Enhancing the Security of Over-The-Air-Activation of LoRaWAN Using a Hybrid Cryptosystem

Kevin Feichtinger[†], Yuto Nakano[††], Kazuhide Fukushima[††] and Shinsaku Kiyomoto[††]

KDDI Research, Inc.
2-1-15 Ohara, Fujimino-shi, Saitama, 356-8502 JAPAN

Summary
The Internet of Things (IoT) pursues the objective to connect daily used devices to the web and support humans in their everyday life. Most IoT devices like sensors and cameras are optimized for low power consumption and sometimes spread over many kilometers, which makes different communication protocols necessary than WiFi or ZigBee. Low Power Wide Area (LPWA) networks organize low-power devices in energy efficient networks while allowing these devices to be spread and move over an enormous geographical area. One of the established communication protocols for long-range communication in LPWANs is LoRaWAN, which is optimized for battery-powered end devices. In this paper, the LoRaWAN join procedures are investigated, and possible threats to joining devices are shown. This paper proposes an extension of the Over-The-Air-Activation join procedure of LoRaWAN to enhance the security of the unencrypted join-request, using a hybrid cryptosystem. Furthermore, this paper considers advantages and disadvantages of possible hardware or software implementations and investigates the current specification of LoRaWAN and the payload size differences of LoRaWAN regional parameters to ensure the feasibility of the proposed extension.

Key words:
Internet of Things (IoT), Low Power Wide Area Networks (LPWAN), LoraWAN, Over-The-Air-Activation (OTAA), Hybrid Cryptography

1. Introduction

The Internet of Things (IoT) describes the connection of everyday objects to the web to realize home and industrial automation or simplifying monitoring, health care and energy supply [1]. Engaged devices such as sensors, cameras and microphones collect and exchange personal data for supporting humans in their daily life, which makes secure communication the most valuable asset in IoT [1,2]. Most of these devices are optimized for low power consumption but spread over an enormous geographical area, which makes other communication protocols with different topologies than common IoT protocols like WiFi or ZigBee necessary. One technology which offers secure communication protocols and organizes wide spread devices in energy efficient networks is Low Power Wide Area (LPWA) [2-4]. For LPWA multiple communication protocols exist, which each is optimized for its applications

[3,4]. A proven LPWA long range communication protocol is LoRaWAN [5], which is optimized for battery powered end devices and organizes participating devices in a star-of-stars topology. LoRaWAN is based on the physical LoRa protocol, which allows end devices to be distributed over many kilometers [2-4].

LoRaWAN is an open specified communication protocol, which operates in different ISM spectra. Depending on the deployed area, LoRaWAN uses a different band, which changes the available data rates for transmission [6,7]. The protocol supports a symmetric cryptosystem to secure the communication after a successful join. LoRaWAN end devices have to join the network either via Over-The-Air-Activation (OTAA) or Activation-By-Personalization (ABP) before they can send data through the network. During the join process of OTAA, the data is unencrypted, which exposes the join-request to malicious nodes. A joining end device sends a join-request to the LoRa network, such that the network can validate if the device is authorized to participate in the network. An adversary might eavesdrop and exploit the information in the request to perform certain attacks to the LoRaWAN network. A hybrid cryptosystem can solve this problem.

A hybrid cryptosystem [8] provides that the advantages of an asymmetric cryptosystem and a symmetric cryptosystem be linked together. An asynchronous cryptosystem is much slower than a synchronous cryptosystem but therefore offers the best security because it uses two different keys for encryption and decryption. Most practical hybrid cryptosystems are used to distribute session keys and encrypt data with the symmetric cryptosystem. Usually, a hybrid cryptosystem includes the following steps:

1) The sender obtains the public key of the receiver.

2) The sender generates a symmetric key for encrypting future messages.

Manuscript received February 5, 2018
Manuscript revised February 20, 2018

3) The sender encrypts the symmetric key with the public key of the receiver.

4) The sender transfers the encrypted symmetric key to the receiver.

5) The receiver decrypts the symmetric key with its private key.

6) The sender and the receiver encrypt further messages with the now shared symmetric key.

7) Encrypting the symmetric key by the receiver's public key solves the key management problem of storing the symmetric key somewhere on the device. By using a hybrid cryptosystem for key distribution, the initiator generates a new symmetric key (session key) whenever a new communication session begins and destroys it when the connection terminates [8]. In some implementations, the initiator also includes a data message in the first package to start the communication with the first message. The initiator encrypts the data by the generated session key.

