(In)security of the Radio Interface in Sigfox*

Loïc Ferreira

Orange Labs, Applied Cryptography Group, Caen, France
loic.ferreira@orange.com

Abstract. Sigfox is a popular communication and security protocol which allows setting up low-power wide-area networks for the Internet of Things. Currently, Sigfox networks operate in 72 countries, and cover 1.3 billion people. In this paper, we make an extensive analysis of the security mechanisms used to protect the radio interface. We describe news attacks against data authenticity, which is the only mandatory security property in Sigfox. Namely we describe how to replay frames, and how to compute forgeries. In addition, we highlight a flaw in the (optional) data encryption procedure. Our attacks do not exploit implementation or hardware bugs, nor do they imply a physical access to any equipment (e.g., legitimate end-device). They rely only on the peculiarities of the Sigfox security protocol. Our analysis is supported by practical experiments made in interaction with the Sigfox back-end network. These experiments validate our findings. Finally, we present efficient counter-measures which are likely straightforward to implement.

Keywords: Sigfox · Security protocol · Internet of Things · Low-power Wide-area Network · Cryptanalysis.

1 Introduction

1.1 Overview

Sigfox is a communication system used in the Internet of Things (IoT). It allows establishing a low-power wide-area (LPWA) network between a set of remote end-devices and a central back-end network (see Figure 1). The back-end network, owned by the Sigfox company, is an intermediary between a service provider and its fleet of end-devices (e.g., sensors, actuators). The messages sent by an end-device is received on the Sigfox's back-end network where they are made available to the service provider. Conversely, the service provider can send messages to its end-device through the back-end network managed by Sigfox. Different kind of subscriptions are proposed by Sigfox, from the "One" subscription (which allows 1-2 daily uplink messages, 0 downlink message), up to the "Platinum" subscription (101-140 daily uplink messages, 4 downlink messages).

The Sigfox system enables different services such as asset tracking, geolocation, sensitive site monitoring, smart home, smart metering, healthcare. Sigfox uses free but regulated frequency bands (e.g., 868-869 MHz in Europe, 902-905 MHz in North America, 922-923 MHz in Japan and South Korea). Supplied with an autonomous battery, a Sigfox end-device is supposed to communicate through several kilometers. Its lifespan is expected to be up to five or ten years. With respect to the radio specificities, Sigfox can be compared, up to some point, to LoRaWAN [8, 23] and NB-IoT.

Currently, Sigfox operates in more than 72 countries on all continents [16]. The

*This is the full version of the paper "Sigforgery: Breaking and Fixing Data Authenticity in Sigfox" accepted at Financial Cryptography 2021.

multiple networks cover 1.3 billion people, and represent 56 million daily messages from 17 million IoT devices. Figure 2 depicts the coverage of Sigfox networks in several geographic areas [15].

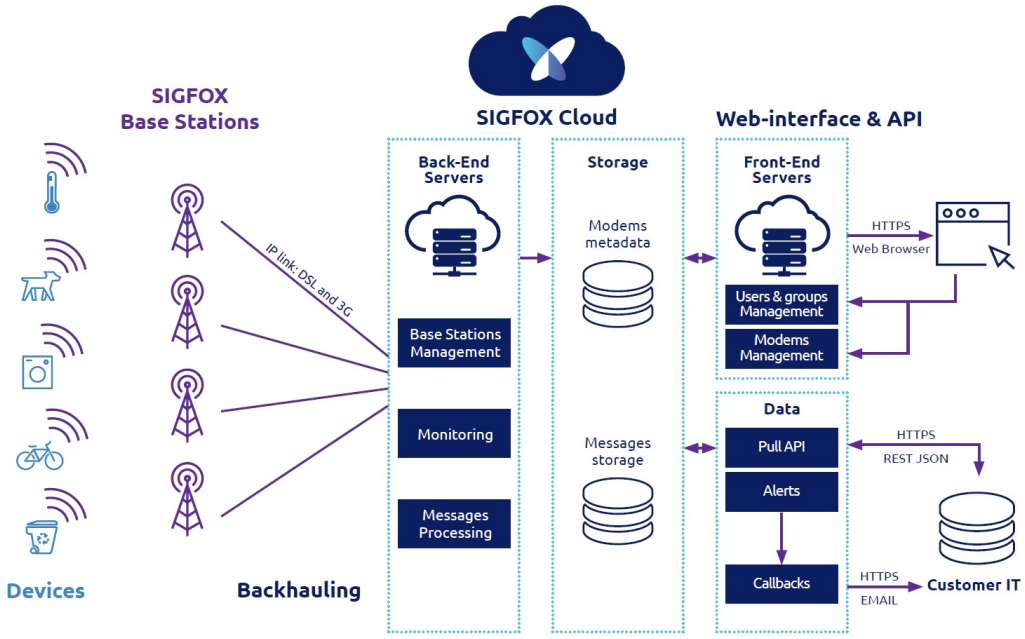


Figure 1: Sigfox architecture (source: [19])

1.2 Contributions

In this paper we present different flaws and attacks that are practicable against Sigfox, and describe counter-measures that are straightforward to implement. More precisely, our contributions are the following.

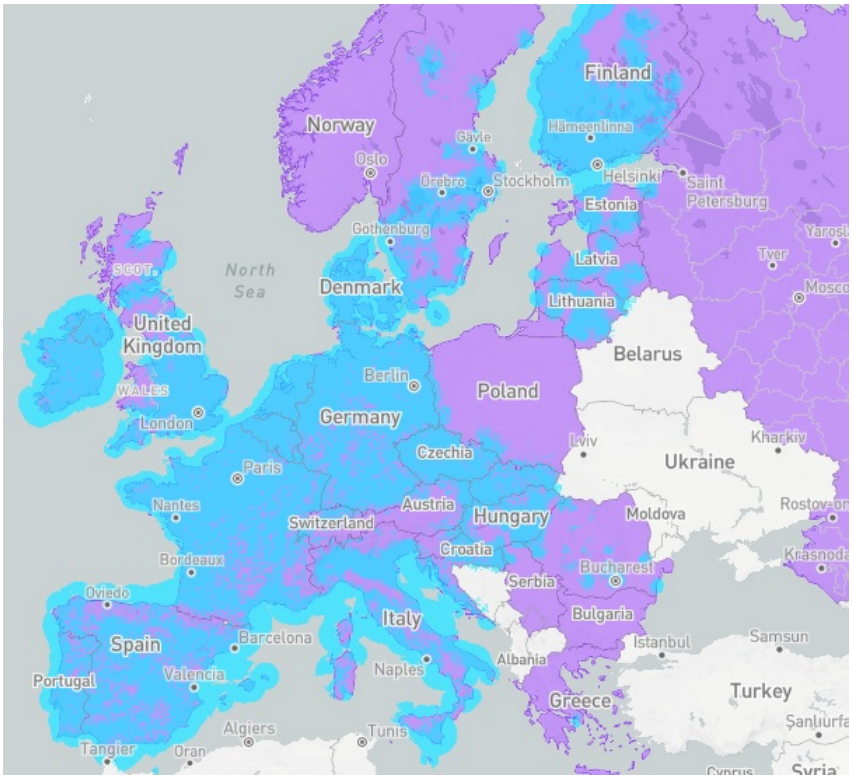
Unified Description of the Security Mechanisms. For several years, the precise security mechanisms implemented in Sigfox remained undisclosed. Several past official technical documents have presented some security features, but with very few details. It is only recently that a (incomplete) official specification was published. With the help of unofficial technical documents, we provide in this paper a unified and detailed description of the security mechanisms used in Sigfox, confirmed by our own experiments.

Recap and Extension of Previous Attacks. We recall and describe with details the attacks that have been formerly proposed by other authors (and which we deem valid): Lifchitz [7], Euchner [5], and Coman, Malarski, Petersen, and Ruepp [1]. We extend several of these attacks with scenarios that have not been considered so far.

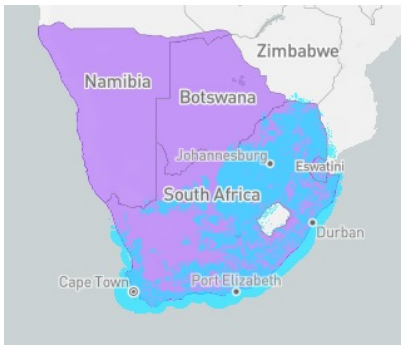
These attacks break data authenticity, and hinder availability of the system. We also recall another type of attack (key extraction), which is not directly related to the way the security mechanisms are defined and used in Sigfox. Yet, this attack still poses a threat which, according to us, must be considered.

New Attacks against Sigfox. We present a flaw in the Sigfox encryption procedure, and two news attack against Sigfox. These two attacks break data authenticity.

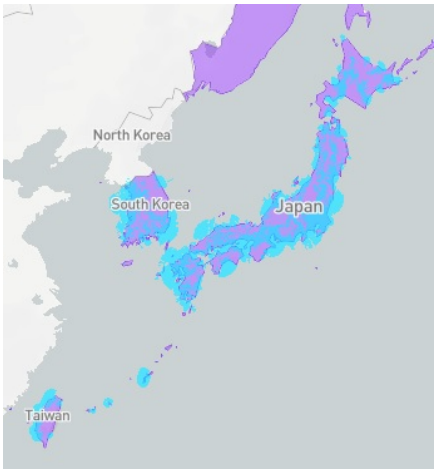
The flaw in the encryption procedure allows to passively get access to the plaintext



(a) Europe



(b) South Africa



(c) Taiwan, South Korea and Japan

Figure 2: Sigfox coverage in several geographic areas (source: [15]). Actual deployments appear in blue, ongoing deployments appear in purple.

data in a specific case when encryption is activated.

Regarding the attacks, we first explain how to replay previous downlink encrypted frames to an end-device (even once encryption is deactivated). Then, we describe how to forge a valid uplink frame (i.e., with a correct MAC tag) from a genuine uplink frame. We describe how to forge such frames with encrypted and clear frames. The complexity of this attack is $O(1)$ (in contrast to previous attacks against Sigfox), and it allows deceiving the back-end network.

