Extended 3-Party ACCE and Application to LoRaWAN 1.1*

Sébastien Canard[1] and Loïc Ferreira[1,2]

[1] Orange Labs, Applied Crypto Group, Caen, France
[2] Univ Rennes, INSA Rennes, CNRS, IRISA, France
{sebastien.canard,loic.ferreira}@orange.com

Abstract. LoRaWAN is an IoT protocol deployed worldwide. Whereas the first version 1.0 has been shown to be weak against several types of attacks, the new version 1.1 has been recently released, and aims, in particular, at providing corrections to the previous release. It introduces also a third entity, turning the original 2-party protocol into a 3-party protocol. In this paper, we provide the first security analysis of LoRaWAN 1.1 in its 3-party setting using a provable approach, and show that it suffers from several flaws. Based on the 3(S)ACCE model of Bhargavan et al., we then propose an extended framework that we use to analyse the security of LoRaWAN-like 3-party protocols, and describe a generic 3-party protocol provably secure in this extended model. We use this provable security approach to propose a slightly modified version of LoRaWAN 1.1. We show how to concretely instantiate this alternative, and formally prove its security in our extended model.

Keywords: Security protocols · Security model · Internet of Things · LoRaWAN.

1 Introduction

Establishing a secure communication between two parties is a fundamental goal in cryptography as well as formally proving that such a protocol is secure. In their seminal paper, Bellare and Rogaway [9] propose a security model for the symmetric 2-party setting, and describe provably secure mutual authentication and key exchange protocols. Subsequent models have been proposed (e.g., [8, 13, 15, 17, 18, 30, 35, 41] to name a few). Of particular interest are the security models proposed to analyse real-world protocols such as TLS [30, 34, 41], IPsec and IKE [16, 19, 26], SSH [6]. All these models consider protocols in a 2-party setting. However there exist concrete deployments making use of protocols defined or improperly seen as 2-party schemes, that involve, in fact, three (or more) entities, which different cryptographic operations are attributed to. For example, the 3G/4G mobile phone technology can be described at first glance as a 2-party scheme involving, on the one hand, a set of end-devices, and, on

* Extended version of the paper accepted at Africacrypt 2019.

the other hand, a backend network owned by the operator. However, when the end-device is abroad, the communication is relayed by a server affiliated with a different operator. Such a protocol, unlike classical 2-party setting, requires three participants, and known 2-party security models cannot be seamlessly applied to such a 3-party setting.

Whereas the field of 2-party protocols has been intensively investigated, the 3-party case has received less attention so far. Yet (unsurprisingly) this does not prevent 3-party protocols from being deployed in real-life, despite the lack of a suitable security model that allows seizing precisely, and incorporating their specifics. An example of such a protocol is LoRaWAN 1.1.

1.1 Overview of LoRaWAN 1.1

The LoRaWAN protocol has been designed to set up a Low-Power Wide-Area Network (LPWAN) based on a long range, low rate, and wireless technology dedicated to IoT and M2M. The version 1.0 [51] has been released in 2016. It has been shown to be weak against several types of attacks [4], hence its security is quite questionable. The last version, LoRaWAN 1.1 [50], has been published in 2017. This version aims, in particular, at providing corrections to the previous release, and is assumed to be more secure. A typical LoRaWAN 1.1 network involves a set of end-devices (ED) communicating with a backend network through the intermediary of a Network Server (NS), which is the entry point to the network. Security in LoRaWAN 1.1 is based on two static symmetric master keys: a *communication key*, and an *application key*. In order to establish a secure channel, the ED executes a key exchange with a Join Server (JS), relayed by the NS. Once the session keys are derived, they are provided by the JS to the NS, and to an Application Server (AS). During the subsequent application phase, messages are exchanged between the ED and the NS, which may forward data to the AS (see Figure 1). Thus, NS is an important node in the network, and, together with the ED and the JS, is involved in the LoRaWAN 3-party authenticated key exchange and secure channel establishment.