8) This paper proposes an enhancement to the OTAA join procedure of LoRaWAN. The proposed handshake uses a hybrid cryptosystem to encrypt the join-request of the OTAA process to counteract known threats to the LoRaWAN network.

9) The rest of the paper is organized as follows. Section 2 introduces the two join procedures of LoRaWAN and presents possible threats to the proceedings. Section 3 introduces an extended OTAA handshake using a hybrid cryptosystem to encrypt the join-request. Section 4 discusses the feasibility of the new handshake considering the current specification of LoRaWAN and its regional parameters and differences between hardware and software implementations. The proposed handshake and its implementation requirements are evaluated in section 5. Finally, section 6 concludes the paper and outlines future work.

2. Join procedures in LoRaWAN

All components of a LoRaWAN [5] network send data via a gateway to the receiver. A gateway has only simple tasks. It mainly passes the information through to the network server or the end device. The network server is the intelligent entity in the network, which validates all join-requests and sends the personalized information to the end device if it is allowed to participate in the network [9,10]. It also gets all the data sent by the end devices and application servers and forwards them to the correct application server or end device. A symmetric encryption scheme secures the communication between the network server and the end devices after the devices successfully joined the network.

End devices have to join the network by personalization and activation. Every end device has to repeat this personalization and activation process when it loses its personalized information, it is reset, or it confirms its presence in the network, which each end device does at least once a day [11]. During the join process, an end device gets equipped with a device address (DevAddr), a network session key (NwkSKey) and an application session key (AppSKey). The symmetric cryptosystem uses this two session keys to encrypt messages, which the device and the network server exchange and to compute the message integrity code (MIC) [12]. The network server uses the DevAddr to identify all packages of the same end device. For joining the network, two different join procedures are available. Over-The-Air-Activation [5] (OTAA) and Activation-By-Personalization [5] (ABP).

2.1 Activation-By-Personalization (ABP)

If a device can join a specific network via ABP, then it is already equipped with the DevAddr and the session keys when started. During ABP the two steps of personalization and activation are done in one step, and the device can start sending data from the beginning, bypassing the necessity of a handshake with the network server [13]. When the keys of an ABP device get exposed to the public network, an attacker can decrypt all the messages of this end device for the lifetime of the device. Every exposed session key pair may also compromise the communication of other ABP end devices in the network. The challenge for manufacturers is to derive a unique set of session keys and a unique DevAddr for each produced end device to avoid compromising the communication of other end devices if any key or the DevAddr gets exposed [9].

2.2 Over-The-Air-Activation (OTAA)

If a device does not have the required information on startup, it has to initiate a handshake with the network server to obtain its DevAddr and the network identifier (NetID) from the LoRa network. With the NetID the device can compute the NwkSKey and an AppSKey to start a secure communication with the network. The handshake consists of a join-request and a join-accept message. An end device sends a join-request message via the gateway to the network server, which then validates the request. If the end device has the permission to participate in the LoRaWAN, the server responds with a join-accept message.

The join-request message consists of an application identifier (AppEUI), a device identifier (DevEUI) and a random nonce (DevNonce). Each end device gets shipped with a DevEUI and an AppEUI, which are eight-byte identifiers in IEEE EUI64 address space to uniquely identify the device and the application entity. The DevEUI is used to assign each join-request to the correct end device. The AppEUI is used to determine the application service which processes the join-request frame, to verify if the device is authorized to join the network. The DevNonce is a two-byte random value generated by the end device before initiating the OTAA protocol. Together with the DevEUI,

size (bytes)	8	8	2
join-request	AppEUI	DevEUI	DevNonce

Fig. 1 The structure of the join-request in OTAA [5]

the DevNonce value identifies each join-request of the end device at the network server. The network server keeps track of the sent DevNonce for each device to find and reject duplicated join-requests to counteract replay attacks. Figure 1 shows the 18-byte join-request structure.

Before the device sends the request to the network server, it signs the join-request with an AES-128 key (AppKey) to ensure the integrity of the join-request [13]. The join-request message is not encrypted because the end device does not have the necessary session keys before sending it to the network server [2, 13]. The network server checks the integrity of the received request and validates it. The network server sends a join-accept message when the device is allowed to participate in the network. Otherwise, the server drops the request of the device and does not send a message to the end device [2]. The join-accept message contains the necessary information to select a communication channel, to derive the session keys for encryption and the unique DevAddr to tag messages from the end device. The server sends the join-accept message encrypted by the AppKey to the end device.