The attacks that we propose do not exploit potential implementation or hardware bugs. They do not imply a physical access to any equipment (in particular a legitimate end-device). They are independent of the means used to protect the secret parameters (e.g., a secure element in the end-device). These attacks depend exclusively on the peculiarities of the Sigfox MAC and encryption functions. The adversary needs only to act on the air interface (i.e., to eavesdrop on legitimate frames, and to send the forged frames to the back-end network).

Table 1 summarises the different attacks presented in this paper.

Table 1: Attacks against Sigfox. The context indicates if the frames are encrypted ("ENC") or not ("MAC only"). The direction indicates if the uplink frames ("UL") or the downlink frames ("DL") are targeted. The symbol "*" indicates that the corresponding attack is doable under conditions, or probabilistic.

Attack	Description	Security property	Context	Direction
Frame replay	[7, 5] Section 3.1.3	Data authenticity	MAC only, ENC*	UL, DL
Denial of service	[5] Section 3.2.3 Section 3.2.4	Availability	MAC only, ENC*	UL, DL
Forgery: exhaustive search	[5] Section 3.3.3	Data authenticity	MAC only, ENC	UL, DL*
Lack of encryption	Section 4.1	Data confidentiality	ENC	UL
Replay of downlink encrypted frames	Section 4.2	Data authenticity	ENC	DL
Forgery: completion attack	Section 4.3	Data authenticity	MAC only, ENC	UL
Key extraction	[5]	Entity authentication, data authenticity, data confidentiality	MAC only, ENC	UL, DL

Practical Experiments. We have validated the MAC tag forgeries that we describe in two ways. First "offline", with the librenard library developed by Euchner [3]. This library implements the same cryptographic functions as a legitimate Sigfox end-device (except the encryption function). Euchner has validated librenard with practical experiments done in interaction with the Sigfox back-end network. In addition, librenard provides a function which takes as input an uplink frame, and verifies it (including the MAC tag). We have completed the librenard library in order to support encryption, and used it to successfully validate the MAC tag forgeries that we describe. Secondly, we have made real-life experiments, and we have observed that the forged frames were accepted on the Sigfox back-end network.

Efficient Counter-measures. We present efficient counter-measures which are likely straightforward to implement for most of them. They allow thwarting all the aforementioned attacks.

1.3 Responsible Disclosure

We reported the results of our analysis (attacks and counter-measures) to Sigfox (August 19, 2020). Sigfox acknowledged receipt of our paper. At the time of writing this extended version, and despite new messages sent to Sigfox, we have received no further news.

1.4 Outline

Section 2 presents the security mechanisms used in Sigfox. The previously known attacks against Sigfox are presented in Section 3. The new attacks are described in Section 4. The counter-measures are presented in Section 5. Finally, we conclude in Section 6.

1.5 Notation

`0x`	This prefix indicates an hexadecimal format (e.g., `0x4D`).
`0b`	This prefix indicates a binary format (e.g., `0b10`).
$x \| y$	Concatenation of values x and y.
$\|x\|$	Size of x.
$\mathsf{msb}(x, n)$	n most significant bits of x.
$\mathsf{lsb}(x, n)$	n least significant bits of x.
$\max(a, b)$	Maximum value between a and b.
$\min(a, b)$	Minimum value between a and b.
$x[i]$	Byte i of x.
$x[i \cdots j]$	Bytes i to j, $i \leq j$, of x.
$\mathsf{selfpad}(B)$	The function selfpad takes as input a byte string B, and pads B with itself until the length of the resulting string is a multiple of 16 bytes. If B's length is a multiple of 16 bytes, then $\mathsf{selfpad}(B) = B$.

2 Description of the Sigfox Security Protocol

This section presents the cryptographic mechanisms used in the Sigfox system in order to protect the radio interface (i.e., between an end-device and the back-end network). More specifically, it focuses on the cryptographic and security mechanisms used to compute uplink (from the end-device to the back-end network) and downlink messages.

A partial description of the security and cryptographic mechanisms used in Sigfox can be found in the official specification [21]. This specification can be completed by explanations provided by Euchner in his "open Sigfox specification" [5]. The latter is the result of reverse engineering, validated by practical experiments made with a legitimate Sigfox end-device. To the best of our knowledge, there exists no official specification describing the Sigfox encryption function which is publicly available. Pinault has presented this encryption function [14]. We have corrected Pinault's description through the reverse engineering of the X-CUBE-SFOX package [24] done with the Ghidra tool [12]. Our findings have been validated with practical experiments made in interaction with the back-end network.

We provide the technical details that are necessary and sufficient for the remainder of the paper. We skip the description of the operations done on the PHY layer (error correction, "whitening", etc.) prior to transmitting a frame, as well as the optional use of replicas in the case of an uplink frame. We refer the interested reader to the Sigfox specification [21].

The security mechanisms used on the radio interface are based on the AES block cipher [11], and a static symmetric key called "Network Access Key" (NAK). The NAK key is shared between the end-device and the back-end network. No session key derivation is made, and the same static NAK key is used by the end-device for its whole lifetime.

2.1 Frame Format

2.1.1 Uplink Frame.

The format of an uplink frame (i.e., sent by the end-device to the back-end network) is the following (length in bit)

$$\mathsf{ft}\ (13)\|\mathsf{hdr}\ (48)\|\mathsf{payload}\ (0\text{-}96)\|\mathsf{mac}\ (16\text{-}40)\|\mathsf{crc}\ (16)$$

where hdr corresponds to

$$\mathsf{li}\ (2)\|\mathsf{bf}\ (1)\|\mathsf{rep}\ (1)\|\mathsf{cnt}\ (12)\|\mathsf{devid}\ (32)$$

The frame type ft depends mainly on the nature of the frame (application, control), and its length. The payload field carries the (optionally encrypted) data, which size ranges from 0 to 12 bytes, or is 1-bit long (in such a case the data is carried in the header hdr, and payload is empty). The field mac carries the frame's MAC tag (which length ranges from 2 to 5 bytes – see Section 2.4), and crc carries the CRC tag.

In the hdr field, the parameter li is used to indicate the size of the MAC tag, or to carry the 1-bit data. The parameter bf indicates if a downlink frame is expected in response to the uplink frame. rep is always set to 0. The 12-bit parameter cnt is a frame counter, incremented each time a new uplink frame is sent (see Section 2.2). The parameter devid corresponds to the end-device's identifier (encoded in little endian format).

2.1.2 Downlink Frame.

The format of a downlink frame is the following (length in bit)

$$\mathsf{ft}\ (13)\|\mathsf{ecc}\ (32)\|\mathsf{payload}\ (64)\|\mathsf{mac}\ (16)\|\mathsf{crc}\ (8)$$

The frame type ft is constant. The parameter ecc is an error correction code computed over payload‖mac‖crc. The (optionally encrypted) data is carried in payload, which is 8-byte long. The fields mac and crc carry respectively the frame's MAC tag and CRC tag.

2.2 Frame Counter

The purpose of the frame counter cnt is to detect frame replays. Although, the length of the cnt field is 12 bits, the maximum value of this parameter is $2^i - 1$, $i \in \{7, \ldots, 12\}$. Presumably, i depends on the Sigfox subscription (i.e., the maximum of allowed daily uplink frames). When the maximum value of cnt is reached, this parameter must be set to 0.

A new uplink message is accepted by the back-end within an acceptance interval (see Figure 3). Let $\mathsf{cnt} = n$ be the counter value of the last (valid) received uplink message, and $\mathsf{cnt} = n'$ be counter value of the newly received message. Assuming that all other parameters are correct, especially the MAC tag mac, the following rules are applied by the back-end [17]:

- If $n' \leq n$ then the message is silently discarded.
- If $n < n' \leq n + mi$ then the message is accepted, and a warning is raised.
- If $n' > n + ma$ then the message is not delivered, and an error is raised.

The values mi and ma depend on the number of elapsed days d between the two frames n and n' (sliding day; reception the same day accounts for 1), and the maximum amount of daily uplink messages c (defined per contract):

Figure 3: Acceptance interval

- $mi = \min(300 \times d, c \times (d + 2))$,
- $ma = \max(300 \times d, c \times (d + 2))$.

The warning and the error events mean that a corresponding information is notified on the back-end side (but no specific message is sent to the end-device).

If a desynchronisation occurs with respect to the message counter between the end-device and the back-end network (error case), the delivery of the incoming messages is interrupted. This issue can be fixed through a "disengage" procedure. It implies for the end-device's owner to log into the back-end network. Eventually the delivery of uplink messages by the back-end network to the end-device's owner is resumed [18]. Yet the uplink messages sent in the meantime are discarded by the back-end network, hence lost [17].[1]

2.3 Encryption Function

The format of encrypted and clear frames is the same. Encryption is (de)activated on the back-end side, upon request of the end-device's owner. Encryption cannot be done on a per frame basis. That is, either all the frames are encrypted or none. Then, the back-end acts accordingly. This implies in particular that, if encryption is activated, the downlink frames are also encrypted.

The encryption of a frame is made with AES-CTR. From the NAK key K, and two 16-byte values V_0, V_1, an encryption key K_e and a value W for the counter mode are computed.

The value V_b, $b \in \{0, 1\}$, is the concatenation of the bit b, followed by the 4-byte end-device's identifier devid, and 95 zero bits:

$$\begin{aligned} V_0 &= 0 \| \mathsf{devid} \| 0 \cdots 0 \\ V_1 &= 1 \| \mathsf{devid} \| 0 \cdots 0 \end{aligned}$$

The encryption key K_e and the value W are computed as

$$\begin{aligned} K_e &= \mathsf{AES}(K, V_0) \\ W &= \mathsf{AES}(K, V_1) \end{aligned}$$

A counter ctr is computed by concatenating the first 104 bits of W, a 4-bit direction value dir, the 1-byte counter rc, and the 12-bit frame counter cnt:

$$ctr = \mathsf{msb}(W, 104) \| dir \| \mathsf{rc} \| \mathsf{cnt}$$

The parameter rc is an implicit counter which is incremented any time cnt wraps around. The value dir indicates the direction: if uplink, then $dir = 0$, otherwise $dir = 1$.

