End-to-end security in LoRa and NB-IoT sensor networks.

Niek van Noort
Security and Network Engineering
University of Amsterdam
Amsterdam, Netherlands
niek.vannoort@os3.nl

Jason Kerssens
Security and Network Engineering
University of Amsterdam
Amsterdam, Netherlands
jason.kerssens@os3.nl

***Abstract*—In this research we discuss the authentication, confidentiality and integrity of user data sent over the Low Power Wide Area Network (LPWAN) technologies, LoRa and NB-IoT. In particular, we look at possibilities for end-to-end security. The use case of this research is an IoT sensor network, where sensors report back to a monitoring platform. LoRa already supports end-to-end confidentiality, however, integrity is implemented in a hop-by-hop manner. NB-IoT on its own does not provide any end-to-end security, but hop-by-hop authentication, confidentiality and integrity can be achieved by trusting a Mobile Network Operator (MNO). Therefore, we have added end-to-end security mechanisms on top of LoRa and NB-IoT in the application layer, by using AES-GCM and AES-CMAC. Furthermore, we measured the effect of our security additions on the latency between the IoT sensors and the monitoring platform.**

I. Introduction

IoT has grown from small scale integration of devices at home to large scale IoT networks with national coverage [1]. For example, IoT sensors are being deployed in the Netherlands to monitor critical infrastructures, such as tunnels or bridges. Security is an important aspect of such a network, as the sensor data must be accessible at different locations, e.g. a monitoring platform, and must therefore travel through networks controlled by third parties. Data sent by IoT devices has to be verified, as acceptance of false data could have catastrophic consequences in critical infrastructures. Additionally, data has to be encrypted to ensure confidentiality.

To achieve cost efficiency, wireless communication is used by most IoT sensors, making deployment easier in structures that were initially not designed with IoT in mind. There are many technologies with which wireless networks can be established. For IoT devices, especially when it comes to IoT sensors, low-power wide area network (LPWAN) technologies are utilised, due to the low power requirements of IoT. Prominent technologies pertaining to LPWAN include Long Range (LoRa), Sigfox, and NarrowBand IoT (NB-IoT) [2]. For IoT devices that do not require low energy consumption, LTE can be used for wireless communication instead. As LTE offers a higher data rate, there might be situations in which LTE is preferred over LPWAN technologies.

In this research we will focus on the capability of sending data securely with LoRa, and NB-IoT. The use case of our research is a sensor network for Dutch critical infrastructure, where the sensors have a send rate of at least one message per minute. Sigfox allows the sensors to send 140 messages per day [2]. Therefore, it is not suitable for our use case and is left out of scope. We will identify the risks and capabilities in using different transport mechanisms and make a comparison between LoRa and NB-IoT. We will mainly be looking at the ways these technologies can provide integrity of data and guarantee the authenticity of its sender, without relying on cryptographic keys known by third parties, e.g. NB-IoT providers. Additionally, if end-to-end confidentiality, integrity and authenticity are not supported by an LPWAN technology, we will consider the possibility of implementing this ourselves by encrypting and signing monitoring data.

A. Research questions

We defined the following main research question:

How can end-to-end confidentiality, authentication, and data integrity be achieved with IoT devices that make use of LoRa and NB-IoT?

To answer our main research question we have defined the following subquestions:

- What capabilities do LoRa and NB-IoT have in terms of confidentiality, authentication and integrity?
- What security risks are present in LoRa and NB-IoT, relating to confidentiality, authentication, and data integrity?
- What security measurements could be taken by an administrator of an IoT network to achieve end-to-end security?

B. Structure

The structure of this paper is as follows. In Section II we look into the architecture and security measures present in LoRa, NB-IoT, and LTE. In Section III we discuss related work looking at security flaws found in LoRa and NB-IoT. In Section IV we discuss how we implemented additional security measures for both LoRa and NB-IoT and explain how we measured the impact this has on the performance of the aforementioned technologies. In Section V we show the results of our measurements. In Section VI we discuss our implementation and our results. Finally, in Section VII we give

a conclusion and in Section VIII we will give suggestions for future work.

II. Background

In this section we describe the security measures that are already in place for LoRa and NB-IoT. Furthermore, we will give an overview of the conventional architectures that are used for these technologies.

A. LoRa

1) LoRa Architecture: LoRaWAN networks consist of end-devices, gateways, network servers, application servers, and join servers.

End-devices communicate with one or multiple gateways over LoRa. As such, a single message could be accepted and relayed by multiple gateways at once. End-devices may, depending on the activation procedure, contain two identifiers, DevEUI and JoinEUI, as well as two cryptographic root keys, AppKey and NwkKey. The DevEUI uniquely identifies an end-device. The JoinEUI uniquely identifies the Join Server with which the end-device needs to communicate in order to join the network. The two cryptographic keys are used for generating session keys (AppSKey, FNwkSIntKey, SNwkSIntKey, and NwkSEncKey).

Gateways receive messages from all end-devices that are within range and relay messages to network servers over either WiFi, Ethernet or LTE. They may also relay messages to other end-devices over LoRa.

Network servers manage the network. They check device addresses, which are different from the DevEUIs, to authenticate end-devices. To verify integrity and decrypt certain packets, the server also has the session keys FNwkSIntKey, SNwkSIntKey and NwkSEncKey. They route application packets to the appropriate application servers and forward join-request and join-accept packets between the appropriate join servers and end-devices.

Join Servers manage the activation procedure, needed for end-devices to join the network. In order to do this they require the DevEUI, and the two root keys - AppKey and NwkKey - of each end-device that wishes to join the network. An end-device sends a join request to this server, which the server can check with the aforementioned parameters. Once accepted, the join server and end-device generate the session keys and the join server sends the required keys to the application server and network server.

Table I
LoRa PHYPayload

Size (bytes)	1	7 ... N	4
PHYPayload	MHDR	MACPayload	MIC

Table II
LoRa MACPayload

Size (bytes)	7 ... 22	0 ... 1	0 ... M
MACPayload	FHDR	FPort	FRMPayload

Table III
LoRa FHDR

Size (bytes)	4	1	2	0... 15
FHDR	DevAddr	FCtrl	FCnt	FOpts

Finally, application servers handle and interpret the data sent by end-devices and can send downlink messages to end-devices. This server also contains a session key (AppSKey), needed to decrypt packets coming from end-devices.

2) LoRa Security: LoRaWAN makes use of symmetric cryptography. The creation of message integrity codes (MIC) and encryption of payloads is achieved by making use of four session keys (FNwkSIntKey, SNwkSIntKey, NwkSEncKey, and AppSKey). The way these session keys are created, depends on the activation procedure. Activation, that is joining a LoRaWAN network, is either possible through Over-The-Air Activation (OTAA) or Activation By Personalization (ABP). In the case of OTAA, the four session keys are derived from the NwkKey and AppKey in combination with the DevEUI and JoinEUI. The way these keys are derived can be seen in Appendix A. For ABP, these session keys are directly stored onto the end-device itself, meaning that NwkKey, AppKey, DevEUI, and JoinEUI are not necessary.

Table I, II, and III show most of the typical fields of a packet sent with LoRa. For encryption both the FRMPayload and FOpts fields are encrypted separately. After encryption, the MIC (which is a message authentication code) is created for each packet using the AES128-CMAC scheme. FOpts and FRMPayload are both encrypted with an encryption scheme based on IEEE 802.15.4/2006 Annex B [3] with an AES key of length 128 bits. The key used for this differs, however. In the case of FOpts it is always encrypted using the NwkSEncKey. The key used for FRMPayload depends on FPort. Either NwkSEncKey is used if FPort has a value of zero or AppSKey is used in all other cases.

One part of the MIC is calculated using the Forwarding Network session integrity key (FNwkSIntKey). Another part is calculated using the Serving Network session integrity key (SNwkSIntKey). Both parts are calculated by taking all fields in the message, i.e. MHDR and MACPayload. This results in a 4 byte MIC that is added to the PHYPayload as seen in Table I.

Encryption is end-to-end between the end-device and application server. Integrity is, however, guaranteed through the MIC in a hop-by-hop fashion. This means that a network server may alter the packet's contents without it being detected.

Figure 1 shows the keys that are present in each device, as well as confidentiality and integrity attributes of certain connections.