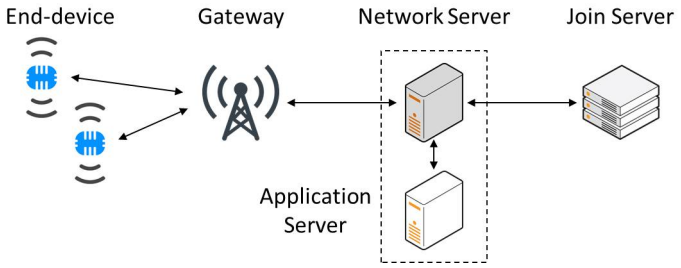

Fig. 1. LoRaWAN 1.1 network (simplified view)

Whereas the version 1.0 of LoRaWAN describes a 2-party protocol between an end-device (ED) and a Network Server (NS), the version 1.1 introduces a third entity: the Join Server (JS). In this new version, the cryptographic operations of the authenticated key exchange are now shared among these three components. This change turns the previous 2-party protocol into a non-standard 3-party protocol with *unknown* security properties. Tampering with a 2-party protocol in order to turn it into a 3-party protocol should be done with care. This motivates a formal analysis in order to define the security goals, and to verify that the protocol meets the latter.[3]

1.2 Related Work

Security Models. Alt, Fouque, Macario-Rat, Onete, and Richard [2] analyse the authenticated key exchange of the 3G/4G mobile phone technology in its complete 3-party setting (with the addition of components from the core network). Based on their formal analysis, they describe how to provide a much stronger security with a small modification which can be easily incorporated in the protocol (despite previous results which indicate privacy flaws and suggest strong changes). Regarding the same key exchange scheme, Fouque, Onete, and Richard [27] use a 3-party security model to show that several remediations proposed in order to thwart end-device-tracking attacks are, in fact, ineffective. In addition, they propose an improvement that aims at mitigating these attacks while retaining most of the 3G/4G key exchange scheme structure.

Regarding the secure channels, Bhargavan, Boureanu, Fouque, Onete, and Richard [12] consider the use of TLS when it is proxied through an intermediate middlebox (such as a Content Delivery Network (CDN)). They propose the notion of 3(S)ACCE-security in order to analyse such a setting. This model extends the classical 2-party ACCE model of Jager, Kohlar, Schäge, and Schwenk [30] to the 3-party setting, and in particular adds, to the properties of entity authentication and channel security, a third property aiming at "binding" several entities involved in the protocol. They describe several attacks targeting a specific CDN architecture, and show that the latter does not meet its claimed security goals.

Naylor, Schomp, Varvello, Leontiadis, Blackburn, Lopez, Papagiannaki, Rodriguez, and Steenkiste [42] describe a *multi-context TLS* protocol (mcTLS) which extends TLS to support middleboxes, in order to offer in-network services. With mcTLS the middlebox becomes visible to the client and the server. In addition these two end-points control the (read, write) privileges attributed to the middlebox.

In turn, Naylor, Li, Gkantsidis, Karagiannis, and Steenkiste [43] propose *Middlebox TLS* (mbTLS) which aims, in particular, at supporting legacy client, and being almost seamlessly integrated in current TLS deployments (based on new TLS extensions). Although Naylor et al., and Naylor et al. list a set of security requirements that mcTLS and mbTLS are supposed to guarantee, they do not formally prove in an explicit security model that their proposal actually achieves

[3] In the remainder of this paper, "LoRaWAN" refers to "LoRaWAN 1.1".

these security goals.

In the same context of proxied TLS connections, Bhargavan, Boureanu, Delignat-Lavaud, Fouque, and Onete [11] describe several types of attacks against mcTLS, showing that the latter is in fact insecure. They propose a security model called *authenticated and confidential channel establishment with accountable proxies* (ACCE-AP), and describe a generic 3-party construction secure in their model, that they instantiate with TLS 1.3 [46]. Their model aims at providing fine-grained rights (defined through the context) to the middlebox. They observe that their model is complex and achieves limited record-layer guarantees in multi-middlebox setting. Their analysis also suggests that the most important middlebox be placed closest to the server.

These works illustrate that 3-party protocols deserve suitable security models in order to be properly analysed and to enlighten subtleties that, otherwise, would remain ignored to the cost of the security.