Let $ptxt$ be some n-bit plaintext data ($n \leq 96$) to be sent in a frame corresponding to counter cnt. The encryption of $ptxt$ is done as follows:

1. $msk = \mathsf{msb}(\mathsf{AES}(K_e, ctr), n)$
2. $ctxt = msk \oplus ptxt$

The encrypted data $ctxt$ is carried in the payload field.

[1] Regarding the interpretation and the consequences of the error event see Section A.

2.4 MAC Function

The message authentication code (MAC) is based on AES in CBC-MAC mode [6] with a null IV. The key used to compute a MAC tag is the static NAK key K. The MAC function outputs a tag which length (ranging from 2 to 5 bytes) depends on the size of the input data. The MAC tag of a downlink frame is 2-byte long.

In order to get an input which length is a multiple of 16 bytes, the data to be authenticated is padded with itself. For instance, if the data corresponds to 7 bytes $B_0\| \cdots \|B_6$, the input to the inner CBC-MAC computation is then $B_0\| \cdots \|B_6\|B_0\| \cdots \|B_6\|B_0\|B_1$. If the length of the data is a multiple of 16 bytes, then the data is unchanged.

Let *data* be a byte string. Let selfpad be the function that pads the input byte string with itself, and outputs a byte string which length is a multiple of 16 bytes. A MAC tag t (carried in the field mac) is computed as follows:

1. $\tilde{data} = \mathsf{selfpad}(data)$
2. $C = \mathsf{AES\text{-}CBC\text{-}MAC}(K, \tilde{data})$
3. $t = \mathsf{msb}(C, n)$

with $n \in \{16, 24, 32, 40\}$.

The MAC tag of an uplink frame is computed with the following *data* as input:

- $\mathsf{hdr}\|\mathsf{payload} = \mathsf{li}\|\mathsf{bf}\|\mathsf{rep}\|\mathsf{cnt}\|\mathsf{devid}\|\mathsf{payload}$ if the Sigfox encryption function is not used;
- $\mathsf{rc}\|\mathsf{hdr}\|\mathsf{payload} = \mathsf{rc}\|\mathsf{li}\|\mathsf{bf}\|\mathsf{rep}\|\mathsf{cnt}\|\mathsf{devid}\|\mathsf{payload}$ if the Sigfox encryption function is used (then payload carries the encrypted data).

The MAC tag of a downlink frame is computed with the following *data* as input (length in bit):

$$\mathsf{devid}\ (32)\|\mathsf{lsb}(\mathsf{cnt}, 8)\|\mathtt{0b0000}\ (4)\|\mathsf{msb}(\mathsf{cnt}, 4)\|\mathsf{payload}\ (64)\|\mathsf{msb}(\mathsf{devid}, 16)$$

where devid is encoded in little endian format.

3 Known Vulnerabilities in Sigfox

This section presents the vulnerabilities and attacks against Sigfox previously published (that we are aware of).

3.1 Frame Replay

3.1.1 Issue

The security in Sigfox is based on a static symmetric key shared between the end-device and the back-end network. This same key is used to compute a MAC tag for all frames. In order to forbid replays, counters are involved in the MAC tag computation:

- the at most 12-bit counter cnt when the frame is not encrypted,[2]
- cnt and the 8-bit counter rc when the frame is encrypted.

When cnt reaches its highest value, it is reset to 0. Therefore, the ability to replay frames is natural in Sigfox due to the short size of the message counter cnt. Yet, two cases must be considered depending on whether the frame is encrypted or not.

[2]The parameter cnt is encoded as a 12-bit field. Yet its maximum value varies. In the remainder of this document, we write $|\mathsf{cnt}| = i$ bits to indicate that the maximum value of cnt is $2^i - 1$.

3.1.2 Clear Frame

The maximum value for the message counter cnt is $2^i - 1$, $i \in \{7, 8, 9, 10, 11, 12\}$. Once cnt reaches its highest value, it is reset to 0. When cnt wraps around, the counter rc is also incremented. Nonetheless, this latter parameter is not involved in the MAC tag computation when encryption is not applied. Therefore when frames are protected only with a MAC tag, past frames become cryptographically valid anew when cnt is reset. Hence these previous frames can be replayed by an adversary to the back-end network as indicated by Lifchitz [7] and Euchner [5]. In addition, as mentioned by Euchner [5], the adversary can also replay downlink frames (to the end-device) if the bidirectional procedure is applied by the targeted end-device.

An end-device can send at most 140 frames per day with the "Platinum" subscription. This means that cnt wraps around every $2^{12}/140 \simeq 29.26$ days. Therefore, in such a case, roughly every month at most 4096 new uplink messages, and at most $29.26 \times 4 \simeq 117$ new downlink messages become available to the adversary for this replay attack.[3]

With this scenario, the adversary can deceive the end-device's owner (who receives the replayed uplink frames), or the end-device which can be an actuator, and executes actions upon reception of a downlink frame. In addition, each replayed uplink frame lowers the number of remaining frames (which is bounded per contract) that the legitimate end-device is allowed to send daily.

3.1.3 Encrypted Frame

The ability to replay uplink encrypted frames is questionable. Indeed, when encryption is applied, the 8-bit counter rc is involved in the MAC tag computation, in addition to cnt.

Let us consider $|\mathsf{cnt}| = 7$ bits, and assume that the end-device can send no more than $c = 2$ uplink frames per day with such a size for cnt. Then the maximum value for $\mathsf{rc}\|\mathsf{cnt}$ is reached after $2^{|\mathsf{rc}|+|\mathsf{cnt}|}/c = 2^{14}$ days $\simeq 45.5$ years. Yet, if the adversary is able to compel the end-device to send messages at a higher frequency (e.g., the end-device sends frames constantly), this amount can be reached much faster. For instance, the shortest uplink frame is 112-bit long. If the end-device uses a 600-baud symbol rate to send a frame, then the number of messages sent in one day at this speed reaches $24 \times 3600 \times 600/112 \simeq 2^{18.8} > 2^{15} = 2^{|\mathsf{rc}|+|\mathsf{cnt}|}$.

Likewise, let us assume that $c = 140$ uplink frames per day when $|\mathsf{cnt}| = 12$ bits. Then the maximum value for $\mathsf{rc}\|\mathsf{cnt}$ is reached after $2^{|\mathsf{rc}|+|\mathsf{cnt}|}/c = 2^{12.9}$ days $\simeq 20.8$ years. Nonetheless, at a speed of 112/600 second per frame, the maximum value for $\mathsf{rc}\|\mathsf{cnt}$ is reached after $\dfrac{2^{|\mathsf{rc}|+|\mathsf{cnt}|}}{24 \times 3600 \times 600/112} \simeq 2^{1.2}$ days (if uplink frames are sent continuously).

Therefore, assuming that the adversary is able to force a legitimate end-device to send more frames than allowed, the adversary is able to replay encrypted frames. This is possible either if rc wraps around, or if rc gets stuck to its highest value once the latter is reached, because cnt still wraps around.

3.2 Denial of Service

3.2.1 Issue

As explained in Section 2.2, the back-end network verifies the frame counter cnt according to an acceptance interval. The natural ability provided by Sigfox to replay previous (non encrypted) frames can be used to twist these rules, and harm the availability of the system.

Below we present a scenario and its variant from [5, 1] that allow performing a denial

[3] The time necessary for the counter to wrap around depends on the number of maximum daily uplink frames. The best case for the adversary corresponds to the "Platinum" subscription. That is, with respect to this attack, the more expensive the subscription, the less the security level.

of service attack (DoS). These scenarios rely upon a frame replay, hence are practicable if encryption is not applied. We later discuss what can be done if the frames are encrypted.

3.2.2 Attack

Let $\mathsf{cnt} = n$ be the counter value of the last frame sent by the legitimate (targeted) end-device, and received by the back-end network. In this scenario, presented by Euchner [5], the adversary makes use of a previous uplink frame such that

- its counter value $\mathsf{cnt} = n'$ belongs to the acceptance interval defined by n, and
- n' is as high as possible compared to n.

Since the frame replayed by the adversary is valid (with respect to the MAC tag, and the counter), n' becomes the new counter reference on the back-end side. Therefore any new frame sent by the legitimate end-device is rejected until its counter value exceeds n'.

For instance, let us assume that the adversary acts the same day as the last legitimate sent uplink frame (i.e., the elapsed time between the two frames is $d = 1$). If the maximum daily amount of uplink frames is $c = 140$ (corresponding to the "Platinum" subscription), then

$$\begin{aligned} 300 \times d &= 300 \\ c \times (d+2) &= 420 \end{aligned}$$

The adversary replays a previous uplink frame which counter is equal to $n' = n + 420$. Since $n' \leq n + \max(300 \times d, c \times (d+2))$, and the MAC tag is valid, the frame is accepted by the back-end network. All subsequent uplink frames sent by the legitimate end-device are silently discarded by the back-end network as long as their counter is lower or equal to $n' = n + 420$. The duration of this DoS is at least $420/140 = 3$ days. It may be longer if the legitimate end-device sends less than 140 frames per day.

Now, let us assume that $c = 2$ (corresponding to the "One" subscription). Then

$$\begin{aligned} 300 \times d &= 300 \\ c \times (d+2) &= 6 \end{aligned}$$

The adversary replays a previous uplink frame which counter is equal to $n' = n + 300$. Again, the frame is accepted by the back-end network. In this case, the duration of the DoS is at least $300/2 = 150$ days.

Coman, Malarski, Petersen, and Ruepp [1] describe a variant of the above scenario where the adversary replays not one but several frames. The adversary uses frames such that the counter gap between two frames it sends consecutively is the maximum possible. That is, $\max(300 \times d, c \times (d+2))$. Then the duration of the DoS is multiplied by the number of replayed frames.

For instance, with the same figures as above, in the case of the "Platinum" subscription, the adversary replays first a frame which counter is equal to $n + 420$, followed by frames which counter is respectively equal to $n + 840$, $n + 1260$, ..., $n + 3780$, and finally $n + 4095$. Doing so, the adversary replays the minimum number of frames, and achieves the longest duration for the DoS. That is, 10 frames are used, which causes an interruption in message delivery for $4095/140 = 29$ days at least.[4]

[4] Coman et al. present also another DoS attack based on a previous definition of the acceptance interval (which is described in [1] as static) [9]. This attack is now thwarted by the use of the evolving interval (which is computed in particular from the number d of elapsed days). Seemingly, the definition of the acceptance interval changed at some point in time after the publication of Coman et al.'s paper.

