as beacons could not be validated by more than one device. While over the last decade major advances have been made in secure key storage on embedded devices, this is a rather strong assumption to make, especially if the IoT platform is widely enough deployed to be of specific interest to the attacker. This dilemma of bootstrapping the secure channel is also faced by protocols such as 802.11 and its latest additions such as 802.11w.

While this threat model would only compromise the security of network management frames (the actual IoT application data is still protected with the proposed AEAD scheme), it can be argued that this extra effort is not worth the risk. Given this threat model, a better solution here would be the introduction of a cryptographic signature in place of, or in addition to, a PHY MIC which comes at the extra cost of putting up a public key infrastructure. This would allow a single beacon to be cryptographically verified by all end devices, regardless of user and application provider.

## *F. New Key Hierarchies*

One of the most profound changes that was introduced in the specification 1.1 is the key derivation and application scope of key material. As described above, v.1.0.2 basically relied on two types of keys, the NwkSKey to perform an integrity check up to the network server and AppSKey for encrypting the payload until the application server. In OTAA of the original specification, both these keys were derived from a single root key.

In OTAA of LoRaWAN 1.1, the logical separation of network and application operator is also considered in the key derivation. Now there are two root keys, the NwkKey and an AppKey which are implanted into the device before delivery. While it is the sole purpose of the AppKey now to derive the encryption key used between the device and the application server, the NwkKey is now used to derive 3 session keys, which provide the necessary key material for the purpose of network communication and management on the wireless interface. Two of these session keys, FNwkSIntKey



LoRaWAN Security
Confidentiality Compromise
Plaintext Recovery
Reset of frame counter without rekeying
Availability Compromise
Falsification of Delivery Reports
and
GW Compromise/Adversary with Transmission Modification Capacity
Cache and replay ACKs
Battery Exhaust./ Node Desync.
Transmit falsified gateway beacons, wake up sensors repeated at erratic times
Selective DoS of ABP-Nodes
and
Build a dictionary of past messages
Inject old message in counter gap window
Integrity Compromise
Undetected Data Modification
and
Network access between Network + App Server
Modify packet content

Fig. 13. Attack Tree: requirements and results of the vulnerabilities presented in this paper.

and SNwkSIntKey are used for MIC computation. SNwkSIntKey computes a MIC on transmissions from the network to the device. For uplink messages, both FNwkSIntKey and SNwkSIntKey are used in combination, which allows a scope limitation on what may be verified by a network operator where the device is currently roaming. MAC commands are now also protected by a new session key NwkSEncKey.

The split in scope is also introduced for counters. Instead of maintaining only two counter values for the state of the uplink and download channel, specification 1.1 also here separates out application traffic from management traffic. Uplink frames are still counted by FCntUp, in case of download messages a different counter is applied depending on the origin of the transmission. Messages from the application server are counted by the value AFCntDown, while MAC frames are kept in the value NFCntDown. The current state of confirmed messages is now also kept, and reported during uplink frames by the sensor in the ConfFCnt counter.

## VI. Conclusion

LoRa is a proprietary spread-spectrum modulation scheme for the Internet-of-Things (IoT) that has been gaining popularity fast in recent years. It promises long-range communication at low energy consumption, thereby providing an ideal use case of scattered deployments operating for a long time on battery power. The LoRaWAN protocol is a MAC-layer protocol for LoRa, which provides the communication infrastructure and interfaces for gateway-sensor topologies, node coordination, and medium access. As LoRaWAN is a fairly novel protocol, its level of security has not yet been rigorously studied in the academic literature.

In this paper, we have presented the standardized security features of LoRaWAN v1.0.2 and provided an analysis on the effectiveness of the security mechanisms in place in an adversarial scenario. We have found five noteworthy weaknesses that can compromise the confidentiality, integrity and availability of a LoRaWAN deployment; these attacks are depicted in the attack tree shown in figure 13:

- First, the use of the message counter value in the AES-CTR construction can lead to a key stream reuse. According to the specification, this counter is monotonically increasing and will thus overflow, and must reset after power up. Combined with the lack of rekeying in these events, this will allow an adversary predictions about the message plaintext. To realize such an attack, the attacker should have basic knowledge with respect to message type and format and should be able to collect messages with the same key stream. Moreover, if the attacker is able to reset ABP-activated end devices, this could increase the probability for messages to be decrypted correctly.
- Second, as the acknowledgement mechanism is insufficiently bound to the uplink messages, an adversary may withhold ACKs and use them to selectively acknowledge messages that actually had not been delivered. This attack requires a gateway compromise or an adversary with the capability to selectively collide messages on the air. To execute this type of attack, the attacker should be able to control a gateway.
- Third, in class B networks, the beacon mechanism with which nodes are synchronized and woken up for downstream messages is not cryptographically protected. This allows an adversary to let sensors believe they are mobile, and thus exhaust their battery.

- Fourth, by caching old messages and replaying them within the accepted counter gap window, a selective denial-of-service attack may be launched against specific nodes. This DoS vector is actually the backlash from the windowing mechanism preventing out-of-order frames in combination with a lacking MIC on the MAC header. For this attack, an attacker should be able to capture and resend a message whose session keys and DevAddr are the same as those of the victim's end device.
- Fifth, as the integrity check in the current LoRaWAN protocol is terminated too early at the network server, transmissions between the infrastructure operator and IoT application provider may be selectively altered if no other security measures are protecting that link.

In this paper, we have proposed a number of countermeasures and changes to the LoRaWAN protocol which will render all of these attack vectors harmless, many of which can be implemented with minimal changes to the LoRa ecosystem.

## References

[1] Gartner, http://www.gartner.com/newsroom/id/3165317, 2015.

[2] M. Abomhara and G. M. Køien, "Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks," *Journal of Cyber Security*, vol. 4, pp. 65–88, 2015.

[3] G. Margelis, R. Piechocki, D. Kaleshi, and P. Thomas, "Low throughput networks for the IoT: Lessons learned from industrial implementations," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 181–186.

[4] B. Singh and B. Kaur, "Comparative study of internet of things infrastructures & security," in *Global Wireless Submit*, 2016.

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

[6] R. Miller, "LoRa security - building a secure LoRa solution," https://labs.mwrinfosecurity.com, 2016.

[7] S. Zulian, "Security threat analysis and countermeasures for LoRaWAN join procedure," http://tesi.cab.unipd.it/53210/, 2016.

[8] 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)*. IEEE, 2017, pp. 1–6.

[9] 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.

[10] J. Michorius, "Whats mine is not yours: LoRa network and privacy of data on publishing devices," in *25th Twente Student Conference on IT*, 2016.

[11] S. Naoui, M. E. Elhdhili, and L. A. Saidane, "Enhancing the security of the IoT LoRaWAN architecture," in *Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), International Conference on*. IEEE, 2016, pp. 1–7.

[12] E. Aras, N. Small, G. S. Ramachandran, S. Delbruel, W. Joosen, and D. Hughes, "Selective jamming of LoRaWAN using commodity hardware," *arXiv preprint arXiv:1712.02141*, 2017.

[13] J. Lee, D. Hwang, J. Park, and K.-H. Kim, "Risk analysis and countermeasure for bit-flipping attack in LoRaWAN," in *Information Networking (ICOIN), 2017 International Conference on*. IEEE, 2017, pp. 549–551.

[14] T. Dazell, "Many time pad attack – crib drag," http://travisdazell.blogspot.nl/2012/11/many-time-pad-attack-crib-drag.html, 2012.

[15] LoRa Alliance, https://www.lora-alliance.org/.