2.3 Vulnerabilities in the OTAA handshake

The OTAA handshake is considered a secure way to authenticate end devices since each end device derives a unique set of session keys each time the device connects to the network [9]. The problem with the current OTAA handshake is that the join request is sent unencrypted via the exposed network. An adversary can easily eavesdrop the join-request and extract the containing values perform any attack to the network. According to [7], an eavesdropper might find out how the attacked network is structured and perform specific attacks to the network because the attackers can collect and analyze all join-requests in their target area.

Another problem appears with the algorithm to generate the random identification value. In [7] the authors noticed that the seed pool for the random number generation gets smaller if the end device becomes stationary. In [11] the authors showed that within one year the procedure would generate an already used random number because an end device has to re-join the network once a day. Both cases result in an algorithm, which only generates a constant number after a sufficient amount of time. A constant number causes problems with the joining procedure because the network server keeps track of the last sent random values per end device and drops requests with the same random value to countermeasure a replay attack. A replay attack describes the re-sending of collected join-requests to the network server to either claim to be the original device or to force the network server to drop the requests of the initial device. Moreover, a malicious node may collect all sent join-requests from the end device to hack the random number generator to generate new join-requests, using the DevEUI and the AppEUI of the attacked device [11].

In [14] an experiment showed, how easy it is to perform a replay attack. Therefore, the authors eavesdropped the join-request, extracted the DevNonce random value and sent them to the network server after a sufficient amount of time. The problem is that there is no timing information available in the request package. An attacker can exploit the collected DevNonce values to send already used join-request again to the network server, which cannot validate the time of the message and processes them again. If the attacker collected enough join-requests, the attack might cause a DoS for the original device, or the attacker joins the network replacing the original device [9, 14].

A similar attack is a wormhole attack [15], where an adversary forwards packages to another node, which is located in a different part of the network. In LoRa networks this attack can be used to replace a valid end device with a malicious one from a different part of the network or to perform a replay attack to cause a DoS to the original device without replacing it. According to [9], two devices are used to realize a wormhole attack. The first node pretends to be a gateway to a certain network and collects all join-requests of an attacked device and forwards them to the second node. This second node sends the received join-requests to the network server, via a real gateway and connects to the network. The original device does not receive any messages and sends the request again to the network. The malicious gateway again collects the sent request to extend the DoS and forwards them to the second node. The second node then uses the requests to reconnect to the real network.

The described threats to the OTAA join procedure exploit the unencrypted join-request to either disconnect devices from the network or to smuggle malicious nodes into the network. Most deployed end devices join a LoRa network via OTAA, therefore encrypting the join-request message would enhance the security of OTAA and LoRaWAN.

3. Paragraphs and Itemizations

LoRaWAN devices already implement an AES algorithm, which the end devices and the network server use to decrypt respectively encrypt the join-accept message and to encrypt messages after a successful join to the network. When the end device and the network server also implement an asynchronous cryptosystem, it can be combined with the existing synchronous cryptosystem to enable a hybrid cryptosystem. The proposed extension uses a hybrid cryptosystem to encrypt the join-request of OTAA before the end device transfers the request to the network server for validation. Similar to the current join procedure, the new handshake then is executed between the end device and the network server. The gateway has the same responsibilities as before and passes the messages through to the network server or the end device.

The network server implements the decryption function and generates the key pair for the asynchronous cryptosystem. The network server provides the public key to all end devices, which want to join the LoRa network. End devices implement a corresponding encryption function to encrypt packages with the public key to complete the hybrid cryptosystem. An end device must obtain the public key from the network server to start the new handshake. Each end device is equipped with a unique 128-bit AES key (AppKey), which end devices use to encrypt the generated join-request. The devices use the public key of the network server to encrypt the AppKey. After the network server received these two encrypted messages, it decrypts the join-request and validates it according to the OTAA process. The new join procedure now includes the following steps to connect to a LoRa network:

1) The end device connects to the network server to request the public key of the server (communication-request).

2) The network server transfers the public key to the end device (communication-accept).

3) The end device creates the join-request for OTAA and encrypts the request with its AppKey and the AppKey with the received public key from the network server.

4) The end device transfers the encrypted join-request and the encrypted AppKey separately to the network server.

5) The network server decrypts the AppKey using its private key and decrypts the join-request with the just decrypted AppKey.

6) The network server validates the join-request and creates a join-accept message if applicable and sends it encrypted by the AppKey to the end device.

Figure 2 shows the extended handshake between the end device and the network server. Grey shaded boxes highlight additional or altered messages of the new handshake.