3.2.3 Downlink Frames

We observe that the scenario described in Section 3.2.2 does not only forbid the back-end network from receiving uplink frames from the legitimate end-device. It forbids also the latter from receiving downlink frames from the back-end network.

First, the back-end network sends a downlink frame only upon reception of an uplink frame (when the bidirectional procedure is applied). Therefore, if the adversary does not send further uplink frames, the back-end network will not transmit downlink frames. In addition, a downlink frame does not carry a frame's counter (presumably the downlink frame is computed from the corresponding uplink frame's counter, although this is not explicitly stated in the Sigfox specification [21]). It is highly likely that, upon reception of a downlink frame, the end-device verifies the MAC tag based on its current value of the frame's counter (that is, the uplink frame's counter). In the DoS scenario, the legitimate end-device and the back-end network are desynchronised with respect to the frame's counter (the back-end network takes as new reference a value which is much higher that the current counter value stored by the end-device). Consequently, even if the back-end network sends a downlink frame, the verification of its MAC tag by the end-device yields an error. Hence the downlink frame is rejected by the end-device. This shows that the DoS attack impacts not only on the back-end network but also on the legitimate end-device.

3.2.4 Encrypted Frames

The same scenario and variant presented in Section 3.2.2 may be practicable with encrypted frames under condition. This is possible if the adversary succeeds in forcing a legitimate end-device to send frames at a higher frequency (i.e., more daily frames than allowed) so that the extended 20-bit counter $\mathsf{rc}\|\mathsf{cnt}$ wraps around (see Section 3.1.3).

With respect to the variant described by Coman et al., the adversary can then replay much more than 10 frames, and each replayed frame extends the DoS by $420/140 = 3$ days at least.

3.3 MAC Tag Forgery: Exhaustive Search

In this section, we describe a scenario aiming at forging a valid uplink frame. That is, producing a valid MAC tag. This scenario holds whether encryption is applied or not.

3.3.1 Issue

This scenario is based on the fact that the frame's MAC tag in Sigfox is rather short (down to 2 bytes).

3.3.2 Uplink Frame

As indicated by Euchner [5], an adversary can try to forge a valid MAC tag by successive trials. The complexity of this scenario is $O(2^t)$ where t is the bit length of the MAC tag.

Sigfox allows two different symbol rates to transmit an uplink frame: 100 and 600 bauds. Let us consider an uplink frame with an empty payload (possibly with 1-bit data stored in the "length indicator" parameter li). In such a case the MAC tag is 2-byte long. The size of such a frame is 14 bytes = 112 bits (be the frame encrypted or not). In order to find a correct value for the MAC tag, $2^t = 2^{16}$ trials must be done by the adversary on average. That is, the adversary must choose a 2-byte value, build a frame, and send it to the back-end network. This translates into $2^{16} \times 112/600 \simeq 12233$ seconds $\simeq 3.4$ hours. With a 100-baud symbol rate, the duration of the attack is roughly 20.4 hours.

Likewise, a frame carrying a 12-byte payload is 26-byte long, with a 2-byte MAC tag. The duration of the same kind of attack corresponds to 6.3 (resp. 37.9) hours at 600 (resp.

100) bauds.

These figures correspond to an attack made with one end-device. Yet, the attack can be paralleled with multiple end-devices, which reduces its duration in proportion. Moreover, these figures are regulation respecting in regard to the transmission speed. Nonetheless, the adversary does not have to comply with these rules, and may push its end-devices to their speed limit.

According to [2], if the back-end network receives two many invalid frames, the corresponding end-device is blocked. Therefore, if the MAC forgery is not successful, this scenario can turn to a DoS attack.

3.3.3 Downlink Frame

The adversary can also try to forge a valid MAC tag for a downlink frame, as briefly mentioned by Euchner [5]. First the adversary must expect an uplink frame which bf parameter is set to 1 (which triggers a bidirectional exchange).

When the end-device sends such an uplink frame, it waits for some time for a downlink frame. The duration T_{rx} of the reception window depends on the geographic area. For instance, in Europe $T_{rx} = 25$ seconds [21].

Let v be the time needed by the end-device to receive and process a downlink frame, and $t = 16$ bits be the (fixed) size of a MAC tag in a downlink frame. The number of trials allowed to the adversary is T_{rx}/v, and the probability of success is $p = 2^{-t} \times T_{rx}/v$. For instance, assuming that $v = 1$ second, we have that $p \simeq 2^{-11.4}$. The back-end network uses a 600-baud symbol rate to send a downlink frame. Let us assume that the end-device is able to receive and process a frame at this corresponding speed.[5] A downlink frame is 28-byte long. Therefore, a downlink frame is sent in 224/600 second, and the probability of success is then $p = 2^{-t} \times T_{rx} \times 600/224 \simeq 2^{-9.9}$.

3.4 Key Extraction

In this section, we present yet another attack, although it is not related to the way the security mechanisms are defined and used in Sigfox. However, it poses a threat which, according to us, is worth to consider.

3.4.1 Issue

As indicated by Euchner [5], the static symmetric key NAK shared between the back-end network and the end-device is not always protected when it is stored in the end-device.

3.4.2 Attack

An adversary able to get a physical access to the end-device may extract the NAK key. For instance, Euchner validated such a possibility with a pycom SiPy 1.0 hardware module.

With the end-device's symmetric key, the adversary can passively decrypt encrypted uplink and downlink frames. It can also forge uplink and downlink frames. That is, it can impersonate the end-device to the back-end network, and conversely.

4 New Attacks against Sigfox

In this section we present two new attacks against Sigfox, and highlight a flaw in the encryption procedure.

The attacks that we present break data authenticity. The first attack allows replaying

[5] This implies in particular that the cryptographic computations are negligible compared to the communication process, which is likely the case in such a context.

downlink encrypted frames. The second attack allows forging uplink (clear or encrypted) frames.

4.1 Lack of Encryption: Single-bit Case

4.1.1 Issue

In Sigfox, data authenticity is mandatory. Sigfox provides also an optional encryption scheme.

Variable length data can be transmitted in an uplink frame: 1 to 12 bytes, and 1 bit. If the length is 1-byte long at least, data is carried in the payload field. If the data is 1-bit long (single-bit case), the payload field is empty, and data is carried in the "length indicator" li field, which is located in the frame's header hdr.

When encryption is activated, data is encrypted if it is carried in the payload. However, our experiments show that, in the single-bit case (i.e., empty payload), data is not encrypted.

It is unclear to us if the lack of encryption in the single-bit case is a choice or a bug. It may be possible that computing 128 bits in order to encrypt 1 bit is deemed to energy costly. In the other hand, it is also possible that the encryption function reads the data to be encrypted in the payload field, which would yield nothing when this field is empty.

4.1.2 Consequence

When encryption is activated, data is not always encrypted. In the single-bit case the plaintext data remains accessible to a passive eavesdropper.

The fact that a single bit be not encrypted may appear as not very significant. However, according to us, one should not presume which purpose the final user intends to achieve with a security mechanism. If a security protocol claims to provide some property (in that case, confidentiality), it must ensure the latter. Here, Sigfox does not achieve the intended security level.

4.2 Replay of Downlink Encrypted Frames

4.2.1 Issue

As explained in Section 3.1, Sigfox allows "naturally" frame replays because the message counter cnt is rather short, and wraps around when it reaches its maximum value. The size of this counter is at most 12 bits. Nonetheless, when encryption is activated, an additional 8-bit counter rc is involved in the MAC tag computation of an uplink frame. In such a case the possibility for this extended counter rc||cnt to wrap around is questionable (see Section 3.1.3). However, according to the Sigfox specification [21], the parameter rc is not involved in the MAC tag computation of a downlink frame even when it is encrypted.

4.2.2 Attack

Since only cnt is involved in the MAC tag computation of a downlink (encrypted or clear) frame, it is possible to replay downlink encrypted frames when the counter cnt wraps around. In addition, such encrypted frames can be replayed even if encryption is later deactivated. Indeed, the MAC tag is valid (anew), and the encrypted data will be accepted as clear data. In such a case, this may lead the end-device to adopt an incoherent (possibly harmful) behaviour because what is then taken as plaintext data is essentially random data. Conversely, the adversary can replay a previous clear frame once encryption is activated. The end-device will use the plaintext data as input to the decryption function, which yields pseudo-random data.

In response to the ability to replay (MAC only) uplink frames, Sigfox indicates (to

Euchner [2]) that activating encryption extends the frame's counter from 12 to 20 bits with the use of the 8-bit implicit counter rc, and thwarts frame replays. We observe that the way downlink encrypted frames are computed contradicts this argument raised by Sigfox.

4.3 MAC Tag Forgery: Completion Attack

In this section we present a scenario that allows an adversary to forge valid uplink frames (encrypted or not). Although this attack is an existential forgery (rather than universal), its complexity is $O(1)$ (in contrast to the attack described in Section 3.3). We describe unconditional and conditional forgeries.

We have experimentally validated the MAC tag forgeries first with the librenard library provided by Euchner [3]. We recall that librenard has been validated by Euchner with practical experiments done in interaction with the Sigfox back-end network. These experiments confirm in particular that librenard implements correctly the Sigfox MAC function. Secondly, we have validated the forgeries with real-life experiments done in interaction with the back-end network.

This attack scenario does not apply to downlink frames because they all have the same fixed size.

4.3.1 Issue

The MAC tag is computed in CBC-MAC mode (with AES). This mode is insecure for variable length inputs, and yet, in Sigfox, the data to be authenticated in an uplink frame may have different sizes (in a downlink frame, the length of payload is fixed to 8 bytes).