LoRaWAN 1.1. Butun, Pereira, and Gidlund [14] make a threat analysis of LoRaWAN 1.1. Their results do not mention any novel attack or vulnerability compared to what has been observed regarding the LoRaWAN 1.0 architecture. Among the few threats they consider relevant (the others being physically intruding an ED and deploying a rogue gateway), Butun et al. describe an attack aiming at exhausting the daily quota of frames ED can send, which is based on a technical limitation that they incorrectly attribute to LoRaWAN (citing another paper [1] that mentions in fact Sigfox with respect to that limitation). They suggest several recommendations in order to improve the security of LoRaWAN 1.1 such as protecting the secret keys stored by ED in a tamper-resistant module (as recommended by the specification [50]), using public-key cryptography in order to update the master keys, or replacing the cnt_E and cnt_J parameters (used as input in the session keys derivation) with a nonce negociated during the key exchange (and to be used during the next key exchange). Seemingly, one nonce only, sent by ED, is used in that proposal. The goal seems to preclude a reuse of the nonce. NS keeps track of all received nonces, and discards any key exchange message carrying an reused value.

The rationale behind Butun et al.'s proposal is unclear to us since the current cnt_E and cnt_J parameters are monotonically increasing counters which are not supposed to wrap around (under the same session keys). In addition, in that proposal, the nonce must be updated after a *successful* key exchange. Therefore, a simple attack appears feasible. Upon reception of a RekeyInd command, NS deems that the key exchange is correct, and updates the nonce (in particular, the nonce used during the current key exchange is stored in the list of old nonces). Let us assume that an attacker forbids ED from receiving the RekeyConf command sent by NS. Then the key exchange is not successful from ED's perspective. Hence, following Butun et al.'s proposal, it does not update the nonce, and uses the same nonce during the next key exchange. Yet this nonce is an old one for NS, which discards the Join Request message. Since ED does not receive a response from NS, it keeps using the same nonce (which is continuously rejected

by NS). ED is then unable to reach the back-end network once and for all.

Dönmez and Nigussie [22] investigate the possible vulnerabilities due to the fallback mechanism allowed in version 1.1. They consider the case when ED in version 1.1 faces NS in version 1.0 (and then switches back to version 1.0) and conclude, predictably, that the attacks against version 1.0 become possible anew. Yet they do not consider the case when the 2-byte monotonically increasing counter that ED uses in version 1.1 meets same values as the 2-byte pseudo-random parameter used by ED in version 1.0. They also investigate the case (not treated by the LoRaWAN specification) when ED in version 1.0 faces NS in version 1.1 (which is assumed by Dönmez et al. to fall back to version 1.0), and come essentially to the same conclusion.

Dönmez and Nigussie [21] consider the possibility to perform against LoRaWAN 1.1 (i.e., when both ED and NS implement that version) the attacks targeting version 1.0 (summarised in [22]) and conclude they are not feasible.

LoRaWAN 1.1 uses two master keys MK_1 and MK_2 in order to cryptographically separate the communication layer (between ED and NS) and the application layer (between ED and AS). Dönmez et al. note that when NS owns the communication master key, it can access the application layer. Indeed, in such a case, NS can compel ED to fall back to version 1.0 which uses one key only to protect the communication and the application layers: the communication master key MK_1. Therefore surrendering MK_1 to NS defeats the purpose of the double-key scheme and the existence of JS in LoRaWAN 1.1. The LoRaWAN 1.1 specification claims that "*when working with a v1.1 Network Server, the application session key is derived only from the* [application master key]*, therefore the* [communication master key] *may be surrendered to the network operator to manage the JOIN procedure without enabling the operator to eavesdrop on the application payload data*" ([50], §6.1.1.3). Dönmez et al. show that this statement is wrong. As a mitigation, they propose ED (and AS) keeps computing the application session key as in version 1.1.

Dönmez et al. claim that the involvement of intermediary NS servers (the so-called *serving* and *forwarding* NS) between ED and its *home* NS extends the possibility to alter an application frame. Yet, according to us, this is incorrect. We acknowledge the lack of clarity of the LoRaWAN 1.1 specification regarding this point. Nonetheless, according to the specification [50] (cf. §6.1.2.3) and a companion document [53] (cf. §11.3.1 and §12.2.1) both serving NS and forwarding NS may be able to verify at least partially the MAC tag of an uplink frame (which means by the way that, at this point, data integrity relies upon 16 bits only instead of 32 bits since these servers cannot verify the whole tag). Eventually, the home NS *should* verify the whole 32-bit MAC tag.