Lastly, LoRa is secured against most replay attacks as it makes use of frame counters. This is a 32 bit counter that is stored on the end-device which should not reset after shutting it down. Messages with a frame counter lower than or equal to the frame counter of the last accepted packet are discarded. As the frame counter is taken into account when creating the

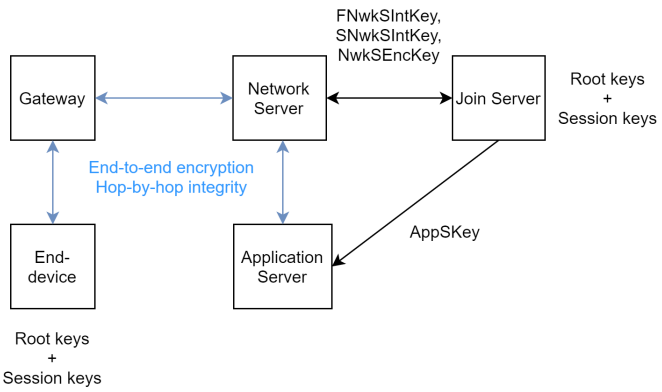

Figure 1. LoRaWAN architecture showing session keys, confidentiality, and integrity attributes

MIC, re-sending a message with an altered frame counter (one that has been incremented by one, for example) is not possible either.

B. LTE

NB-IoT is based on LTE [4]. Therefore, we will first describe the architecture and security measures of LTE, beginning with some of its important components.

1) LTE Architecture: We are interested in the confidentiality and integrity of user data that is sent over LTE. Therefore, in this section, we cover the components of an LTE architecture that have a role in authenticating users, and routing user data [5]. Figure 2 shows how each component is placed in the LTE architecture of a mobile network operator (MNO).

- **User Equipment (UE):** Devices that make use of an LTE network, e.g. mobile phones and IoT sensors, are called User Equipment. The UE contains the Universal Integrated Circuit Card (UICC), which contains cryptographic keys that are also known by the MNO. Furthermore, the UE also stores two identifiers, the International Mobile Equipment Identifier (IMEI) and the International Mobile Subscriber Identity (IMSI). The IMEI identifies a mobile device, while the IMSI identifies the subscriber, who probably is the owner of the mobile device.
- **Evolved Node B (eNB):** In an LTE network, a base station is called an Evolved Node B. The eNB is the access point for the UE. The eNB and the UE communicate through the Uu interface. The X2 interface is used to communicate between two eNBs. An S1 interface (S1-MME or S1-U) is used between eNBs and the core network.
- **The Core Network:** The core network is responsible for authenticating UEs, and routing their user data through the network. Components of the core network, important to this research, are described below.
  - **Mobility Management Entity (MME):** The Mobility Management Entity has an important role in the control plane of an LTE network. As its name suggests, it provides mobility for the UE, but it is also used for user authentication and selecting gateways.
  - **Serving Gateway (S-GW) and PDN Gateway (P-GW):** The user data from the UE is sent via an eNB to a Serving Gateway in the core network. The S-GW forwards the user data to a Packet Data Network (PDN) Gateway. The P-GW forms a gateway to other networks, e.g. the Internet.
  - **Home Subscriber Server (HSS):** The MNO stores information on its subscribers in the Home Subscriber Server. An important entry, which is stored for a subscriber in the HSS, is the cryptographic key K that is also stored within the UUIC in the UE.

2) LTE Security: Because we focus on the confidentiality and integrity of IoT sensor data, we are mostly interested in the security of the LTE user plane. However, to get secure data transport at the user plane, a secure control plane is also important. Therefore, we first briefly discuss how a UE gets access to the user plane, by identification and authentication.

When a UE wants to connect to an LTE network, it first needs to identify itself. The identification is done following the Initial Attach Procedure [5]. In this procedure, the UE sends its IMSI and IMEI to the MME. The result of the procedure is that the UE is provided a Globally Unique Temporary Identity (GUTI). The GUTI is from there on used as identifier instead of the IMSI. The IMSI is used as little as possible, because it can also be used by eavesdroppers to identify subscribers. Next, the Authentication and Key Agreement (AKA) procedure is used to mutually authenticate the UE and the LTE network [5]. To authenticate, the UE and HSS use the shared cryptographic key K. If the authentication succeeds, the UE can access the network and make use of the user plane.

Figure 3 and 4 show the protocol stack of the LTE user plane and control plane respectively. The layers labeled RLC, MAC and PHY provide the ability to transport bytes between the UE and eNB, e.g. error correction, multiplexing, modulation [6]. In contrast to the Packet Data Convergence Protocol (PDCP), these layers do not add anything to the security of LTE. PDCP provides confidentiality and integrity to its upper layers, e.g. RCC, between the UE and eNB on the Uu interface. It supports encryption of user plane data and control plane data, and it supports integrity protection and integrity verification of control plane data [7]. Whether or not encryption and/or integrity protection is used by the PDCP is decided by the upper layers.

In the control plane, Radio Resource Control (RRC) signaling is used between the UE and eNB, and Non-Access Stratum (NAS) signaling is used between the UE and MME. While RRC depends on PDCP for confidentiality and integrity, NAS provides its own confidentiality and integrity. This is the Non-Access Stratum Security (NAS security). The security provided to RRC and the user plane by PDCP is called Access Stratum Security (AS security) [8]. Furthermore, while

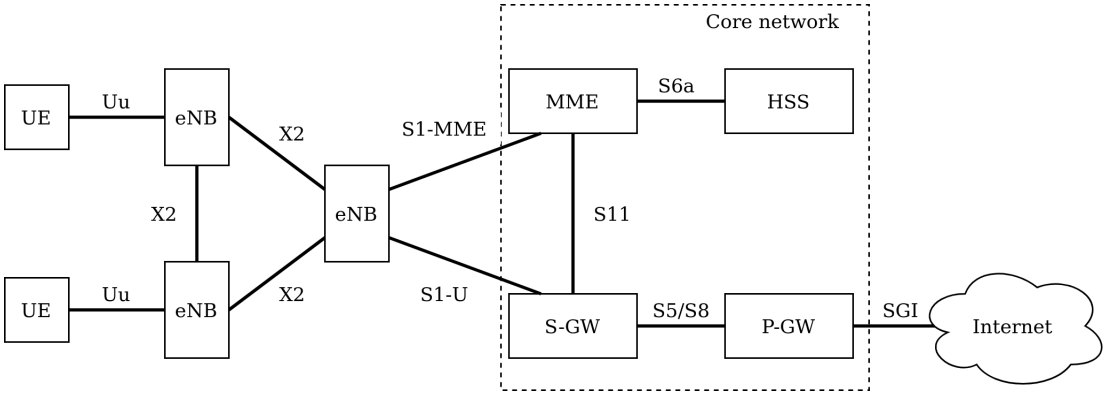

Figure 2. A basic architecture of an LTE network. The names of the interfaces that are used by the components to communicate are also shown.

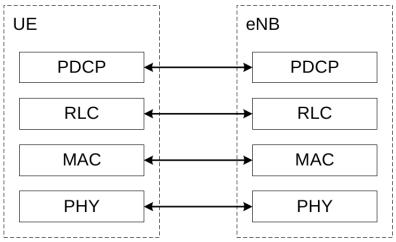

Figure 3. The protocol stack of the LTE user plane [6].

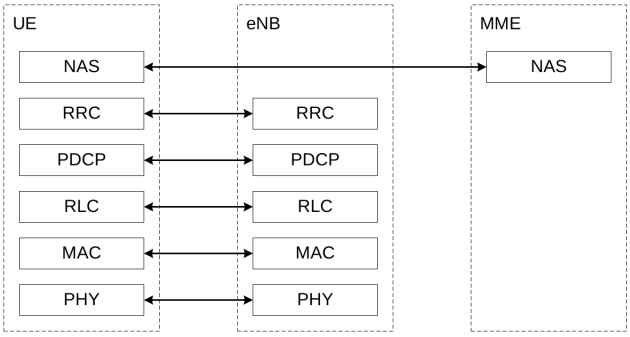

Figure 4. The protocol stack of the LTE control plane [6].

confidentiality is supported, confidentiality for RCC signaling and NAS signaling is optional to the MNO. However, integrity protection is not optional and must be provided for RCC signaling and NAS signaling. The fact that no integrity is provided to the user plane on the Uu interface and confidentiality is recommended, but optional to the MNO, is important to this research. The cryptographic algorithms that are supported by LTE to achieve confidentiality and integrity are based on SNOW 3G, Advanced Encryption Standard (AES) and ZUC [8]. For example, AES CTR mode can be used to achieve confidentiality and AES-CMAC to achieve integrity [5].

In addition to the Uu interface, the eNB also uses the X2 and S1 interfaces to connect to the core network. Following the LTE security specification [8], user plane and control plane data must both have confidentiality and integrity protection on these interfaces. This is achieved by a mandatory IPsec Tunnel mode, with Internet Key Exchange version 2 (IKEv2) for certificate authentication.