The handshake adds the communication-request, the communication-accept and the encrypted AppKey message to the join procedure. The join-request message is now transferred encrypted to the network server. The communication-request is a small package from the end device, which the network server acknowledges by sending the public key in the communication-accept package. The end device again confirms the receipt of the public key, when it sends the encrypted join-request. The requested acknowledgements ensure that each end device receives the

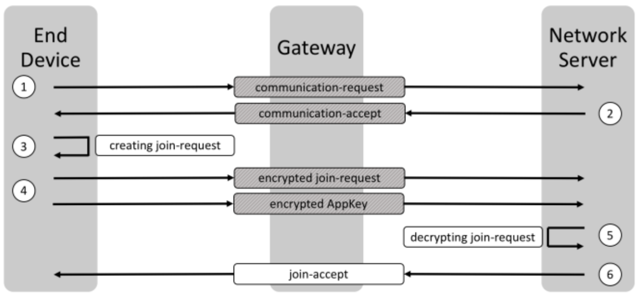

Fig. 2 The new join procedure using a hybrid cryptosystem

public key of the network server. The new handshake enhances the security of the join-request and the AppKey by encrypting them by different keys and sending them separately to the network server. Simultaneously the handshake reduces the number of exchanged messages between the end devices and the network server, by sending the encrypted join-request and the encrypted AppKey without requesting an acknowledgement to the network server because the end device will receive a join-accept message if it is allowed to participate in the network. End devices use the AppKey as the symmetric key for the new handshake to bypass the need for generating a new symmetric key, which benefits power and computational constrained end devices.

4. Implementation Requirements

While executing the new handshake the end device and the network server exchange the encrypted join-request, the encrypted symmetric key and the public key, using the communication-request and communication-accept messages. These communication packages must satisfy the specification of LoRaWAN and should be as small as possible to transfer these packages with any data rate offered by the regional ISM band used by LoRaWAN. Each end device and the network server must also implement an asynchronous cryptosystem to complete the hybrid cryptosystem to enable the proposed handshake.

4.1 The LoRaWAN package specification

LoRaWAN [5] distinguishes between uplink messages, where the end device sends messages to the network server, and downlink messages, where the network server sends messages to the end device. Following each uplink transmission, the end device opens two short receive windows for downlink transmissions. If the network server plans to send data to the end device, it will always initiate the transmission at the beginning of one of those two transmission windows. The frequency and the used data rate can be modified through MAC commands. Uplink and downlink messages use the LoRa radio packet explicit

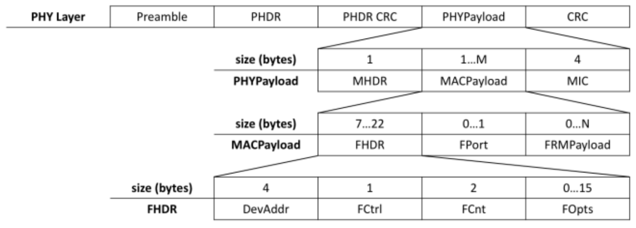

Fig. 3 The package format of LoRaWAN for uplink messages [5]

mode, where the LoRa physical header (PHDR) and a header CRC (PHDR CRC) are included in each packet. Figure 3 shows the packet specification of LoRaWAN for uplink messages, which all messages from the end device must satisfy.

Uplink messages add a payload CRC at the end of each message to ensure the integrity of the transferred physical payload (PHYPayload). Downlink messages do not contain a payload CRC in the physical layer (PHY Layer) to keep messages from the network server as short as possible. The payload of the physical layer (PHYPayload) contains a MAC message header (MHDR), the payload (MACPayload) and a message integrity code (MIC). The length of the MACPayload is region specific and contains the frame header (FHDR), an optional port field (FPort) and the application payload (FRMPayload). The FHDR contains the device address (DevAddr), a frame control field (FCtrl), a frame counter (FCnt) and an optional frame options field (FOpts).

The FHDR is used to identify the sender of the package, to number the packages and to administrate the connection by setting control bits and exchanging MAC commands. MAC commands are never visible to the application of the end device or the application server and exchanged either piggybacked in the FOpts field or instead of the FRMPayload between the end device and the network server. If the FHDR contains a FRMPayload, the FPort field is set to an application specific non-zero value. If the FRMPayload only contains MAC commands the FPort field must be zero. The MHDR defines the message type of the package. LoRaWAN knows seven different message types: join-request, join-accept, unconfirmed data for uplink and downlink messages, confirmed data for uplink and downlink messages and proprietary messages. Proprietary messages are used to support message formats, which cannot be realized with the other message types. The network server and the end device must have a common understanding of the message format extension to use this message type. For the encrypted join-request and the encrypted symmetric key, an unconfirmed data package offers the proposed behavior of reducing the number of exchanged messages. For transferring the communication-request and the public key (communication-accept), a confirmed data package offers the best security and triggers the intended acknowledgement messages by the receiver. To create an acknowledgement, message the receiver sets the acknowledgement bit (ACK) in the FCtrl in the next message it sends to the sender. An acknowledgement message is only sent once in response to the latest message and never retransmitted.