Dönmez et al. also investigate the possibilities enabled by a malicious NS. They suggest a session key reuse (hence the possibility to replay or decrypt application frames) if the counter cnt_J managed by JS wraps around (despite the fact that the specification demands the latter be not possible). We observe that this allows to target AS but not ED. Indeed the session keys computation involves also the cnt_E counter managed by ED. If, in such a case, a malicious NS

can replay to JS any Join Request message (previously received from ED) which carries an old counter value, ED faithfully uses this monotonically increasing counter for the key derivation. Dönmez et al. recommend JS to keep track of the counter values sent by ED (the specification demands only NS do this). In contrast, we explain in Section 2.2 how a malicious (or corrupted) NS can compel AS to reuse a previous session key, without JS's counter wrapping around.

Finally, Dönmez et al. observe that the protocol does not provide forward secrecy (as it is based on static master symmetric keys), and suggest to apply the proposal of Kim and Song [31]. The latter consists essentially in using the current session keys to perform the next key exchange, and then to replace the current keys (or the initial master keys during the first key exchange) with the newly computed session keys. We observe that this update mechanism implies that one party makes the first move (it is ready to use the new keys) while the other still holds the current keys. Then the other party follows upon reception of one of the messages sent during the key exchange. An issue arises when that message, which triggers the replacement of the keys, is not received. The two parties are then desynchronised with respect to the keys evolution. In such a case, both parties remain unable to perform any subsequent key exchange. This means that ED is forbidden once and for all from reaching the back-end network. Neither Kim et al. nor Dönmez et al. explain how to maintain this essential synchronisation between ED and the back-end network.[4]

In turn, Han and Wang [28] propose that ED and JS update the LoRaWAN master keys prior to each new key exchange. Their proposal implies exchanging two additional messages during the key exchange phase. According to us, this proposal leads also to a synchronisation issue. One party (ED or JS) has to make the first move (i.e., replacing its current master keys with the new ones). Then the other party does the same upon reception of some message sent during the key exchange phase. If this message is not received, then the parties are desynchronised with respect to their master keys. In such a case, they are unable to perform any subsequent key exchange. Han et al. do not deal with this issue, and leave it unsolved.

Eldefrawy, Butun, Pereira, and Gidlund [24] perform a formal security analysis of LoRaWAN 1.1 using the Scyther verification tool. They conclude to the absence of weaknesses in the protocol. Yet they acknowledge there may still exist vulnerabilities not found due to the limitations of the model they employ.

In contrast to previous analyses, we address, in this paper, the new 3-party aspect of the LoRaWAN protocol introduced in version 1.1, using a provable security approach. We also describe new vulnerabilities.

1.3 Contributions

In this context, our contributions are threefold:

[4] An example of an authenticated key exchange protocol in the symmetric-key setting that provides forward secrecy and explicitly deals with the synchronisation issue is the SAKE protocol [3].

1. We present an improved security model that we call 3-ACCE, based on that of Bhargavan et al. [12]. This meets the need of a general framework that incorporates the subtleties of a LoRaWAN-like protocol, and allows expliciting the security requirements of such 3-party protocols. As additional enhancements, we add (i) ED authentication, (ii) the security operations done by NS during the channel establishment, and (iii) an extended "binding" property that links all the entities involved in the key exchange.
2. We describe a generic 3-party protocol that is provably secure in our enhanced model. That is, we provide a general theorem with its full proof in our 3-ACCE security notion. This generic protocol can be concretely instantiated with LoRaWAN but also other protocols.
3. We present the first security analysis of LoRaWAN 1.1 in its 3-party setting using a provable approach. First, we describe several flaws that weaken the protocol. Next, we apply our generic result to LoRaWAN 1.1, and propose a slightly modified version of the protocol which achieves stronger security properties. We show how to concretely instantiate this alternative, and formally prove its security in our extended 3-party model.