In order to extend the input data up to a multiple of 16 bytes, the data is padded with itself. For instance, if the input to the MAC function corresponds to seven bytes $m = B_0\| \cdots \|B_6$, the data in input to the inner CBC-MAC computation is then $B_0\| \cdots \|B_6\|B_0\| \cdots \|B_6\|B_0\|B_1$ (16 bytes). Now, if the input to the MAC function corresponds to the following 14 bytes $m' = B_0\| \cdots \|B_6\|B_0\| \cdots \|B_6$, then, after padding, the data in input to the inner CBC-MAC computation is $B_0\| \cdots \|B_6\|B_0\| \cdots \|B_6\|B_0\|B_1$. That is, the same data as for m. Consequently, the MAC tag of m' is equal to that of m (see Figure 4). This allows completion attacks against the MAC function in Sigfox.

Due to the constraints that bind the data length, the MAC tag length, and the value of the "length indicator" field li, not all kinds of modification are possible. Nonetheless, several are doable.

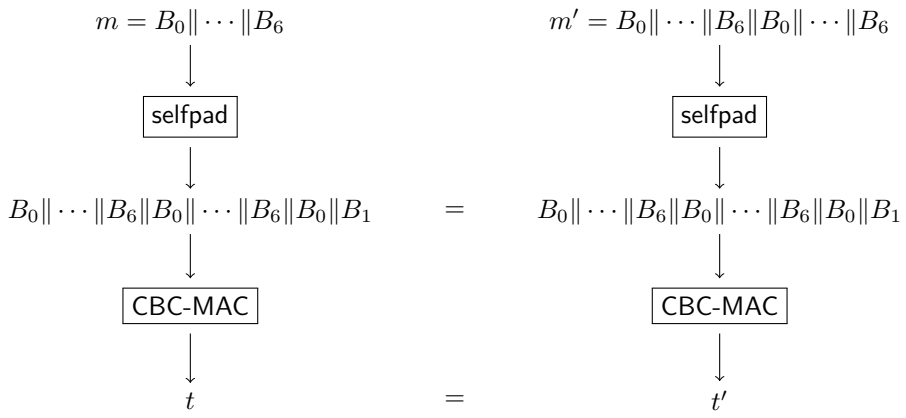


Figure 4: Basic principle of the completion attack against the Sigfox MAC function

4.3.2 Overview

The goal of the adversary is to forge an uplink frame with a valid MAC tag. To do so, the adversary reuses an existing valid uplink frame. Such a frame can be picked from the set of previous frames when they can be replayed (this excludes an encrypted frame), or can be a fresh uplink frame eavesdropped and blocked by the adversary (i.e., not received by the back-end network). In the latter case, the frame used can be encrypted or not. The constraints in order for the adversary to be successful are the following.

1. The "length indicator" parameter li must be the same in the forged frame and in the original frame used by the adversary. This guarantees also that the size of the MAC tag in the forged frame is equal to the size of the tag in the original frame.

2. The 16-byte blocks used as inputs to the inner CBC-MAC computation must be the same for the forged and the original frames. That is, when padded with itself the data of the forged frame must be equal to the data of the original frame when also padded with itself.

3. The payload length of the forged frame must be at most 12-byte.

There are two cases: when the frame is encrypted and when it is not (clear frame).

Table 2 lists the different values for the "length indicator" parameter li, and the corresponding length of the MAC tag mac in an uplink frame (encrypted or not). Table 3 summarises the different forgeries that we present next.

Table 2: li values and MAC tag sizes for an uplink frame

data	$\|\mathsf{hdr}\|\mathsf{payload}\|$ (byte) (clear frame)	$\|\mathsf{rc}\|\mathsf{hdr}\|\mathsf{payload}\|$ (byte) (encrypted frame)	li	$\|\mathsf{mac}\|$ (byte)
0b0	6	7	0b10	2
0b1	6	7	0b11	2
(empty)	6	7	0b00	2
1 byte	7	8		
4 bytes	10	11		
8 bytes	14	15		
12 bytes	18	19		
3 bytes	9	10	0b01	3
7 bytes	13	14		
11 bytes	17	18		
2 bytes	8	9	0b10	4
6 bytes	12	13		
10 bytes	16	17		
5 bytes	11	12	0b11	5
9 bytes	15	16		

4.3.3 Clear Frame

Let us consider first how the adversary can produce a forgery from a non-encrypted uplink frame.

The sizes $|\mathsf{hdr}\|\mathsf{payload}| \in \{7, 8, 12\}$ corresponding to $|\mathsf{payload}| \in \{1, 2, 6\}$ are of interest to the adversary.

Table 3: Summary of the forgeries for uplink frames. The "Cond." field indicates if some (probabilistic) condition must be fulfilled in order for the forgery to be possible. The sizes are given in byte.

Type of frame	Original frame	Forged frame		Cond.	(li, |mac|)
Clear	$\|pld\| = 1$	$pld' = pld\|\|hdr\|\|pld$	$\|pld'\| = 8$	no	$(\texttt{0b00}, 2)$
Clear	$\|pld\| = 2$	$pld' = pld\|\|hdr\|\|pld$	$\|pld'\| = 10$	no	$(\texttt{0b10}, 4)$
Clear	$\|pld\| = 6$	$pld' = pld\|\|hdr[0 \cdots 3]$	$\|pld'\| = 10$	no	$(\texttt{0b10}, 4)$
Encrypted	(empty)	$pld' = rc\|\|hdr\|\|rc$	$\|pld'\| = 8$	no	$(\texttt{0b00}, 2)$
Encrypted	$\|pld\| = 1$	$pld' = pld\|\|rc\|\|hdr$	$\|pld'\| = 8$	yes	$(\texttt{0b00}, 2)$
Encrypted	$\|pld\| = 4$	$pld' = pld\|\|rc\|\|hdr[0 \cdots 2]$	$\|pld'\| = 8$	yes	$(\texttt{0b00}, 2)$
Encrypted	$\|pld\| = 5$	$pld' = pld\|\|rc\|\|hdr[0 \cdots 2]$	$\|pld'\| = 9$	no	$(\texttt{0b11}, 5)$

Type "clear_1". Let us first consider a genuine frame

$$\begin{aligned} frame &= \mathsf{ft}\|\mathsf{hdr}\|\mathsf{payload}\|\mathsf{mac}\|\mathsf{crc} \\ &= ft\|hdr\|pld\|mac\|crc \end{aligned}$$

such that $|\mathsf{payload}| = |pld| = 1$ byte (ft, hdr, etc., correspond respectively to the current values of the parameters ft, hdr, etc.).

From $frame$, the adversary computes

$$\begin{aligned} frame' &= \mathsf{ft}\|\mathsf{hdr}\|\mathsf{payload}\|\mathsf{mac}\|\mathsf{crc} \\ &= ft'\|hdr'\|pld'\|mac'\|crc' \end{aligned}$$

as follows:

1. $hdr' = hdr$
2. $pld' = pld\|hdr\|pld$ (and $|pld'| = 8$ bytes)
3. The adversary chooses the frame type ft' in accordance with $|pld'|$.
4. $mac' = mac$
5. The adversary computes crc' from $hdr'\|pld'\|mac'$.

Since $|pld| = 1$ byte, we have that $(\mathsf{li}, |\mathsf{mac}|) = (\texttt{0b00}, 2)$ in $frame$. Since $|pld'| = 8$ bytes, $(\mathsf{li}, |\mathsf{mac}|)$ in $frame'$ must be, and is indeed, equal to $(\texttt{0b00}, 2)$ (because $hdr' = hdr$).

The MAC tag $mac' = mac$ is a valid tag for $frame'$. Indeed this tag is valid for $frame$. This means that the data used as input to the inner CBC-MAC computation for $frame$ is (size in byte)

$$hdr\ (6)\|pld\ (1)\|hdr\ (6)\|pld\ (1)\|\|\|hdr[0 \cdots 1]\ (2)$$

In turn, the data used as input to the inner CBC-MAC computation in order to verify the MAC tag mac' in $frame'$ is

$$\begin{gathered} hdr'\ (6)\|pld'\ (8)\|hdr'[0 \cdots 1]\ (2) \\ = \\ hdr\ (6)\|pld\ (1)\|hdr\ (6)\|pld\ (1)\|\|\|hdr[0 \cdots 1]\ (2) \end{gathered}$$

Since the MAC tag is computed with the same key, and the same input data in either case ($frame$ and $frame'$), we have that $mac' = mac$ is a valid MAC tag for $frame'$. Hence, $frame'$ is a valid uplink frame forged by the adversary.

Type "clear_2". The same holds with a genuine frame $frame = ft\|hdr\|pld\|mac\|crc$ with $|pld| = 2$ bytes. The adversary forges a frame $frame'$ carrying a payload (size in byte)

$$pld'\ (10) = pld\ (2)\|hdr\ (6)\|pld\ (2)$$

(with the corresponding frame type and CRC value). The frame $frame'$ is a valid frame for the MAC tag $mac' = mac$. Indeed, $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b10}, 4)$ in $frame$ and $frame'$, and

$$hdr'\ (6)\|pld'\ (10) = hdr\ (6)\|pld\ (2)\|hdr\ (6)\|pld\ (2)$$

This corresponds to the data in input to the inner CBC-MAC computation for $frame'$ and $frame$.

Type "clear_6". Another forgery is possible with an original frame $frame = ft\|hdr\|pld\|mac\|crc$ such that $|pld| = 6$ bytes. The adversary forges a frame $frame'$ carrying a payload (size in byte)

$$pld'\ (10) = pld\ (6)\|hdr[0 \cdots 3]\ (4)$$

(with the corresponding frame type and CRC value). The frame $frame'$ is a valid frame for the MAC tag $mac' = mac$. Indeed, $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b10}, 4)$ in $frame$ and $frame'$, and

$$hdr'\ (6)\|pld'\ (10) = hdr\ (6)\|pld\ (6)\|hdr[0 \cdots 3]\ (4)$$

which is the data in input to the inner CBC-MAC computation for $frame'$ and $frame$.

4.3.4 Encrypted Frame

Now let us consider how the adversary can produce a forgery from an encrypted uplink frame.

Unconditional Forgeries. We present first two unconditional forgeries.

Type "encrypted_5". Let us first consider a genuine encrypted frame

$$frame = ft\|hdr\|pld\|mac\|crc$$

with $|pld| = 5$ bytes. The MAC tag mac is computed with the following input data to the inner CBC-MAC function (size in byte)

$$rc\ (1)\|hdr\ (6)\|pld\ (5)\|rc\ (1)\|hdr[0 \cdots 2]\ (3)$$

where rc is the current value of the counter rc.