Finally, we discuss the security in the core network, beginning with the S6a interface. The S6a interface carries sensitive data from the HSS, e.g. cryptographic keys. Therefore, confidentiality and integrity protection are mandatory at this interface. Furthermore, if components of the core network are not located in the same security domain, e.g. physically divided, the interface between the security domains must provide integrity and optionally confidentiality. Also here, IPsec with IKE authentication is mandatory [8, 9].

For our research, we can conclude that LTE's user plane does support authentication and optionally confidentiality between the UE and core network. However, no integrity is supported for the user data. Furthermore, if the UE is for example an IoT sensor and wants to report its measurements to an application server, e.g. a monitoring platform, then the sensor data needs to go over the Internet and would be unprotected there. This is illustrated in Figure 5. Furthermore, we put our trust in the LTE provider here, to make a part of our path from the sensor to the application server confidential. There is no end-to-end authentication, confidentiality, nor integrity. However, end-to-end security can be implemented on top of LTE in higher layers, e.g. Transport Layer Security (TLS).

C. NB-IoT Architecture

NB-IoT is similar to LTE. However, changes have been made for NB-IoT to make low power consumption possible in the end devices. The UE goes into sleep mode when it is not using sensors nor sending data to the application server. The UE is not able to receive data in sleep mode. Therefore, messages to the UE need to be queued by the MNO. Then, the UE first needs to send an arbitrary uplink message, notifying the MNO that it is awake, before it can receive data. A change in LTE that affects the security of NB-IoT is the Control Plane CIoT EPS optimization [10]. This optimization allows transport of user data over the control plane via the MME to the S-GW, which forwards it to the

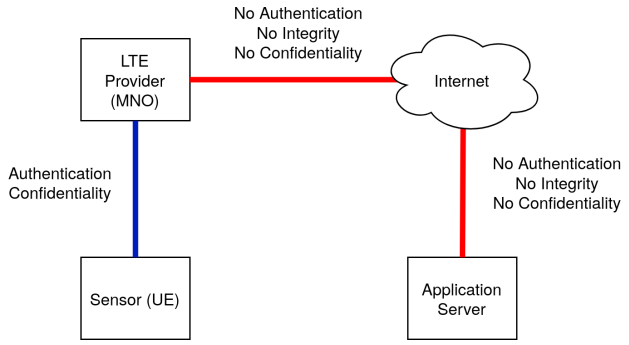

Figure 5. A simplified architecture of an IoT sensor that reports to an application server via LTE. It is labeled at each link if the sensor data has authentication, confidentiality and integrity.

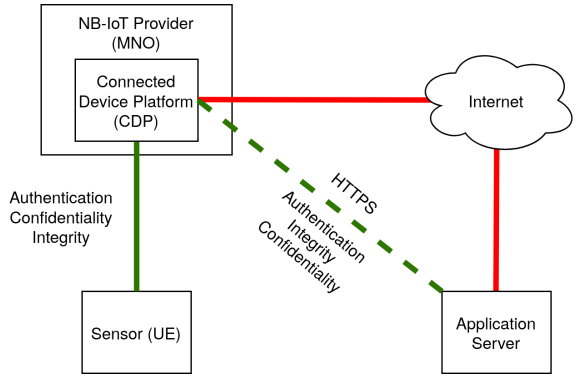

Figure 6. An NB-IoT architecture where the provider hosts a CDP. HTTPS can be used to provide confidentiality, integrity and CDP authentication. By using Data over NAS integrity is added between the UE and MNO.

P-GW. To transport user data between the UE and MME over the control plane, the data is encapsulated in a NAS message (Data over NAS). Transportation of user data over the control plane has a security advantage. NAS security is used instead of AS security. Thus, in addition to confidentiality protection, the user data is now also integrity protected at the Uu interface [11]. With NB-IoT, the UE has two options to send or receive data. Data can be transported as a UDP datagram within an IP packet, or Non-IP Data Delivery (NIDD) can be used [12]. In the case of NIDD, data is transported inside a NAS message without UDP and IP headers.

When UDP is used by the UE, a similar architecture as with LTE is possible (Figure 5). However, another possible architecture is shown in Figure 6. In this architecture the MNO uses a Connected Device Platform (CDP) to store the messages of the UE. Thereafter an API can be used to get the message from the CDP to an application server, e.g. a monitoring platform. The traffic between the CDP and application server can be secured by using an application layer security protocol such as HTTPS. In this case, authentication of the CDP can be done with certificates and the application can be authenticated with user credentials. Thus, with this NB-IoT architecture we have authentication, confidentiality and integrity from the UE to the application server. However, this is not end-to-end. In this case we still put our trust in the NB-IoT provider.

D. End-to-End Security on top of NB-IoT

There are already technologies that allow end-to-end security on top of NB-IoT. In this section we will discuss these technologies.

1) Datagram Transport Layer Security: When the IP packets sent by the UE are also the packets that are received by the application server, i.e. not using NIDD nor a CDP, Datagram Transport Layer Security (DTLS) can be used to create end-to-end encryption [13]. However, it is not standard for the NB-IoT UE to support DTLS. Another disadvantage of DTLS is that it adds the overhead of a handshake.

2) Object Security for Constrained RESTful Environments: Constrained Application Protocol (CoAP) is a protocol that can provide the REST services of HTTP to an NB-IoT network. Within the protocol stack of an NB-IoT network, CoAP is placed on top of UDP or DTLS [14]. Object Security for Constrained RESTful Environments (OSCORE) is a CoAP extension which adds end-to-end security by using Authenticated Encryption with Associated Data (AEAD). OSCORE only encrypts and integrity protects certain fields of the CoAP protocol (protected fields). By using symmetric key cryptography and because only sensitive fields of CoAP are protected, OSCORE adds less overhead than end-to-end security with DTLS [14]. Another advantage over DTLS is that OSCORE makes end-to-end security possible with NIDD. However, just like DTLS, OSCORE and CoAP itself are not included in the NB-IoT standard. Therefore, OSCORE is not mandatorily implemented on NB-IoT UE. However, Lightweight M2M (LwM2M) version 1.1 supports CoAP with OSCORE. LwM2M is used for IoT device management and it has support for NB-IoT and LoRaWAN [13]. Since LwM2M version 1.0 is at the time of writing already implemented by some NB-IoT UE, e.g. UE from u-blox [15], LwM2M 1.1 with OSCORE might be implemented in the near future [16]. Furthermore, OSCORE can only provide end-to-end security if the LwM2M 1.1 traffic is between the UE and application server, not between the UE and MNO which is the case when a CDP is used.

III. Related Work

Kais Mekki et al. [2] compared Sigfox, LoRa and NB-IoT in terms of quality of service (QoS), coverage, range, latency, battery life, scalability, payload length, deployment, and cost. A Sigfox base station has better range (40 km) than LoRa (20 km) and NB-IoT (10 km). Sigfox and LoRa have a lower power consumption and also lower end-device costs than NB-IoT. At the expense of higher power consumption, NB-IoT can operate with low latency and implements QoS. LoRa also has the option to increase the power consumption and lower the latency. NB-IoT scales up to 100 000 end devices per cell, while Sigfox and LoRa scale up to 50 000 end devices per cell. A disadvantage of Sigfox is that it limits the maximum messages per day (140 uplink, 4 downlink). Sigfox also has

the smallest payload length with 12 bytes uplink and 8 bytes downlink. LoRa's payload length is 243 bytes and NB-IoT's is 1600 bytes. An advantage of LoRa is its local network deployment that can be connected to the public network via a LoRa gateway.

Florian Laurentiu Coman et al. [17] have carried out a vulnerability analysis of LPWAN technologies LoRaWAN, Sigfox, and NB-IoT. They discuss the possibility of LoRaWAN packet forging by bruteforcing the Message Integrity Code (MIC). While this does not allow an attacker to send proper messages, as the Application Session Key remains unknown, it does allow them to send gibberish that could lead to Denial of Service (DoS). They note, however, that this requires either an unauthenticated and unencrypted connection between the gateway and the network server, or an attacker having a malicious gateway connected to the victim's network server. Being able to forge a MIC for a packet with the maximum frame counter value, would make it possible for an attacker to perform a DoS attack. This is because packets with frame counters lower than previously accepted packets, are discarded by the network server.