End devices and network servers follow different policies if they do not receive an acknowledge package for their messages. An end-device might have lost the connection to the network, which means that it has to lower its data rate to regain connectivity to the network and resend the lost packet. The network server considers the end device as unreachable and does not apply any further actions and resends the packet. If the maximum number of retransmissions is reached, the end device and the network server can decide if the message in question is retransmitted again or forfeit. During the new handshake, the end device transfers the communication-request as a confirmed package to the network server, which means that the network server has to transfer the public key as an acknowledge package to each end device. If the end device does not receive the public key, the device resends the communication-request. The end device confirms the receipt of the public key, by sending the encrypted join-request as an acknowledge package. The network server cancels the connection process if it does not receive an acknowledgement, and the end device has to restart the handshake.

The engaged devices must recognize the new involved messages as valid LoRa packages. Therefore, the newly introduced messages are transferred either as the MACPayload of a message or as the FRMPayload of a container message. The receiver of the message must be able to distinguish between the different messages either from the package header or by the total amount of transmitted data. In the current LoRaWAN specification the message type code 110 is not used. The new handshake uses it to identify the communication-request at the network server and the communication-accept at the end device. The new message type is named communication-start and triggers the same behavior as a confirmed data package to force the receiver to confirm the receipt of the message by the next package of the handshake. The network server has to assign a temporary DevAddr to each end device, which wants to join the network because it has to recognize related packages of the same end device. The device will receive the real DevAddr for the communication in the join-accept message. Each message which contains a FPort field must set it to a non-zero value to indicate a non-empty FRMPayload. The different fields of the packet structure for each new packet are set as follows:

communication-request: The communication-request message uses the new message type communication-start to let the network server recognize the sender as a new end device. The MACPayload of the package contains an eight-byte random value and the DevEUI of the device, to identify the request and to give the network server a possibility to counteract replay attacks by keeping track of the DevEUI and the random values. The MIC value for the PHYPayload is computed as listed in the equations (1) and (2) to ensure the integrity of the message.

$$\mathrm{mic} = (\mathrm{DevEUI} \mid \mathrm{MHDR}) \oplus \mathrm{random} \qquad (1)$$
$$\mathrm{MIC} = \mathrm{mic}[0..3] \qquad (2)$$

communication-accept: The communication-accept message type is communication-start to identify the response of the network server. The FRMPayload contains the public key of the network server, and the FHDR includes a temporary DevAddr, which the network server uses to assign the next messages to this end device. The ACK bit in the FHDR is set to acknowledge the receipt of the communication-request. The MIC value for the accept is computed as listed in the equation (3), using the first 16 bytes of the public key (pk) to ensure the integrity of the communication-accept. As in the request, four bytes are used for the MIC field, as displayed in equation (2).

$$\mathrm{cmac} = \mathrm{aes128cmac}(\mathrm{pk}[0..15], \mathrm{MHDR} \mid \mathrm{FHDR} \mid \mathrm{FPort} \mid \mathrm{FRMPayload}) \qquad (3)$$

encrypted join-request: The end device generates a join-request, which it encrypts with the AppKey. A container package carries the encrypted join-request as the FRMPayload to the network server. The message type in the MHDR of the container message is set to an unconfirmed uplink message and the DevAddr field in the FHDR is set to the received temporary DevAddr. The ACK bit in the FCtrl field of FHDR is set to one to confirm the receipt of the public key. The MIC of the container message is computed as in equation (3) and set as in equation (2).

encrypted AppKey: The encrypted AppKey is transferred as the FRMPayload of another container message. The message type in the MHDR is again an unconfirmed uplink message, and the DevAddr field in the FHDR is set to the temporary address. The ACK bit is set to zero because the encrypted join-request confirmed the last message from the network server. The MIC of the message is computed and set as in the equations (3) and equation (2).