The adversary computes $frame' = ft'\|hdr'\|pld'\|mac'\|crc'$ as follows:

1. $hdr' = hdr$
2. $pld' = pld\|rc\|hdr[0 \cdots 2]$ (and $|pld'| = 9$ bytes)
3. The adversary chooses the frame type ft' in accordance with $|pld'|$.
4. $mac' = mac$
5. The adversary computes crc' from $hdr'\|pld'\|mac'$.

In order to verify the MAC tag mac', the data used as input to the inner CBC-MAC function is

$$rc\ (1)\|hdr'\ (6)\|pld'\ (9) = rc\ (1)\|hdr\ (6)\|pld\ (5)\|rc\ (1)\|hdr[0 \cdots 2]\ (3)$$

Moreover, $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b11}, 5)$ in $frame$ and $frame'$. Therefore $mac' = mac$ is a valid MAC tag for $frame'$. Hence $frame'$ is a valid encrypted frame forged by the adversary.

Type "encrypted_empty". A second unconditional forgery is the following. Let us consider an original empty encrypted frame $frame = ft\|hdr\|mac\|crc$. In such a case $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b00}, 2)$, and the data in input to the inner CBC-MAC function is (size in byte)

$$rc\ (1)\|hdr\ (6)\|rc\ (1)\|hdr\ (6)\|rc\ (1)\|hdr[0]\ (1)$$

The adversary computes $frame' = ft'\|hdr'\|pld'\|mac'\|crc'$ with $hdr' = hdr$, and $pld' = rc\|hdr\|rc$. The values ft' and crc' are computed in accordance with the other fields of $frame'$. The payload pld' is 8-byte long, which corresponds also to $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b00}, 2)$.

The data used as input to the inner CBC-MAC function in order to verify mac' is

$$rc\ (1)\|hdr'\ (6)\|pld'\ (8)\|rc\ (1) = rc\ (1)\|hdr\ (6)\|rc\ (1)\|hdr\ (6)\|rc\ (1)\|rc\ (1)$$

Hence, $mac' = mac$ is a valid MAC tag for $frame'$ if $hdr[0] = rc$.

The first byte of the header hdr corresponds to (length in bit)

$$\mathsf{li}\ (2)\|\mathsf{bf}\ (1)\|\mathsf{rep}\ (1)\|\mathsf{msb}(\mathsf{cnt}, 4)$$

Since $frame$ carries no data, $\mathsf{li} = \mathtt{0b00}$. The parameter bf can be equal to 0 (unidirectional procedure) or 1 (bidirectional procedure), and rep is always equal to 0. Therefore $rc = hdr[0]$ implies that

$$rc = \begin{array}{|cc|c|c|cccc|} \hline \multicolumn{2}{|c|}{li} & bf & rep & \multicolumn{4}{c|}{\mathsf{msb}(cnt, 4)} \\ \hline b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \hline \end{array} = \begin{array}{|cc|c|c|cccc|} \hline 0 & 0 & \cdot & 0 & \cdot & \cdot & \cdot & \cdot \\ \hline \end{array}$$

where cnt, li, bf, rep are respectively the current values of the message counter cnt, the li, bf, and rep parameters in $frame$.

Therefore, the adversary can forge a valid frame $frame'$ with any original frame $frame$ which fulfils the following characteristics:

- $frame$ carries an empty payload (i.e., $li = \mathtt{0b00}$), and
- $cnt = j \times 2^8 + i$, and
- $rc = bf \times 2^5 + j$,

with $(i, j) \in \{0, \ldots, 255\} \times \{0, \ldots, 15\}$, and $bf \in \{0, 1\}$.

The number of such encrypted uplink frames that the adversary can forge for a given end-device is at most $2^4 \times 2^8 = 2^{12}$ (bf takes only one value in $\{0, 1\}$ for each possible value $rc\|cnt$). This figure does not take into account the number of times the counter cnt is reset, which multiplies in proportion the number of usable uplink frames.

Conditional Forgeries. Now we describe two conditional forgeries.

Type "encrypted_1". The first conditional possibility is the following. From a genuine encrypted frame $frame$ with $|pld| = 1$ byte, the adversary computes $frame' = ft'\|hdr'\|pld'\|mac'\|crc'$ with $hdr' = hdr$, and $pld' = pld\|rc\|hdr$. The values ft' and crc' are computed in accordance with the other fields of $frame'$. We have that $|pld| = 1$ byte and $|pld'| = 8$ bytes, which both correspond to $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b00}, 2)$. The MAC tag mac in $frame$ is computed with the following input data to the inner CBC-MAC function (size in byte)

$$rc\ (1)\|hdr\ (6)\|pld\ (1)\|rc\ (1)\|hdr\ (6)\|pld\ (1)$$

The MAC tag mac' in $frame'$ is verified with the following input data to the inner CBC-MAC function

$$rc\ (1)\|hdr'\ (6)\|pld'\ (8)\|rc\ (1) = rc\ (1)\|hdr\ (6)\|pld\ (1)\|rc\ (1)\|hdr\ (6)\|rc\ (1)$$

Therefore, $mac' = mac$ is a valid MAC tag for $frame'$ if $pld = rc$. Since pld corresponds to encrypted data, the probability of success in this case is roughly 2^{-8}.

Type "encrypted_4". The second conditional possibility is the following. From a genuine encrypted frame $frame$ with $|pld| = 4$ bytes, the adversary computes $frame' = ft'\|hdr'\|pld'\|mac'\|crc'$ with $hdr' = hdr$, and $pld' = pld\|rc\|hdr[0 \cdots 2]$. The values ft' and crc' are computed in accordance with the other fields of $frame'$. We have that $|pld| = 4$ bytes and $|pld'| = 8$ bytes. Both cases correspond to $(\mathsf{li}, |\mathsf{mac}|) = (\mathtt{0b00}, 2)$. The MAC tag mac in $frame$ is computed with the following input data to the inner CBC-MAC function

$$rc\ (1)\|hdr\ (6)\|pld\ (4)\|rc\ (1)\|hdr[0 \cdots 3]\ (4)$$

The MAC tag mac' in $frame'$ is verified with the following input data to the inner CBC-MAC function (size in byte)

$$rc\ (1)\|hdr'\ (6)\|pld'\ (8)\|rc\ (1)$$
$$=$$
$$rc\ (1)\|hdr\ (6)\|pld\ (4)\|rc\ (1)\|hdr[0 \cdots 2]\ (3)\|rc\ (1)$$

Therefore, $mac' = mac$ is a valid MAC tag for $frame'$ if $rc = hdr[3] = devid[1]$, where $devid$ is the value of the (targeted) end-device's identifier (encoded in little endian format).

The end-device's identifier is fixed, and the parameter rc is a counter. Therefore, in this case, for each end-device, the adversary can forge as many uplink encrypted frame as distinct values for cnt (unless rc wraps around). For instance, with respect to the "Platinum" subscription, the number of forgeries is 2^{12}.

Other Types of Forgery. Other kinds of forgery are possible but with stronger constraints (i.e., equality between two bit strings which length is higher than 8 bits), hence lower probability of success.

4.3.5 Consequence

The ability to forge a valid uplink frame allows the adversary to impersonate the legitimate end-device to the back-end network. Moreover, each forgery lowers the number of remaining uplink frames (which is bounded per contract) that the end-device is allowed to send (the same holds regarding the forgery presented in Section 3.3). This is all the more problematic as this number can be rather low.

The back-end network accepts the forged frame and relays it (i.e., the data included therein) to the service provider because it is cryptographically valid. What happens to that point depends on the application layer. For instance a format check may detect a discrepancy and reject the application data. Yet, we point out that the first bytes in the forged frame correspond to the application data of the genuine frame it originates from, as shown in Table 3 (when encryption is activated, these genuine first bytes in the forged frame decrypt correctly).[6] The format check can be based on some header within the application data (e.g., equality test with an expected byte string). This header in the forged frame being then the same as in the genuine frame, the data carried in the forged frame may be accepted by the service provider in the end.

[6] Except if the genuine frame is an encrypted empty frame.

4.3.6 Experiments

We have validated the MAC tag forgeries that we describe in two ways.

"Offline" Experiments. First, we have used the librenard library developed by Euchner [3]. Librenard implements the Sigfox cryptographic functions (except the encryption function). We have completed librenard in order to support encryption. That is, this library implements the same cryptographic functions as a legitimate Sigfox end-device.

Euchner has validated librenard with practical experiments done in interaction with the Sigfox back-end network [5]. With the NAK key of a legitimate end-device, he has computed uplink frames for all possible payload length, but an empty content.[7] The computed frames have been sent to the Sigfox back-end network and accepted by the latter. This confirms in particular that librenard implements correctly the Sigfox MAC function. In addition, librenard provides a function which takes as input an uplink frame, and verifies it (including the MAC tag). We have used this function to successfully validate all the MAC tag forgeries that we describe.