Emekcan Aras et al. [18] have looked into security vulnerabilities of LoRa. They mention possible attacks such as compromising the root keys when an attacker has physical access to an end-device. Jamming by making use of either dedicated hardware or by commercial off-the-shelf LoRa hardware is also possible. Furthermore, if an attacker were to be able to reset a device, this would reset its frame counter, which would subsequently allow them to re-send messages which were sniffed before. Lastly, they mention wormhole attacks where an attacker would sniff a packet and subsequently jam the frequency such that it does not reach the gateway. The attacker could then send (or not) the sniffed packet at any time that they would like. Such a sniffed packet can only be sent once.

Because NB-IoT itself has no measures for end-to-end security, there are no end-to-end vulnerabilities in NB-IoT like LoRaWAN does have, e.g. forging parts of messages such as the MIC. Most vulnerabilities of NB-IoT have to do with DoS attacks as described by Florian Laurentiu Coman et al. [17]. An attacker could for example try to jam NB-IoT traffic between the UE and eNB. Another DoS method would be to drain the battery of UE by pinging it from a compromised UE. However, to make this possible the UE must be able to communicate between themselves, which is not the case in an NB-IoT architecture where UE only reports to a central server. Furthermore, Florian Laurentiu Coman et al. mention the problem of renewing cryptographic keys in an LPWAN end-device as the keys might need to last the lifetime of the UE (10 years), while NIST recommends to renew symmetric keys after 2 years [19].

IV. Methodology

As seen in Section II, LoRa, NB-IoT, and LTE all lack in some aspects when it comes to security. LoRa does not offer end-to-end integrity. NB-IoT and LTE do not have end-to-end authentication, integrity or confidentiality included in their specifications. Standardized technologies exist that are able to add end-to-end security to LTE and NB-IoT, e.g. TLS, OSCORE and DTLS. However, these technologies are not always supported by the UE. In this section, we will describe measures that could be taken in order to mitigate the shortcomings of LoRa and NB-IoT and how we implemented them. We will not consider LTE separately, as this research is focused on IoT technologies. The importance of LTE to this research is that NB-IoT is based on LTE. We will describe the setup of our environments as well.

A. IoT Cryptography Considerations

The use case of this research is an IoT sensor network. The sensors report their measurements to a centralized monitoring platform. Because the sensors and the monitoring platform are managed by the same organisation, it is possible to use symmetric cryptography with a pre-shared key between end-device/UE and application server. By using a pre-shared key, there is no key exchange needed that adds extra round-trips. This is an important consideration in an IoT environment, as extra round-trips lead to higher energy consumption and more time to set up a session between the end-device/UE and application server.

In LoRa we need to add end-to-end integrity and in NB-IoT also confidentiality. To achieve this, we use Advanced Encryption Standard (AES) as symmetric cipher for both LoRa and NB-IoT. We chose AES because it is a widely used and extensively analysed cipher approved by NIST [20].

B. LoRa

In order to set up a LoRa environment we made use of ChirpStack [21] to set up a network server, application server, and gateway bridge. We made use of the Robustel R3000 LG LoRa gateway. For the end-device we made use of a Microchip LoRaBee RN2483A module connected to a SODAQ Autonomo board, programmable via Arduino IDE [22], using the Sodaq_RN2483 library [23]. The LoRa module supports the LoRaWAN v1.0.3 specification. We also connected a TPH v2 sensor to the board to get proper monitoring data. As for the activation procedure, we made use of ABP. We transmitted over LoRa with code rate 4/5 and a 125 kHz bandwidth on EU868 frequencies (868.1MHz, 868.3MHz, and 868.5MHz).

In an effort to provide end-to-end integrity, we create a MIC by using the AppSKey, which is already present on the end-device. This is an 128 bit AES key. As both the end-device and the application server have access to this key, it can be used to achieve end-to-end integrity. To achieve this, we used the same scheme as is used for the hop-by-hop MIC that is already present in LoRa, i.e., AES128-CMAC. This is used as it does not require any random data to be initialised. As the SODAQ Autonomo only has a pseudorandom number generator, we cannot make use of any other schemes that require random data. We used the implementation of AES-CMAC from WolfSSL for this [24]. AES128-CMAC requires

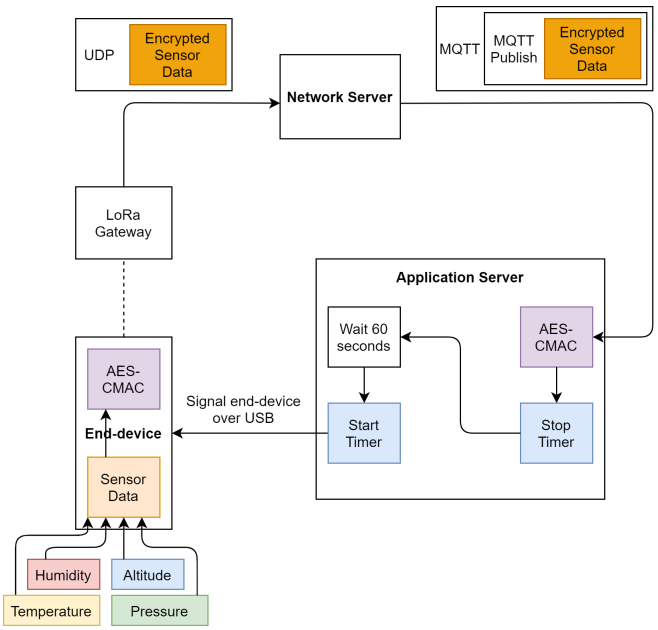

Figure 7. The flow of our LoRa latency measurement.

an AES-128 key, a message, and the message length as input. Different from the LoRa specification where a truncated 4 byte MIC is created, we made use of the full-length 16 byte MIC that is created with AES128-CMAC. This was done to add security, further minimising the possibility of the MIC being forged by brute forcing, as described in Section III. For our implementation, AES128-CMAC takes the message as well as the frame counter as input, alongside the message length and the AES key. The frame counter is taken into account, to provide end-to-end integrity for the frame counter as well (as opposed to only for the message). This makes it infeasible for a rogue network server to manipulate it, subsequently mitigating certain replay attacks. The newly created MIC is ultimately appended to the monitoring data in the payload, adding 16 bytes of overhead.

1) Latency measurement: In order to measure the effect that calculating and adding a MIC to the payload has on the performance, we measured the latency. That is, the execution time it takes for creating a MIC and sending data, and the time it takes for data to be received and verified at the application server. The end-device is connected with a USB to the application server. It waits for the server to instruct it to send a packet. The end-device creates a MIC and sends the packet over LoRa, eventually arriving at the application server, after instructions have been received. The application server sets a timer, timing how long it takes for a packet to arrive and be verified after is has given the instruction. A packet is sent every minute, as sending packets at a higher rate could block the end-device. We also measure the time it takes to create and verify a MIC separately, to see the impact that this has on the execution time. These experiments are carried out 1000 times. Figure 7 shows our setup for these experiments.

C. NB-IoT

In our NB-IoT environment we used an SODAQ NB-IoT Shield board [25] with a u-blox SARA N211 02B-00 NB-IoT module [26], and a temperature and humidity sensor. This board is attached to a Crowduino M0 SD [27], programmable via Arduino IDE [22]. The MNO in our environment is T-Mobile [28]. T-Mobile uses a CDP to give the application server access to the UE messages via HTTP or HTTPS. The application server is authenticated with a username and password. T-Mobile does not support NB-IoT without a CDP [29]. Therefore, this environment is a perfect example of an environment where end-to-end encryption is not possible via DTLS or OSCORE. The DTLS and CoAP traffic would be between the UE and the CDP [30] and not end-to-end between the UE and the application server. The only way to have end-to-end security in an environment like this, is to encrypt the payload itself. Therefore, we have implemented authentication, confidentiality and integrity from the UE to the application server ourselves. In the remainder of this section, we will discuss the choice of the mode of operation used with AES and how we implemented it securely. We end with an experiment in which we measure the effect of the chosen cipher on the latency from the UE to the application server.

1) Mode of operation: AES can be used with different modes of operation, e.g. Galois Counter Mode (GCM) or Cipher Block Chaining (CBC). We chose GCM as mode of operation with two reasons. Firstly, AES-GCM is an Authenticated Encryption with Associated Data (AEAD) cipher [31]. Therefore, it provides both confidentiality and integrity based on a pre-shared key. Secondly, the initialization vector (IV) of AES-GCM is allowed to be predictable. This is important to the NB-IoT environment, as the UE might not be able to generate random data. The Crowduino used in this research is only able to generate pseudorandom data. Therefore, using AES-CBC would not be secure as it needs randomly generated IVs [32].