1.4 Paper Outline

In Section 2, we describe the protocol LoRaWAN 1.1, and show that it suffers from several flaws that enable theoretical attacks. A general framework that we call 3-ACCE, and aiming at analysing the security of 3-party protocols is presented in Section 3. In addition, we propose a generic 3-party protocol that we formally prove to be secure in this extended model. We use this framework, in Section 4, to propose a slightly modified version of LoRaWAN 1.1 with stronger security properties, that we prove to be secure in our 3-ACCE model. Finally, we conclude in Section 5.

2 LoRaWAN 1.1

2.1 Description of the Protocol

We recall the main lines of the LoRaWAN 1.1 protocol. Figure 2 depicts the protocol. A complete description can be found in the specification [50].

Architecture. Three entities are involved in the key exchange and secure channel establishment:

- ED: the end-device, wireless sensor or actuator, that communicates with the NS through gateways.
- NS: the entry point to the network.
- JS: the server, located in the backend network, that owns the master keys of each ED.

A fourth entity, the AS, participates to the session once the authenticated key exchange is completed, and the secure channel is established.

LoRaWAN 1.1 offers three sub-protocols to establish a session, called *Join procedure*, *Rejoin type 1 procedure*, and *Rejoin type 0/2 procedure*. The Join procedure is the standard way to start a session. The Rejoin type 1 procedure is an "emergency" method aiming at reconnecting an ED in case of total loss of the cryptographic context by the NS. The Rejoin type 0/2 procedure is mainly used to change the radio parameters, even if it may also be used to update the session keys. In this paper we consider the method likely the most used to execute the protocol, that is the Join procedure.

Authentication and Key Exchange. LoRaWAN 1.1 is a protocol based on shared (static) master keys. Each ED stores two distinct 128-bit master keys: a *communication key* MK_1, and an *application key* MK_2, and JS owns the list of all the master keys.

Initiated only by ED, the key exchange is made of four main messages. The first two (*Join Request* and *Join Accept*) are used to mutually authenticate ED and JS, and to share the data used to compute the 128-bit session keys. The other two (*RekeyInd* and *RekeyConf*) are used to validate the session keys. The Join Request message sent by ED carries three main parameters: JS's identifier id_J, ED's identifier id_E, the current ED's counter cnt_E. These parameters are protected with a 4-byte MAC (= AES-CMAC) tag computed with the master key MK_1.[5]

$$\begin{aligned}&\text{Join Request} = id_J \| id_E \| cnt_E \| \tau_E\\ &\text{with } \tau_E = \mathsf{MAC}(MK_1, id_J \| id_E \| cnt_E)\end{aligned}$$

Upon reception of the Join Request message, NS checks that the cnt_E counter is valid (i.e., greater than the last value received from that ED), and forwards the message to JS. In turn, JS verifies the MAC tag, and computes a Join Accept response. This message carries a counter cnt_J, NS's identifier id_N, and other parameters $prms$ (such as radio parameters).[6] It is protected with a CMAC tag computed under a (static) master key MK_3, which is derived from ED's master key MK_1 and ED's identifier id_E. The MAC tag involves in addition the parameters cnt_E and id_J sent by ED.[7] The data carried in the Join Accept message are encrypted with the AES decryption function in ECB mode, and the master key MK_1. Once ED receives the Join Accept message, it verifies the

[5] For the sake of clarity, we slightly simplify the formulas and do not make appear the value corresponding to the message's type (which is also sent and involved in the MAC tag computation).

[6] Since these parameters are not relevant to the remaining of this paper, we skip their description and refer the interested reader to the specification [50].

[7] This aims at forbidding a replay of previous Join Accept messages to the ED (which the previous version of the protocol, LoRaWAN 1.0, is subject to [4]).

MAC tag and the cnt_J counter.

$$\begin{array}{l}\text{Join Accept} = \mathsf{AES}^{-1}(MK_1, cnt_J \| id_N \| prms \| \tau_J)\\ \text{with } \tau_J = \mathsf{MAC}(MK_3, id_J \| cnt_E \| cnt_J \| id_N \| prms)\\ \text{and } MK_3 = \mathsf{KDF}_{mk}(MK_1, id_E)\end{array}$$

The KDF_{mk} function corresponds to

$$\mathsf{KDF}_{mk}(K, x) = \mathsf{AES}(K, \texttt{0x06} \| x \| cst')$$

where cst' is some constant value.