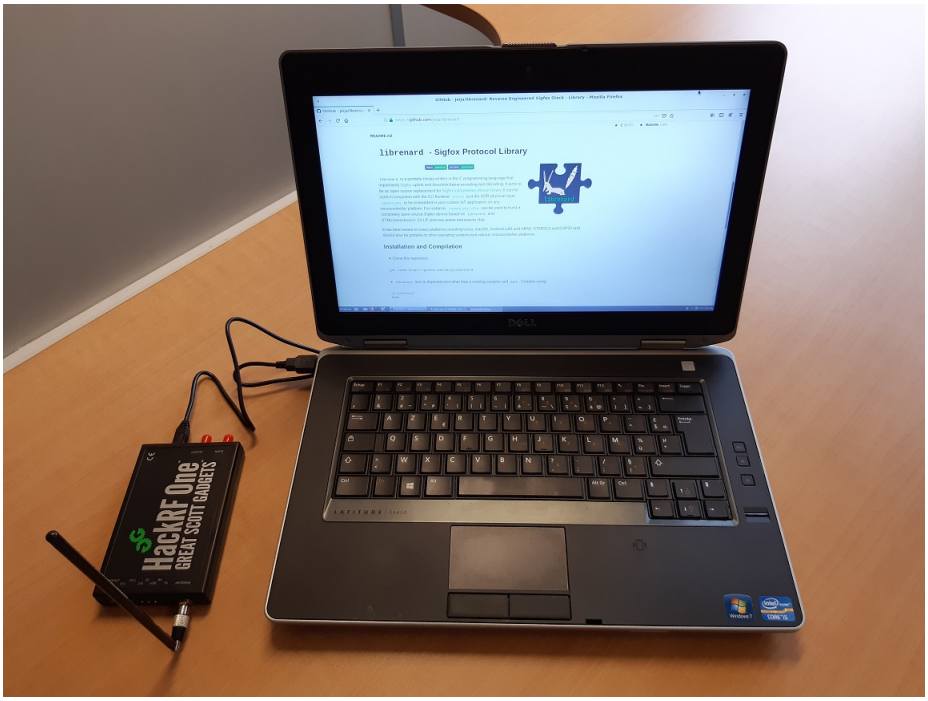


Figure 5: Experiment bench

Real-life Experiments. Secondly, we have conducted experiments in real conditions of use in interaction with the Sigfox back-end network. For each forgery type, we have generated several genuine uplink (clear or encrypted) frames with the NAK key corresponding to our legitimate end-device. From these frames, we have computed forgeries (without the NAK key). Each forged frame has been transmitted to the back-end network. The latter has accepted all the forged frames.

The only forgery type that we have not been able to test in interaction with the back-end network is the "encrypted_4" type (i.e., from a 4-byte genuine encrypted payload).

[7] An empty content corresponds to an uplink frame with no application data. In a single-bit frame, the payload is empty, and the 1-bit application data is included in the frame's header.

Table 4: Samples of forged frames. If the uplink frame is encrypted, the data received on the back-end is first decrypted, and then stored. The type "encrypted_4" has not been tested in real-life experiments.

Forgery type Genuine frame Forged frame Data stored on the back-end network
clear_1 08d0046895e410100f9b12ac8 6110046895e4101000046895e410100f9b1dff7 000046895e410100
clear_2 35f8049895e41010000b9a169493c11 94c8049895e410100008049895e41010000b9a169493657 00008049895e41010000
clear_6 611804c895e4101000000000000a749be547739 94c804c895e41010000000000000804c895ea749be5448f3 000000000000804c895e
encrypted_empty 06b0001895e4101adcf6d5f 6110001895e4101000001895e410100adcfe183 81879dc719010339
encrypted_1 08d0014895e4101006ddc072e 6110014895e410100000014895e41016ddccbe0 731180c554618155
encrypted_4 [35f0001895e4101e20095ebbb465029] [6110001895e4101e20095eb5e000189bb463b0d] [00000000697f3bd5]
encrypted_5 611e063895e410182d8c8538ff49fb6d41c149e 94ce063895e410182d8c8538f00e06389f49fb6d41c4356 0000000000e4bd138d

Indeed, in order for this forgery to be possible, it must hold that $rc = devid[1]$ where rc is the value of the counter rc, and $devid$ is the end-device's identifier (encoded in little endian format). In order for rc to be equal to some value x, the counter cnt must wrap around x times. That is, the number of uplink frames which must be sent is $x \times 2^{12}$ (for a "Platinum" subscription with at most 140 daily uplink messages). This corresponds to $x \times 2^{12}/140$ days at least. Given the devid value attributed to our end-device, this would have taken too long in order to reach the corresponding value for rc. Nonetheless, we stress that even this forgery type has been successfully validated with the verification function provided in librenard.

Figure 6 corresponds to screen shots made, from top to bottom, of two forged frames of type "clear_6" (the first original frame is made of 6 zero bytes, the second one of random bytes), and two forged frames of type "encrypted_5" (the first original frame is made of 5 zero bytes, the second one of random bytes) received on the back-end network. Table 4 lists an example of each forgery type.

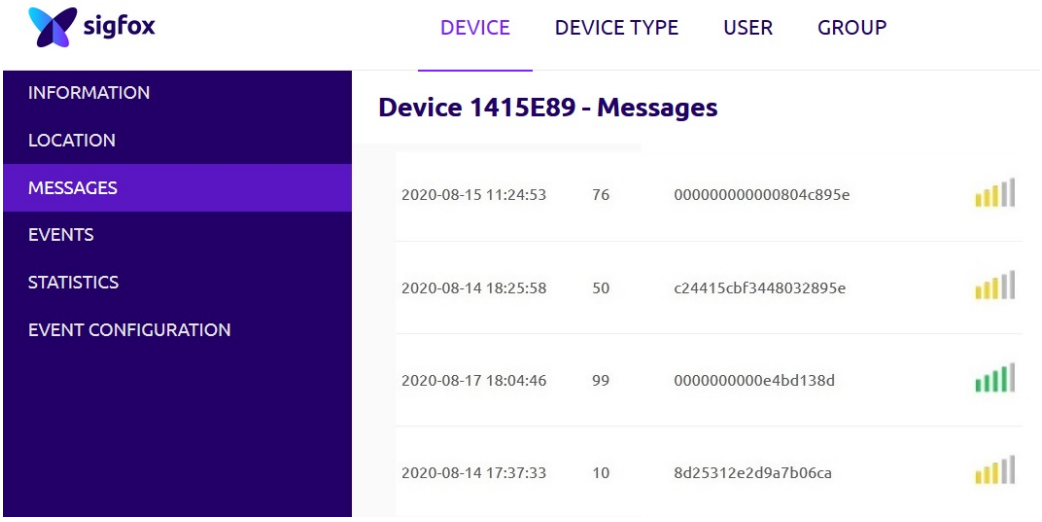


Figure 6: Screen shots of forged frames accepted by the Sigfox back-end network. From top to bottom, the two pairs of frames correspond respectively to the forgery types "clear_6" and "encrypted_5". For each forgery type, the pair of genuine frames corresponds respectively to zero bytes and random bytes in the payload.

Experiment Bench. The experiments have been done with a laptop Dell Latitude E6430 running Debian 10.5 with the (completed) librenard library installed on it. The radio communication has been managed with the module HackRF One [13], and the renard-phy scripts from Euchner [4] (see Figure 5).

5 Counter-measures

In this section we present the counter-measures that we propose in order to thwart the attacks described in Sections 3 and 4. Table 5 summarises the different counter-measures.

5.1 Frame Replay

The frame replays presented in Sections 3.1 and 4.2 are possible because the maximum value for the cnt counter can be rather low. A simple way to fix this issue is to extend this

Table 5: Proposed counter-measures

Attack	Counter-measure
Frame replay	Extended (implicit) message counter
Downlink encrypted frame replay	
DoS	
Forgery: exhaustive search	Longer MAC tag
Forgery: completion attack	CMAC mode
Lack of encryption	Encryption
Key extraction	Secure element

counter. To do so, we can use the rc counter. First we recommend the parameter rc to be involved in the MAC tag computation in any case (i.e., be the frame encrypted or not). That is, the input data to the Sigfox MAC function is prepended with rc, and becomes

- for an uplink frame: $\mathsf{rc}\|\mathsf{hdr}\|\mathsf{payload} = \mathsf{rc}\|\mathsf{li}\|\mathsf{bf}\|\mathsf{rep}\|\mathsf{cnt}\|\mathsf{devid}\|\mathsf{payload}$;
- for a downlink frame:
  - $\mathsf{rc}\|\mathsf{devid}\|\mathsf{lsb}(\mathsf{cnt},8)\|\mathsf{0b0000}\|\mathsf{msb}(\mathsf{cnt},4)\|\mathsf{payload}\|\mathsf{msb}(\mathsf{devid},16)$, or
  - $\mathsf{lsb}(\mathsf{rc},4)\|\mathsf{devid}\|\mathsf{lsb}(\mathsf{cnt},8)\|\mathsf{msb}(\mathsf{rc},4)\|\mathsf{msb}(\mathsf{cnt},4)\|\mathsf{payload}\|\mathsf{msb}(\mathsf{devid},16)$.

Now we estimate how long the size of rc must be.

Let us assume that the bit length of cnt goes with the maximum amount c of daily uplink frames. That is, we can have $(|\mathsf{cnt}|, c) = (12, 140)$ at most (corresponding to the "Platinum" subscription), or $(|\mathsf{cnt}|, c) = (7, 2)$ at least (corresponding to the "One" subscription), but not $(|\mathsf{cnt}|, c) = (7, 140)$. Let n be the lifespan of an end-device. Then, the maximum number of uplink messages an end-device may send during its whole lifetime is $n \times c$. Therefore we must have

$$2^{|\mathsf{rc}|+|\mathsf{cnt}|} \geq n \times c$$

That is,

$$|\mathsf{rc}| \geq \log_2(n \times c) - |\mathsf{cnt}|$$

This implies

- $|\mathsf{rc}| \geq 7$ if $(|\mathsf{cnt}|, c) = (12, 140)$,
- $|\mathsf{rc}| \geq 6$ if $(|\mathsf{cnt}|, c) = (7, 2)$.

with $n = 10$ years.

The current size of rc (8 bits) seems then already sufficient. But this result assumes that the end-device respects the limitation in the number of daily uplink frames. Yet, it may be possible that an adversary succeed in forcing an end-device to send uplink frames at will. That is, possibly at a frequency higher than $c = 140$ frames per day.

Let v be the minimum time to transmit one uplink frame that an adversary may impose to an end-device. The number of uplink frames is then at most n/v. For instance, if $n = 10$ years and $v = 112/600$ second, then $n/v < 2^{31}$. In such a case, $|\mathsf{rc}| = 31 - |\mathsf{cnt}| \in \{19, \ldots, 24\}$.

5.2 DoS

The attack scenario presented in Section 3.2.2 is based on the fact that frames can be replayed in Sigfox. Therefore the mitigation is the same as for the replay attacks (see Section 5.1).

5.3 MAC Tag Forgery: Exhaustive Search

In order to mitigate the MAC tag forgery possible by exhaustive search, and described in Section 3.3 (or the possible DoS attack against the end-device if the forgery is unsuccessful), we recommend to increase the lowest size of the MAC tag for the uplink frames.