The input to AES-GCM is an AES key, an IV, additional authenticated data (AAD) and the plain text. The AAD is optional and contains data that does not need to be encrypted, but must be authenticated [33]. For example, version numbers of protocols can be used as AAD, to make sure that the right versions are used [34]. As the AAD is optional, we left it empty. The AES key is a randomly generated bit string of length 128, 192 or 256. For the IV, NIST recommends [35], using 96 bits of which 32 bits are a fixed field and 64 bits are a counter field. The counter field can start at zero for the first message encryption and increment for each new message. While the same key and fixed IV field are used for each message encryption on a device, the counter must never go back to zero, otherwise security will be broken by repeating IVs [33]. An important security requirement is that an IV is never used twice with the same key. Therefore, the counter must never wrap-around, back to zero. With a 64 bit counter field and a send rate of one message per second, the UE is able to last for more than $5.8 \cdot 10^{11}$ years. This is more than

enough for an NB-IoT device, which is built to last without maintenance for 10 years.

2) Implementation: We implemented end-to-end authentication, confidentiality and integrity between the UE and application server. We used the implementation of AES-GCM from WolfSSL [24] to add the needed cryptographic functionalities to the Crowduino M0 SD. As application server, we used the Python3 HTTP Server library [36]. At the application server we used the Python3 Cryptography library [37] to add support for AES-GCM. As long as each UE has its own pre-shared key with the application server, the messages between the UE and application server can also be authenticated. One could also choose to use the same key for each UE. In this case, the fixed field of each UE must be different. The disadvantage of this solution is limited authentication as the application server can only verify that a particular message came from one of the UE, but not from which one. Also, if one UE gets compromised, each UE needs to be updated with a new key. For each UE we used a different AES key of 128 bits long. Furthermore, an IV must never be used twice with the same key. Therefore, if a key is used for encryption by both the UE and the application server, the fixed field of the IV must be different for the UE and the application server. The output of AES-GCM is a cipher text and a tag of 16 bytes. In addition to the cipher text, the receiving end needs the tag and IV as well to decrypt and check the integrity of the message. Therefore, we append the tag and IV to the cipher text and send them all together over NB-IoT. Thus, 28 bytes of overhead are added to the message by using AES-GCM.

To add replay protection, the receiving end of a message must compare the counter field in the IV with the counter field of the previous message received from the same sender. The new message must contain a counter greater than the counter of the previous message. To protect against a reflection attack, the UE and application server could both use a different key for encryption, or the following two approaches with the fixed IV field could be chosen.

Approach 1: the UE and application server could check that their received messages do not have a fixed IV field that is used by themselves. We used this approach in our implementation.

Approach 2: the the fixed IV field of the UE and the application server could be made known to each other on forehand. With this approach, messages that were not ciphered with the expected fixed IV field can be rejected.

3) Latency measurement: To measure how AES-GCM affects the latency of messages from the UE to the application server, we set up an experiment as shown in Figure 8. The UE is connected to the application server via USB. Each measurement starts with the application server starting a timer. Directly after setting the timer, the application server sends a signal to the UE over the USB connection that it must send a message. When the UE receives the signal, it has its sensor data already available for sending. It must only use the AES-GCM cipher on the sensor data and send it via T-Mobile NB-IoT to the application server. The size of the sensor data is 8 bytes, containing two floating points that represent

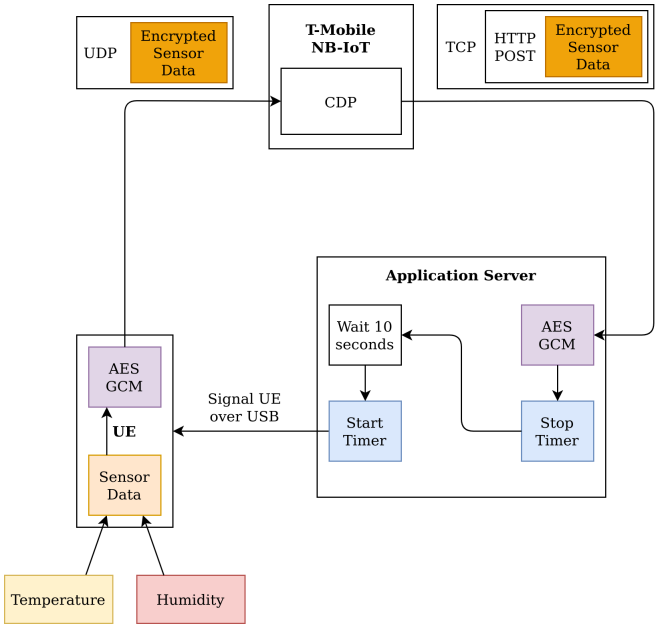

Figure 8. The flow of our NB-IoT latency measurement.

temperature and humidity. Once the application server has decrypted the sensor data and checked the integrity, it stops the timer. After waiting ten seconds, a new measurement will be started. We repeated this measurement 10000 times. Between each measurement was a ten seconds interval, to send messages at a rate that is suitable for NB-IoT. To compare, this experiment will also be performed without AES-GCM. Furthermore, we also separately measure the time it takes to encrypt, decrypt and check the integrity of the 8 byte messages with AES-GCM.

V. Results

In this section we show the results of the LoRa and NB-IoT measurements. For both technologies we measured how much time it takes to execute our added cryptographic functionalities. Furthermore, we measured the latency of the technologies from the end-device/UE to the application server.

A. LoRa

Figure 9 shows the time it takes for the MIC to be created by the end-device and the time it takes for the MIC to be verified by the application server. The mean time it takes for the MIC to be created is 418 μs. Verifying the MIC takes 750 μs on average.

In Figure 10 and 11 we see the result of our latency experiment for LoRa. We see two histograms of the 1000 latency samples both with and without AES-CMAC in Figure 10. In Figure 11 we observe the latency and 99-percentile for both these cases. On average without AES-CMAC there is a latency of 1884ms while with AES-CMAC there is an average latency of 2132ms. The 99-percentile is at 2597ms without AES-CMAC and 3139ms with AES-CMAC.

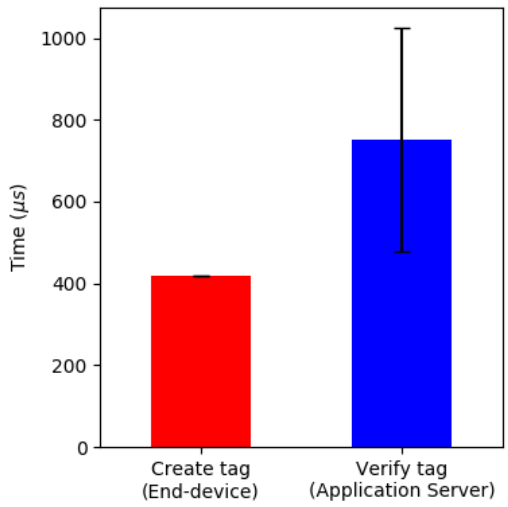

Figure 9. The time taken by the end-device and application server to perform AES-CMAC on one message.

B. NB-IoT

Figure 12 shows the time taken by the UE and application server, to use AES-GCM for encryption and decryption. It takes the Crowduino on average 935 μs to encrypt an 8 byte message with AES-GCM, using an 128 bit key. The application server needs on average 453 μs to decrypt the message and check its integrity.

Figure 13 shows two histograms of the 10000 latency samples, one with and one without AES-GCM. It is notable that both histograms contain spikes that are separated by one second. The highest spike is around 2000 ms. Figure 14 shows the mean latency and the 99-percentile for both cases. The mean latency is also around 2000 ms, with and without AES-GCM respectively 1984 and 1982 ms. The 99-percentile in both cases is 5007 ms.

VI. Discussion

A. LoRa

The way we implemented end-to-end integrity for LoRa, was to append an additional MIC to the payload. One downside to this is that this constitutes as MAC-then-encrypt, i.e. a MIC of the plaintext is created and then encrypted alongside the plaintext. An issue with this, is the fact that the application server is unable to verify the integrity of the message without decrypting it first. This means that when an attacker sends garbage to an application server, it would have to perform an extra step, requiring more time to discard the message. This, in turn, may facilitate other attacks such as DoS attacks. Ideally, encrypt-then-MAC should be used instead [38].

An issue with LoRa's implementation of integrity checks is the fact that it makes use of a MIC that is only 4 bytes long. This goes against the NIST recommendation of having a MAC that is at least 64 bits (8 bytes) long, as described in NIST SP 800-38B [39]. Making use of a MIC that is too short results in vulnerabilities such as MIC forging as described in Section III. This is why the MIC, which is used for end-to-end integrity, created in this research has a length of 16 bytes. While this results in more overhead, it also offers a higher degree of security, which may be needed when handling sensitive data. Making use of a 16 byte MIC makes it infeasible for a MIC to be forged.