4.2 Regional data rate and payload differences

LoRaWAN supports various data rates for each supported ISM band. The data rate defines the maximal MACPayload size transmitted in each packet [6]. In the beginning, the LoRa network enables end devices to choose any available data rate for their connection, and the network tries to use the fastest possible data rate to connect to an end device. To manage the data rates to different end devices and to optimize the communication channels, the network can adapt the data rates, by a technique called adaptive data rate (ADR). When ADR is enabled, each end device can request an evaluation of the connection by setting the ADR field in the FCtrl field of the FHDR. The network server then may adapt the current data rate to a lower or higher data rate. Lower data rates benefit the battery life of the end device but also lowers the amount of data transferred at each connection. Faster data rates enable bigger data packets, but therefore reduces the reliability of the connection and

Table 1: Minimal and Maximal MACPayload size in different regions [6]

Band	Min MACPayload		Max MACPayload	
	Size [byte]	Data Rates	Size [byte]	Data Rates
EU 433 MHz	59	0, 1, 2	230	4, 5, 6, 7
EU 863-870 MHz	59	0, 1, 2	230	4, 5, 6, 7
US 902-928 MHz	19	0	250	3, 4
CN 470-510 MHz	59	0, 1, 2	230	4, 5
CN 779-787 MHz	59	0, 1, 2	250	6
KR 920-923 MHz	73	0	250	2, 3, 4, 5
AS 203 MHz	59	0, 1, 2	230	5, 6, 7
AU 915-928 MHz	19	0	250	3, 4

may cause retransmissions. Table 1 shows the minimal and maximal MACPayload size in bytes and the corresponding data rates for each ISM band defined in [6].

The available space for the FRMPayload in the MACPayload depends on the FHDR and the FPort. The size of the FHDR varies between seven and 22 bytes because it contains the piggybacked MAC commands in the FOpts field. The size of the FPort is fixed to one byte. The rest is available for the FRMPayload. If the total size of the MACPayload exceeds the maximal transmission load of the applied data rate, the package must be split into multiple packages before it can be transferred to the receiver. For downlink messages, LoRaWAN supports a mechanism to divide the package into multiple smaller ones. Therefore, the network server sets the frame-pending bit in the FCtrl field of the FHDR to notify the end device that it has to open another receive window as soon as possible to receive the next data packet from the network server. For uplink messages, no such mechanism is available, which makes an adoption to a higher data rate necessary to transfer the data in one packet to the network server.

The total size of the new message packages depends on the transferred MAC commands and the key size of the implemented asynchronous cryptosystem. The security of the used cryptosystem depends on the used key size, which causes a tradeoff between security and package size because a too small key size may cause problems for the safety of the LoRa network, but a to big key size may be impossible to transfer via a lower data rate. According to [16] the minimal key size for secure communication is 2048 bit for RSA encryption, and 224 bit for Elliptic Curve Cryptography (ECC). ECC enables the same security as RSA with shorter keys, which makes it a great alternative for resource constrained applications. To ensure the same level of security as AES-128, a key length of 3072 bit for RSA and 256 bit for ECC are necessary [16, 17]. RSA-2048, RSA-3072, ECC-224 and ECC-256, are used to approximate the MACPayload size of the new messages for different cryptosystems. The findings of [17] are used to approximate the length of the resulting chipper text for each

Table 2: MACPayload size range in bytes for different cryptosystems

	RSA-3072	ECC-256	RSA-2048	ECC-224
communication-request	16	16	16	16
communication-accept	264-279	36-51	392-407	40-55
encrypted join-request	40-55	40-55	40-55	40-55
Encrypted AppKey	264-279	63-78	392-407	70-85

cryptosystem. Table 2 lists the resulting package size range in bytes for each cryptosystem and each packet.

The communication-request has a constant length because the request itself is transferred as the MACPayload and no FHDR is present. The encrypted join-request does not depend on the used asynchronous cryptosystem, so the size range is constant for all four cryptosystems because the join-request gets encrypted by the 128-bit AES key of the end device. The total payload size of the communication-accept and the encrypted AppKey varies because of the key size differences of the four asymmetric cryptosystems. As seen in table 2, LoRaWAN has to split the packets of the RSA algorithm to transfer the public key and the encrypted AppKey. Since uplink messages cannot be divided, it is impossible to transfer the encrypted AppKey to the network server, which makes RSA inapplicable for the extension. Because of the required security strength, which should be comparable with AES-128, and the limited MACPayload length, an ECC-256 algorithm is the only available algorithm to implement the asynchronous cryptosystem for the new handshake.