If we assume that the maximum symbol rate used by the adversary to send an uplink frame is 600 bauds, the duration of the attack is $2^t \times 112/600$ seconds for the shortest uplink frame, and a t-bit MAC tag. This translates into 25.42 years if $t = 32$. Yet, the adversary can parallel its attack with multiple end-devices. Therefore, being conservative, we would recommend a 4 or 5-byte MAC tag for an uplink frame, whatever its length.

In such a case, the blocking procedure against an end-device which sends too many invalid frames can be removed (unless it is still necessary in order to mitigate DoS attacks targeting the back-end network).

5.4 MAC Tag Forgery: Completion Attack

The attack described in Section 4.3 is possible because the end-device uses the same static NAK key, and the Sigfox MAC function is based on the CBC-MAC mode, which is insecure for variable length inputs.

This issue can be easily fixed. Instead of using the CBC-MAC mode, the MAC function can rely upon the CMAC mode [22]. CMAC is built upon the same underlying CBC operation as CBC-MAC but makes use of two additional sub-keys (which would be static in the case of Sigfox), and an optional specific padding. This allows variable length inputs to the MAC function, and thwarts the completion attack.[8]

5.5 Lack of Encryption

When encryption is activated, in the single-bit case (i.e., empty payload with 1-bit data in the frame's header), data must be encrypted in the same way as when data is carried in the payload field.

5.6 Key Extraction

As indicated by Sigfox [20], in order to mitigate the possibility that the end-device's static key NAK be extracted, the key can be stored in a secure element (e.g., STMicroelectronics STSAFE-A1SX [25], WISeKey VaultIC184 [26]).

6 Conclusion

Sigfox is a communication and security protocol which allows setting up low-power wide area networks for the IoT. Currently, Sigfox operates in 72 countries on all continents. The multiple networks cover 1.3 billion people, and represent 56 million daily messages from 17 million IoT devices.

In this paper, we have first provided a unified and detailed description of the security mechanisms used in Sigfox. Such a description remains, to the best of our knowledge, incomplete in the official Sigfox radio specification, and scattered over several other (official and unofficial) documents.

Next, we have recapitulated and described with details the attacks that have been proposed formerly by other authors. We have also extended several of these attacks with scenarios that have not been considered so far.

[8] In previous Sigfox documents, it is indicated that the MAC function is based on HMAC [10, 19]. Assuming that the wording used in these documents is correct, the change from HMAC to CBC-MAC results then in a reduction of the security level.

Furthermore, we have made a security analysis of the radio interface in Sigfox. That is, the security mechanisms used to protect data exchanged back and forth between a remote end-device and the back-end network. We have presented a flaw that affects the encryption procedure, and is detrimental to data confidentiality. In addition, we have described new attacks against Sigfox. Namely, we have described how to replay downlink encrypted frames, and forge valid (encrypted or clear) uplink frames. These attacks break data authenticity with complexity $O(1)$ (in contrast to previous attacks against Sigfox), and allow deceiving the end-device or the back-end network. We have validated the MAC tag forgeries that we describe with practical real-life experiments.

The attacks that we have proposed do not exploit potential implementation or hardware bugs. They do not imply a physical access to any equipment (in particular a legitimate end-device). They are independent of the means used to protect the secret parameters (e.g., a secure element in the end-device). They depend exclusively on the peculiarities of the Sigfox MAC and encryption functions. The adversary needs only to act on the air interface.

Finally, we have presented efficient counter-measures which are likely straightforward to implement for most of them. They allow thwarting all the aforementioned attacks.

We responsibly disclosed our findings (attacks and counter-measures) to Sigfox.

The theoretical principle of the attacks that we have described is well-known within the cryptographic community. The detailed specifications of the Sigfox system remained confidential for many years even after the deployment of several Sigfox networks. According to us, more openness while designing the security mechanisms would have highly likely prevented the flaws that harm the system. This paper illustrates, if still necessary, that one can hardly expect better than insecurity through obscurity.

Acknowledgment

We thank Florian Euchner and Paul Pinault for their previous work on Sigfox.

References

[1] Coman, F.L., Malarski, K.M., Petersen, M.N., Ruepp, S.: Security Issues in Internet of Things: Vulnerability Analysis of LoRaWAN, Sigfox and NB-IoT. In: 2019 Global IoT Summit (2019)

[2] Euchner, F.: Hunting the Sigfox – Wireless IoT Network Security (December 2018), https://jeija.net/renard-slides/

[3] Euchner, F.: librenard – Sigfox Protocol Library (September 2018), https://github.com/Jeija/librenard

[4] Euchner, F.: renard-phy – Sigfox Protocol Physical Layer (September 2018), https://github.com/Jeija/renard-phy

[5] Euchner, F.: Sigfox Radio Protocol Overview and Specifications (December 2018), https://github.com/Jeija/renard-spec/releases

[6] ISO/IEC: ISO/IEC 9797-1:2011 – Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher (2011)

[7] Lifchitz, R.: IoT & Sigfox security (November 2016), https://speakerdeck.com/rlifchitz/iot-and-sigfox-security, Cyber Security Alliance Conference

[8] LoRa Alliance Technical committee: LoRaWAN 1.0.3 Specification (July 2018), https://lora-alliance.org/resource-hub/lorawanr-specification-v103

[9] Malarski, K.M.: Personal communication (December 2020)

[10] Mallart, R.: IoT Solutions to a Telecom Paradigm Shift (October 2016), https://www.tue.nl/en/our-university/calendar-and-events/2016-research-retreat/

[11] National Institute of Standards and Technology: NIST FIPS 197 Specification for the Advanced Encryption Standard (AES) (November 2001), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[12] National Security Agency's Research Directorate: Ghidra, https://ghidra-sre.org/, v9.1.2

[13] Ossmann, M.: HackRF, https://greatscottgadgets.com/hackrf/

[14] Pinault, P.: Stop telling me Sigfox is clear payload, for real you're just lazy ;) (December 2018), https://www.disk91.com/2018/technology/sigfox/stop-telling-me-sigfox-is-clear-payload-for-real-youre-just-lazy/

[15] Sigfox: Coverage, https://www.sigfox.com/en/coverage, retrieved July 14, 2020

[16] Sigfox: Our story, https://www.sigfox.com/en/sigfox-story, retrieved December 15, 2020

[17] Sigfox: Sequence number: general knowledge, https://support.sigfox.com/docs/sequence-number:-general-knowledge, retrieved December 15, 2020

[18] Sigfox: Sequence number: troubleshooting, https://support.sigfox.com/docs/sequence-number:-troubleshooting, retrieved December 15, 2020

[19] Sigfox: Sigfox – Technical Overview (May 2017), https://www.disk91.com/wp-content/uploads/2017/05/4967675830228422064.pdf

[20] Sigfox: Secure Sigfox Ready devices – Recommendation guide (June 2018), https://www.aerea.nl/wp-content/uploads/2018/06/Secure-Sigfox-Ready-devices-recommendation-guide-II.pdf

[21] Sigfox: Sigfox connected objects: Radio specifications (February 2020), https://build.sigfox.com/sigfox-device-radio-specifications, ref. EP-SPECS, rev. 1.5

[22] Song, J., Poovendran, R., Lee, J., Iwata, T.: The AES-CMAC Algorithm (June 2006), https://tools.ietf.org/html/rfc4493, RFC 4493

[23] Sornin, N.: LoRaWAN 1.1 Specification (October 2017), https://lora-alliance.org/resource-hub/lorawantm-specification-v11

[24] STMicroelectronics: X-CUBE-SFOX – STM32 Sigfox software expansion for STM32Cube, https://www.st.com/en/embedded-software/x-cube-sfox.html

[25] STMicroelectronics: STSAFE-A1SX (June 2019), https://www.st.com/resource/en/datasheet/stsafe-a1sx.pdf

[26] WISeKey: VaultIC184 – Sigfox Secure Element (March 2020), https://docs.wisekey.com/site/justdownload.html?id=104

A Error Event with the Frame Counter

Sigfox technical documents describe the error case as follows: "*The "out of message sequence" error event will cause an* ***interruption in message delivery*** [...] *Incoming messages will be* ***discarded*** *until the* ***disengage sequence number*** *feature is used.*" [17], and: "*When the* ***error*** *event has been raised by the cloud, the only way to* ***resume*** *message delivery is to use the* ***Disengage sequence number*** *feature.* [...] *disengaging the sequence number will cause the sequence number received along the* ***next message*** *to be recorder by the cloud as the* ***reference*** *for future comparisons,* ***whatever its value might be****.*" [18] (highlighted in the original text).

These explanations seem to imply that, as soon as the frame counter cnt is higher than the maximum value allowed ($\mathsf{cnt} = n' > n + ma$), *any* incoming frame is discarded, including a frame which counter belongs to the acceptance interval ($n < n' \leq n + ma$). The experiments that we did do not confirm this understanding.

More precisely we have sent a frame which counter is well above the acceptance interval ($n' > n + ma$). Then an error was raised on the back-end side, and the frame discarded. Next, we have sent a frame which counter is valid ($n < n' \leq n + ma$). This frame was accepted, and the corresponding data appeared on the back-end side. Therefore it seems that these technical documents consider only the case when the end-device's counter reaches suddenly a value higher than the acceptance interval, and keeps increasing. Then all subsequent frames are computed based on an invalid counter ($n' > n + ma$) on the back-end perspective. Hence these frames are all discarded by the latter. In such a case, the disengage procedure allows resetting the frame counter on the back-end side. The latter takes then this value as the new reference for the counter, which indeed resumes the message delivery.

If the error event would interrupt the delivery of *any* subsequent frames (whatever their counter), then this would enable a simple DoS attack, again based on the ability to replay (clear) frames. The scenario is the following. The adversary chooses a previous frame such that its counter cnt is strictly higher than the acceptance interval defined by the counter $\mathsf{cnt} = n$ of the last uplink frame received by the back-end network. That is, the counter of the replayed frame is chosen as $n' > n + ma$. When the back-end network receives this frame by the adversary, it raises an error which would then interrupt the delivery of the subsequent (legitimate) uplink frames. This DoS would last until the end-device's owner executes the "disengage" procedure. However, all uplink frames sent in the meantime would have been lost. We stress that our experiments do *not* show that this scenario is practicable.