Furthermore, we have incorporated the frame counter in our newly created MIC. Replaying the message after the default 4 byte MIC has been forged is thus not possible as the frame counter is also used for the 16 byte MIC. Incorporating the frame counter does not prevent all replay attacks, however. It is, for example, still the case that - after the frame counter is reset, either by resetting the device or by means of a wrap-around - an attacker would be able to replay old messages from before the frame counter was reset.

From Section V we can see that the execution time does not increase significantly, when adding end-to-end integrity through AES-CMAC. We do, however, see that the latency increases significantly, from 1884ms to 2132ms. This is likely because the extra 16 byte tag is included in the payload. More data has to be sent, increasing the latency.

One thing to note is that the implementation of end-to-end integrity described in this research makes use of the AppSKey. Ideally, a separate AES key is used for creating a MIC as AppSKey is already used for end-to-end encryption. If two keys are used this further improves the security, as the compromise of one key only leads to either encryption or integrity failing and not both.

B. NB-IoT

In this research we discussed different ways to make NB-IoT communication between UE and an application server end-to-end secure. We discussed DTLS and OSCORE, but these two possibilities are not supported on all UE and also not supported by every MNO. Therefore, we looked into a third possibility, implementing security at the application layer, using Arduino hardware. In this section we discuss the limitation of our implementation and the results of our experiments.

On a UE reboot it is important that the state of the counter in the AES-GCM IVs is remembered. Otherwise, the UE will start to reuse IVs, breaking the AES-GCM security, and it will accept replayed messages from the application server. To remember the state of the counter, the UE needs persistent storage, which might not be present on the UE. This would form a problem to our method of using AES-GCM in NB-IoT. In this research we assumed that the UE would never reboot. However, this could happen. For example, sensors that are plugged into a power socket instead of a battery will reboot after a power outage. If the counter can't be restored after reboot, alternatively a new key or fixed IV field can be chosen to avoid IV repetition. However, this would be time consuming if it has to be done manually to a device in the field. To solve this problem, a mechanism can be developed with which the application server delivers the fixed IV field to the UE after

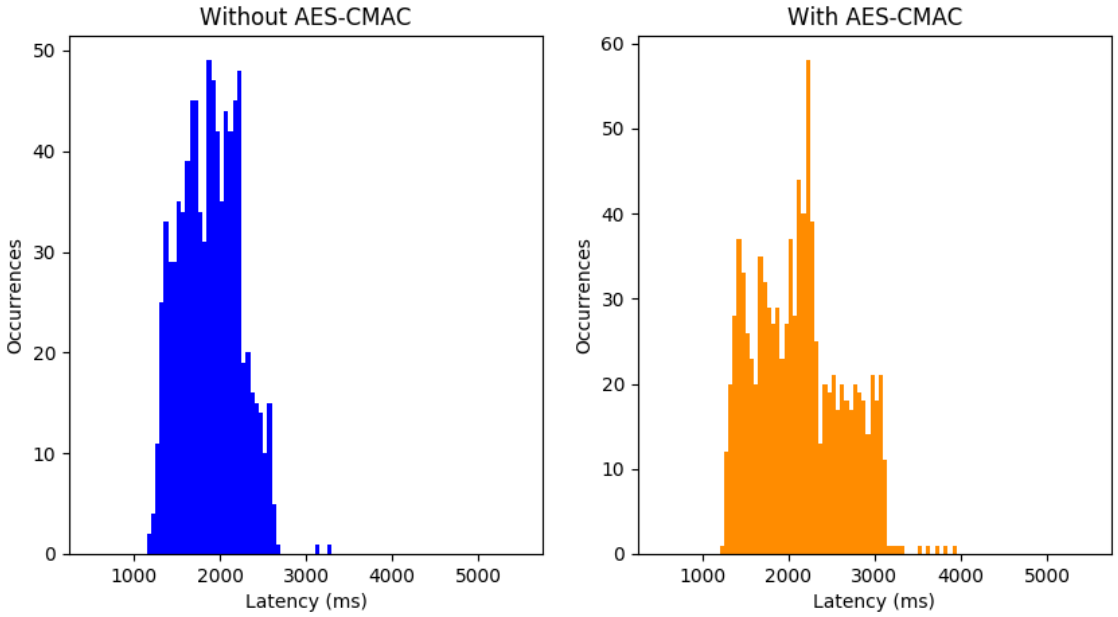

Figure 10. Two histograms showing the latency from the end-device to the application server, with and without AES-CMAC. 1000 samples are shown each.

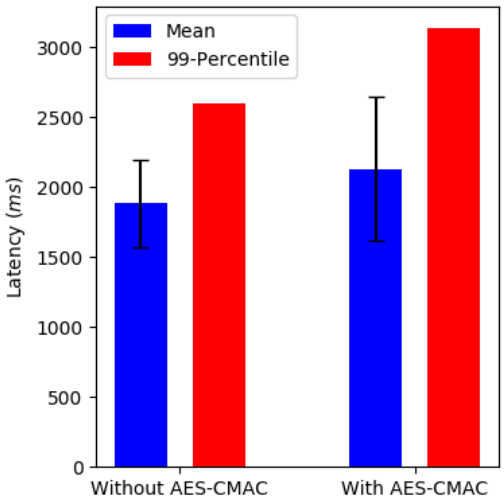

Figure 11. The mean, standard deviation and 99-percentile of the latency between the end-device and the application server.

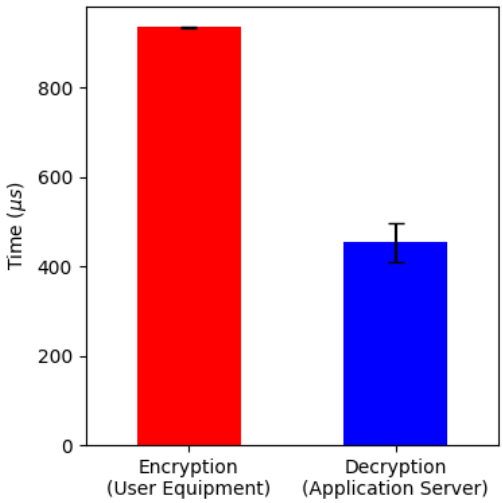

Figure 12. The time taken by the UE and application server to use AES-GCM on one message.

it boots. In this case, the application server could make sure that it chooses a fixed IV field that has not been used before. However, we could not come up with such a mechanism that is protected against replay attacks, as the UE can also not remember the last counter IV field of the application server on a reboot. This would allow an attacker to replay a message with an old fixed IV field. If the UE uses the old fixed IV field with a counter that has been reset, the security of AES-GCM is broken.

Another security issue with AES-GCM is that the length of the cipher text is the same as the length of the plain text. This could give away valuable information to an eavesdropper. For example, one could differentiate between the cipher text of the plain text “Yes” and the plain text “No”, just by looking at its length. This problem can be solved by having a constant message size. Padding can be used to fill smaller messages.

Section V-B has shown that the time it takes to use AES-GCM at the UE and application server is in the order of microseconds. The latency from the UE to the application server was in the order of seconds, with and without AES-

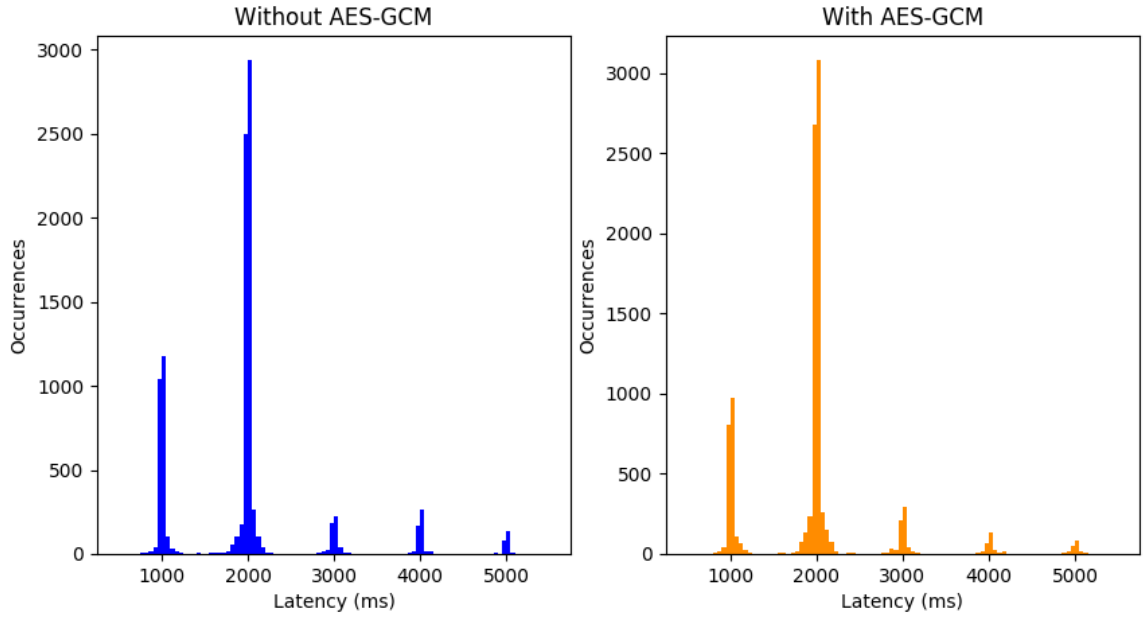

Figure 13. Two histograms of the latency from the UE to the application server, with and without AES-GCM. The histograms contain 10000 samples each.