4.3 Implementing the asynchronous cryptosystem

The asynchronous cryptosystem extension for the end devices and the network server can be implemented either in hardware or software. The end devices must only implement the encryption function and the network server only the key generation and the decryption function of the asynchronous cryptosystem because it is only used to transfer the symmetric key from the end device to the network server. In [18], the authors implemented a hybrid cryptosystem as a blend of hardware and software. They implemented the asynchronous encryption scheme as software and the symmetric cryptosystem as hardware to enhance the performance of the hybrid cryptosystem. In [19], an efficient hybrid cryptosystem using AES and ECC was implemented using hardware and software. The authors optimized the most complex operations to hardware and implemented the rest in software to enhance the performance of the cryptosystem. However, in [20] the authors analyzed performance and energy consumption differences between a hardware and a software implementation of RSA. They found that the hardware solution is approximately 88 times faster and up to 70 times less energy consuming than the software implementation when the algorithm uses a 2048-bit key. In [21] the performance differences between ECC and RSA are investigated, and the authors found that ECC is faster as RSA when the manner of implementation is the same.

Since LoRaWAN is a LPWAN communication protocol, and some end devices have computational constraints, which must remain fulfilled, further deployed end devices should be extended with a hardware implementation of the asynchronous encryption function. Several hardware implementations of ECC exist on the market and fit the requirements of the new handshake [22]. The problem is that a hardware extension is expensive and it hardly can be ported to already deployed end devices. A software extension for already deployed end devices solves that problem. For the network server, a hardware and a software extension are acceptable because the network server does not have those high computational constraints, and should have enough processing power to perform the additional overhead of decrypting the encrypted join-requests and the symmetric key. Nevertheless, a hardware implementation should be preferred because a hardware solution can be encapsulated into a tamper resistant box to secure the decryption function and the key pair [20].

5. Evaluation

The proposed extension encrypts the join-request of the OTAA procedure before a joining end device transfers it to the network server for validation. Encrypting the join-request enhances the robustness of LoRaWAN against replay or wormhole attacks because it makes malicious nodes unable to eavesdrop the identifiers and random values in the join-request or to decrypt the join-accept message. During a wormhole attack, a malicious node can still transfer eavesdropped requests in a different part of the network, but the second node cannot decrypt the encrypted join-request because of the missing key, which makes wormhole attacks much more unappealing. During the communication-request, the handshake transfers the DevEUI unencrypted through the network. When the node would eavesdrop the communication-request and forward it to a second node or the network server, the attacker would receive the public key of the network server, but the attacker would not be able to create a valid join-request because the adversary also needs the AppEUI to create a valid join-request. The AppEUI is kept a secret on the device and only transferred encrypted to the network server, which makes it impossible to misuse the identifier for any attack. A drawback of encrypting the join-request is that the new join-request is approximately 2.2 to 3.05 times bigger than the original request, and more data has to be transferred until the device can participate in the network, but the benefit of a secure authentication for OTAA surpasses this drawback.

6. Conclusion

In this paper, the two join procedures of LoRaWAN were investigated, and potential threats to both proceedings were presented. An extension to the existing Over-The-Air-Activation (OTAA) authentication process was introduced which uses a hybrid cryptosystem to encrypt the join-request. The hybrid cryptosystem uses the already implemented symmetric cryptosystem and the existing AppKey to encrypt the join-request of OTAA. The asynchronous cryptosystem is used to share the symmetric key with the network server. The handshake got evaluated, and further investigations of the specifications of LoRaWAN were made to ensure the technical feasibility of the extension. Also, the regional differences of LoRaWAN were taken into account to ensure that the extension can be deployed in all supported ISM bands. For the future, the presented handshake must be implemented and evaluated in a test scenario to compare its performance, and its security properties to the current OTAA join procedure. Furthermore, a replacement of the DevEUI in the initial communication-request and extending the join-request with timing information is subject to further research.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internetof things for smart cities,” IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22–32, Feb 2014.

[2] D. Dragomir, L. Gheorghe, S. Costea, and A. Radovici, “A survey on secure communication protocols for IoT systems,” in 2016 International Workshop on Secure Internet of Things (SIoT), Sept 2016, pp. 47–62.

[3] U. Raza, P. Kulkarni, and M. Sooriyabandara, “Low power wide area networks: An overview,” IEEE Communications Surveys Tutorials, vol. 19, no. 2, pp. 855–873, Secondquarter 2017.

[4] "LoraWAN what is it? - a technical overview of LoRa and LoRaLAN,” LoRa Alliance, Tech. Rep., 11 2015. [Online]. Available: https://www.lora-alliance.org/lorawan-white-papers

[5] LoRaWAN Specification, LoRa Alliance, Inc., 7 2016, v1.0.2.

[6] LoRaWAN Regional Parameters, LoRa Alliance, Inc., 7 2016, v1.0.