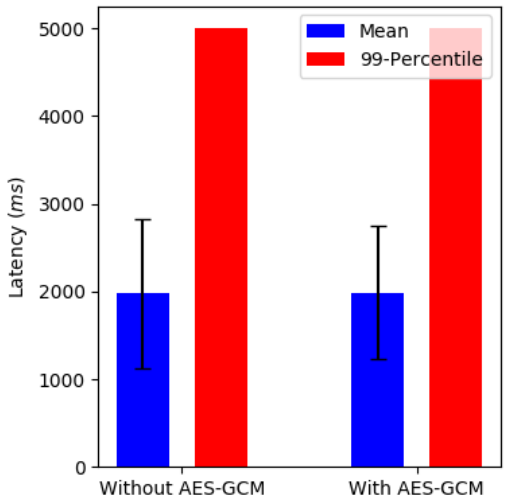

Figure 14. The mean, standard deviation and 99-percentile of the latency between the UE and application server.

GCM. Because, the microseconds encryption/decryption time are almost nothing compared to a few seconds of latency, we can say that AES-GCM does not significantly influence the latency of NB-IoT. However, it could still influence the power consumption of the UE. This is something that we did not research, but is also important to an IoT device. AES-GCM does not only add extra operations per message which increases the power consumption, but it also adds 28 bytes to the message that need power to be transmitted. Power consumption could be lowered by choosing other cryptographic functions than AES-GCM. Instead of AES-GCM, one could use lightweight cryptography methods such as PRESENT for encryption, and SPONGENT as a hash function to provide integrity. These are more efficient algorithms, but are also less secure [40].

Lastly, we also noticed that the latency histograms in Figure 13 contained spikes, separated from each other by one second. We expect that this is caused by the implementation of the CDP of T-mobile. The CDP buffers messages from the UE and does an HTTP post to push the messages to the application server. If the CDP does its posts to our application server on a one second interval, it could be the cause of the noticed effect on the latency.

VII. Conclusion

In this research we discussed the security measures that are in place for LoRa. We have seen that LoRa supports end-to-end confidentiality and hop-by-hop integrity, by default. We have also discussed several attacks and have seen from related work that MIC forging and replay attacks are possible. Hop-by-hop integrity also makes it possible for a rogue network server to alter packets without notice of the end-device or application server.

For LoRa we implemented end-to-end integrity on the payload. This was done by creating a tag with AES-CMAC and adding it to the payload, using a shared key that both the end-device and application server have. AES-CMAC was chosen as it did not require any random data generation to be initialised. This makes it suitable for IoT devices, as there is a high probability that such devices are unable to generate random data. We included the frame counter as input for AES-CMAC in order to mitigate certain replay attacks. In order to

be protected against replay attacks, one needs to make sure that the frame counter is not reset. We have measured the execution time and latency with and without the use of AES-CMAC. From our experiments we had seen that making use of AES-CMAC does increase the latency significantly as more data needs to be sent.

For NB-IoT we discussed DTLS, OSCORE and an application layer implementation to add end-to-end security between UE and the application server. DTLS is not supported by all UE, as it is not a standard for NB-IoT. However, some NB-IoT UE manufactures do implement it. When the UE does support DTLS, also the right MNO must be chosen. One that allows the UE to send its UDP datagrams to the application server. Thus, an MNO without a CDP must be used.

OSCORE extends CoAP with end-to-end security using AEAD. As OSCORE is also part of the LwM2M 1.1, it probable that it will be implemented by manufacturers of NB-IoT UE in the near future. However, at the time of writing, we could only find UE with support for LwM2M 1.0. Thus, without OSCORE. Just like DTLS, OSCORE does not provide end-to-end security if a CDP is used by the MNO, as the CDP will be one of the CoAP endpoints.

If DTLS and OSCORE are not supported and the UE is programmable, we can add end-to-end security on the payload ourselves. We did this by encrypting the NB-IoT payload with AES-GCM, using a pre-shared key only known to the application server and one UE. AES-GCM was chosen, because it allows predictable IVs. Therefore, the UE does not need to have a secure random number generator. To keep our method with AES-GCM secure, we discussed some requirements. An IV must never be used twice with the same key. Therefore, the UE should be able to remember the state of the counter field in the IV when it reboots. To protect against replay attacks, it must be confirmed that a received message has a counter field in the IV that is higher than the previous received message. Lastly, to protect against reflection attacks, the UE and application server should use different keys for encryption and decryption, or the fixed IV field can be used to decide if a message was reflected.

Furthermore, due to NB-IoT's power efficient nature, it has a high latency. Our experiments show that using AES-GCM on the messages from the UE to the application server adds an insignificant amount of time to the latency. However, we did not research how the power consumption is affected.

VIII. Future Work

In this research we implemented end-to-end integrity for LoRa and NB-IoT, as well as end-to-end encryption for NB-IoT. While this does mitigate some attacks, other attacks such as wormhole attacks or certain replay attacks are still possible. One could look at the possibility of mitigating such attacks by, for example, introducing time and allowing packets to only be accepted within a certain time frame.

Furthermore, we only implemented security solutions for LoRa and NB-IoT. There are, however, more technologies that are commonly used in IoT environments, such as Sigfox or LTE-M, that could be researched.

For NB-IoT, one could further research use cases in which the NB-IoT UE supports DTLS or other methods that add end-to-end security. As the interest in secure IoT is growing, most new NB-IoT UE support DTLS.

One could also look at the power consumption. Implementing end-to-end confidentiality and integrity with AES-CMAC and AES-GCM could have impact on the power consumption of a device. As most IoT devices are low power devices, it is important that adding confidentiality and integrity this way does not have significant impact on the power consumption. Finally, the effect of DTLS on the power consumption would be interesting to research.

Acknowledgments

We would like to thank our supervisors Cedric Both and Jeroen de Boer from Datadigest, for their time and guidance throughout this work. We enjoyed our time working at Datadigest and are grateful that we have been able carry out our research there.

References

[1] KPN. *KPN completeert aanbod IoT-netwerken met landelijke dekking LTE-M*. https://www.overons.kpn/nl/nieuws/2018/kpn-completeert-aanbod-iot-netwerken-met-landelijke-dekking-lte-m. Accessed: 04-06-2020. 2008.

[2] Kais Mekki et al. "A comparative study of LPWAN technologies for large-scale IoT deployment". In: *ICT express* 5.1 (2019), pp. 1–7.

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

[4] Yihenew Dagne Beyene et al. "NB-IoT technology overview and experience from cloud-RAN implementation". In: *IEEE wireless communications* 24.3 (2017), pp. 26–32.

[5] Jeffrey Cichonski, Joshua Franklin, and Michael Bartock. *Guide to LTE security*. Tech. rep. National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.SP.800-187.

[6] 3GPP. *Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2*. Technical Specification (TS) 36.300. Version V16.1.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2430.

[7] 3GPP. *Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification*. Technical Specification (TS) 36.323. Version V16.0.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2439.

[8] 3GPP. *3GPP System Architecture Evolution (SAE); Security architecture*. Technical Specification (TS) 33.401. Version V16.2.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2296.

[9] 3GPP. *3G security; Network Domain Security (NDS); IP network layer security*. Technical Specification (TS) 33.210. Version V16.3.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2279.

[10] 3GPP. *Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3*. Technical Specification (TS) 24.301. Version V16.4.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072.

[11] 3GPP. *General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access*. Technical Specification (TS) 23.401. Version V16.6.0. 3rd Generation Partnership Project (3GPP), March 2020. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=849.

[12] Cisco. *Small Data over NAS, S11-U and SGi Interfaces*. Visited on: 05-06-2020. 2020. URL: https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-3_N5-5/Ultra_IoT_CSGN/21-3-Ultra-IoT-CSGN-Guide/21-3-Ultra-IoT-CSGN-Guide_chapter_0111.pdf.

[13] Sergey Slovetskiy, Poornima Magadevan, Yun Zhang, Sandeep Akhouri. *Lightweight M2M 1.1: Managing Non-IP Devices in Cellular IoT Networks*. Visited on: 24-06-2020. T-Mobile, Ericsson. 2018. URL: https://www.omaspecworks.org/wp-content/uploads/2018/10/Whitepaper-11.1.18.pdf.

[14] Francesca Palombini, Göran Selander, John Preuß-Mattsson. *OSCORE: A look at the new IoT security protocol*. Visited on: 24-06-2020. Ericsson. 2020. URL: https://www.ericsson.com/en/blog/2019/11/oscore-iot-security-protocol.

[15] u-blox. *SARA-N2/N3 series*. Visited on: 23-06-2020. 2020. URL: https://www.u-blox.com/sites/default/files/SARA-N2-N3_SysIntegrManual_%5C%28UBX-17005143%5C%29.pdf.

[16] Mats Andersson, Peter Karlsson, Hari Vigneswaran. *Trialing OSCORE for end-to-end IoT security in resource constrained devices*. Visited on: 24-06-2020. u-blox. February 2020. URL: https://www.u-blox.com/en/blogs/tech/trialing-oscore-end-end-iot-security-resource-constrained-devices.

[17] Florian Laurentiu Coman et al. "Security issues in internet of things: Vulnerability analysis of LoRaWAN, sigfox and NB-IoT". In: *2019 Global IoT Summit (GIoTS)*. IEEE. 2019, pp. 1–6.

[18] Emekcan Aras et al. "Exploring the security vulnerabilities of LoRa". In: *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*. IEEE. 2017, pp. 1–6.

[19] Elaine Barker et al. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.

[20] NIST. *ADVANCED ENCRYPTION STANDARD (AES)*. Visited on: 29-06-2020. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[21] ChirpStack. *ChirpStack, open-source LoRaWAN Network Server Stack*. Visited on: 24-06-2020. 2020. URL: https://www.chirpstack.io/.

[22] Arduino. *Download the Arduino IDE*. Visited on: 25-06-2020. 2020. URL: https://www.arduino.cc/en/main/software.

[23] SODAQ. *Sodaq_RN2483*. Visited on: 26-06-2020. 2019. URL: https://github.com/SodaqMoja/Sodaq_RN2483.

[24] WolfSSL. *WolfSSL - Home*. Visited on: 25-06-2020. 2020. URL: https://www.wolfssl.com/.

[25] SODAQ. *SODAQ Support pages - NB-IoT shield*. Visited on: 25-06-2020. 2020. URL: https://support.sodaq.com/Shields_and_Bees/NB-IoT%5C%20Shield/.

[26] u-blox. *SARA-N2 series*. Visited on: 25-06-2020. 2020. URL: https://www.u-blox.com/en/product/sara-n2-series.

[27] SODAQ. *SODAQ Shop - Crowduino M0*. Visited on: 25-06-2020. 2020. URL: https://shop.sodaq.com/crowduino-m0.html.

[28] T-Mobile. *IoT Documentation*. Visited on: 25-06-2020. 2020. URL: https://docs.iot.t-mobile.nl/docs/t-mobile-iot-documentation.

[29] SODAQ. *SODAQ Support Pages - Passthrough*. Visited on: 23-06-2020. 2020. URL: https://learn.sodaq.com/Boards/Sara_AFF/Examples/passthrough.

[30] T-Mobile. *NON IP Data delivery (NIDD)*. Visited on: 25-06-2020. 2019. URL: https://docs.iot.t-mobile.nl/docs/non-ip-data-delivery-nidd.

[31] Joseph Salowey, Abhijit Choudhury, and David McGrew. "AES Galois Counter Mode (GCM) cipher suites for TLS". In: *Request for Comments* 5288 (2008).

[32] Sheila Frankel, R Glenn, and S Kelly. "The AES-CBC cipher algorithm and its use with IPsec". In: (2003).

[33] David McGrew. *An interface and algorithms for authenticated encryption*. Tech. rep. RFC 5116, January, 2008.

[34] Tim Dierks and Eric Rescorla. *The transport layer security (TLS) protocol version 1.2*. Tech. rep. RFC 5246, August, 2008.

[35] Morris J Dworkin. *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac*. National Institute of Standards & Technology, 2007.

[36] Python3 Documentation. *http.server - HTTP servers*. Visited on: 25-06-2020. 2020. URL: https://docs.python.org/3/library/http.server.html.

[37] Python3 Documentation. *Cryptography - Authenticated encryption*. Visited on: 25-06-2020. 2019. URL: https://cryptography.io/en/latest/hazmat/primitives/aead/#cryptography.hazmat.primitives.ciphers.aead.AESGCM.

[38] Mihir Bellare and Chanathip Namprempre. "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2000, pp. 531–545.

[39] SP NIST. "800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication". In: *NIST Special Publication* (2005).

[40] William J Buchanan, Shancang Li, and Rameez Asif. "Lightweight cryptography methods". In: *Journal of Cyber Security Technology* 1.3-4 (2017), pp. 187–201.

[41] LoRa Alliance. "LoRaWAN™ 1.1 Specification". In: *LoRa Alliance* 11 (2017), pp. 2018–04.

APPENDIX A
LORA KEYS

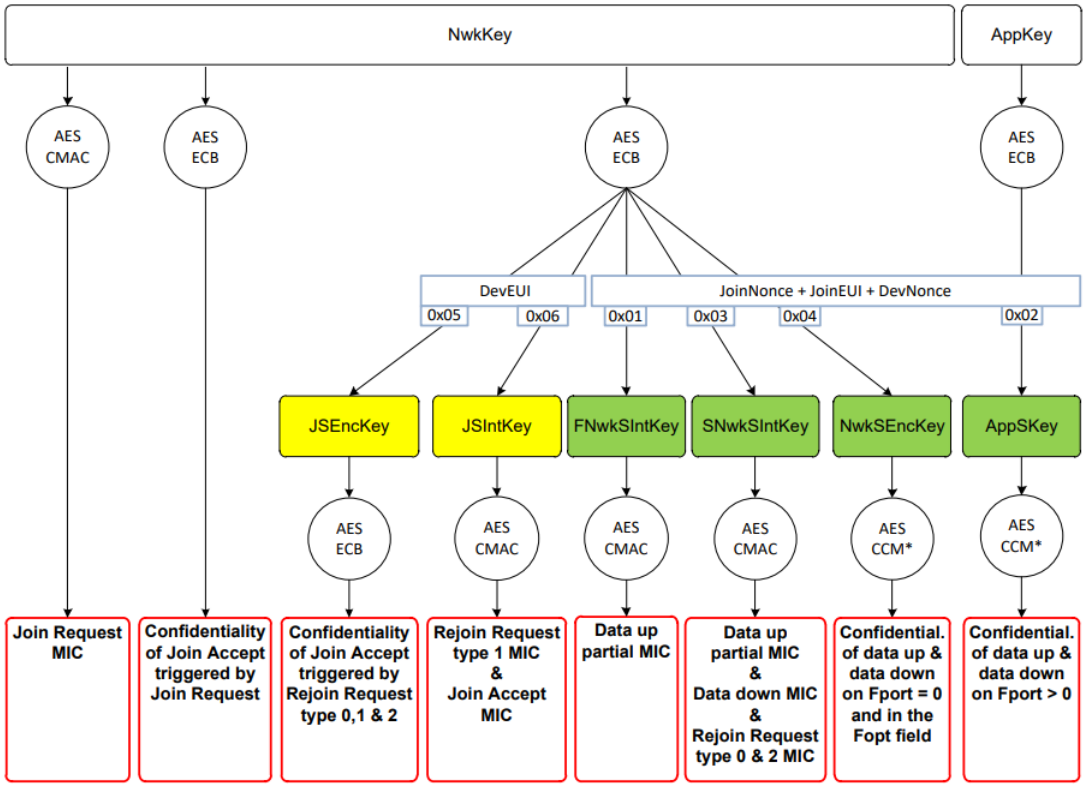

Figure 15. The way various keys, used for creating MICs and encrypting payloads, are derived in the case of OTAA [41].