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

[8] B. Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C. Wiley, 2017. [Online]. Available: https://books.google.co.jp/books?id=Ok0nDwAAQBAJ

[9] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, “Exploring the security vulnerabilities of LoRa,” in 2017 3rd IEEE International Conference on Cybernetics (CYBCON), June 2017, pp. 1–6.

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

[11] S. Tomasin, S. Zulian, and L. Vangelista, “Security analysis of LoRaWAN join procedure for internet of things networks,” in 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), March 2017, pp. 1–6.
[12] “LoraWAN security full end-to-end encryption for IoT application providers,” Alliance, Tech. Rep., 02 2017. [Online]. Available: https://www.lora-alliance.org/lorawan-white-papers
[13] R. Miller, “Lora security: Building a secure LoRa solution,” MWR InfoSecurity, Tech. Rep., 03 2016. [Online]. Available: https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf
[14] S. Na, D. Hwang, W. Shin, and K.-H. Kim, “Scenario and countermeasure for replay attack using join request messages in LoRaWAN,” in 2017 International Conference on Information Networking (ICOIN), Jan 2017, pp. 718–720.
[15] T. Giannetsos, T. Dimitriou, and N. R. Prasad, “State of the art on defenses against wormhole attacks in wireless sensor networks,” in 2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, May 2009, pp. 313–318
[16] E. Barker, “Recommendation for key management,” National Institute of Standards and Technology, Tech. Rep., 01 2016. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4
[17] B. Alese and E. D Falaki, “Comparative analysis of public-key encryption schemes,” International Journal of Engineering and Technology, vol. 2, pp. 1552–1568, 10 2012.
[18] A. Nadjia and A. Mohamed, “AES IP for hybrid cryptosystem RSA-AES,” in 2015 IEEE 12th International Multi-Conference on Systems, Signals Devices (SSD15), March 2015, pp. 1–6.
[19] A. Hafsa, N. Alimi, A. Sghaier, M. Zeghid, and M. Machhout, “A hardware-software co-designed AES-ECC cryptosystem,” in 2017 International Conference on Advanced Systems and Electric Technologies (IC ASET), Jan 2017, pp. 50–54.
[20] A. S. Alkalbani, T. Mantoro, and A. O. M. Tap, “Comparison between RSA hardware and software implementation for WSNS security schemes,” in Proceeding of the 3rd International Conference on Information and Communication Technology for the Moslem World (ICT4M) 2010, Dec 2010, pp. E84–E89.
[21] N. Thiranant, Y. S. Lee, and H. Lee, “Performance comparison between RSA and elliptic curve cryptography-based QR code authentication,” in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, March 2015, pp. 278–282.
[22] H. Houssain, M. Badra, and T. F. Al-Somani, “Hardware implementations of elliptic curve cryptography in wireless sensor networks,” in 2011 International Conference for Internet Technology and Secured Transactions, Dec 2011, pp. 1–6.

Kevin Feichtinger received the B.Sc. degree in Computer Science from Johannes Kepler University in 2016. Currently, he studies computer science (M.Sc.) at Johannes Kepler University Linz in Austria. His major is software engineering.

Yuto Nakano received B.E. and M.E in Electrical and Electronic Engineering from Kobe University, Japan in 2006 and 2008, respectively. He joined KDDI and has been engaged in the research on hash functions and stream ciphers. He is currently a researcher at the Information Security Lab. of KDDI Research, Inc.

Kazuhide Fukushima received his M.E. in Information Engineering from Kyushu University, Japan, in 2004. He joined KDDI and has been engaged in the research on digital rights management technologies, including software obfuscation and key management schemes. He received his Doctorate in Engineering from Kyushu University in 2009. He worked for NFC service planning division of KDDI Corporation during 2012 and 2015. He is currently a research manager at KDDI Research, Inc. He received the IEICE Young Engineer Award in 2012. He is a member of Institute of Electronics, Information and Communication Engineers, Information Processing Society of Japan, and ACM.

Shinsaku Kiyomoto received his B.E. in engineering sciences and his M.E. in Materials Science from Tsukuba University, Japan, in 1998 and 2000, respectively. He joined KDD (now KDDI) and has been engaged in research on stream ciphers, cryptographic protocols, and mobile security. He is currently a senior manager at the Information Security Laboratory of KDDI Research, Inc. He was a visiting researcher of the Information Security Group, Royal Holloway University of London from 2008 to 2009. He received his doctorate in engineering from Kyushu University in 2006. He received the IEICE Young Engineer Award in 2004 and IEICE Achievement Award in 2016, respectively. He is members of JPS and IEICE.