In order to validate the session keys, ED sends a special message called RekeyInd that triggers the change, by NS, of its security context. In turn, NS sends a RekeyConf response. These messages are computed as any other post-accept messages (i.e., sent through the secure channel). Akin to the *Finished* messages in TLS, these messages, protected with the session keys, are used to conclude the key exchange phase.

Session Keys Computation. The counters cnt_E and cnt_J (sent during the key exchange) are unique per ED. They are initialised to 0 and monotonically increased (respectively by ED and JS) at each new session. From these two counters, id_J, and the master keys MK_1, MK_2, ED and JS compute four 128-bit session keys $K_c^{i_1}$, $K_c^{i_2}$, K_c^e, and K_a^e:

$$\begin{cases} K_c^{i_1} \| K_c^{i_2} \| K_c^e = \mathsf{KDF}_c(MK_1, cnt_J \| id_J \| cnt_E) \\ K_a^e \qquad\qquad\quad\;\; = \mathsf{KDF}_a(MK_2, cnt_J \| id_J \| cnt_E) \end{cases}$$

The session keys $K_c^{i_1}$, $K_c^{i_2}$, K_c^e are given by JS to NS (through an undefined by the specification but allegedly secure protocol). K_a^e is given by JS either to AS (through a protocol undefined by the specification), or to NS (in such a case K_a^e is encrypted with a key independent of LoRaWAN, only known to JS and AS [53]). The KDF_c function, on input a key K and value x, outputs the three following values

$$\begin{cases} K_c^{i_1} = \mathsf{AES}(K, \texttt{0x01} \| x \| cst) \\ K_c^{i_2} = \mathsf{AES}(K, \texttt{0x03} \| x \| cst) \\ K_c^e \;\; = \mathsf{AES}(K, \texttt{0x04} \| x \| cst) \end{cases}$$

where cst is some constant value. The function KDF_a is defined as

$$\mathsf{KDF}_a(K, x) = \mathsf{AES}(K, \texttt{0x02} \| x \| cst)$$

Secure Channel. To that point, ED can send protected messages to the network. The messages are encrypted with AES-CTR and K_c^e or K_a^e depending on the message type. A *command message* is encrypted with K_c^e and exchanged between ED and NS. An *application message* is encrypted with K_a^e and exchanged between ED and AS. LoRaWAN provides data integrity only between ED and NS (and relies upon an additional – and undefined – protocol to guarantee data

integrity between NS and AS). The messages are MAC-ed with two different functions (depending on the direction) which are based on a tweaked version of AES-CMAC (a block is prefixed to the input), and output a 4-byte tag. In the downlink direction, the MAC function uses the key $K_c^{i_1}$, and corresponds to (tweaked) AES-CMAC which output is truncated to 32 bits. In the uplink direction, the function used is $\mathsf{MAC}_{\|}$ defined as

$$\mathsf{MAC}_{\|}(K_c^{i_1}, K_c^{i_2}, x) = \mathsf{MAC}_b(K_c^{i_1}, x) \| \mathsf{MAC}_b(K_c^{i_2}, x)$$

where MAC_b corresponds to the downlink MAC function which output is truncated to 16 bits.

2.2 Cryptographic Flaws in LoRaWAN 1.1

Several vulnerabilities that lead to likely practical attacks against LoRaWAN 1.0 have been corrected with version 1.1. Nonetheless, some peculiarities of this last version still allows impairing the security of a LoRaWAN network.

Size of the Counters. The counters cnt_E, cnt_J are respectively 2-byte and 3-byte long. It is likely that so few values can be exhausted, which brings ED to be unable to initiate a new session and be lastly (if not for good) "disconnected" from the network.

There are two other methods that allow ED to initiate a session (the so-called Rejoin procedures). However the Rejoin type 0/2 procedure is available only if a session is *ongoing* (because the first request is sent through the current secure channel). As for the Rejoin type 1 procedure, it is invoked periodically based on a predefined frequency, which means that it is not available at will.

The specification states that if the cnt_E counter wraps around, then ED must use a different id_J value (parameter used in the Join Request and Join Accept messages, and in the session keys computation). In fact, $id_J \| cnt_E$ behaves as a counter where id_J corresponds to the most significant bits, and cnt_E to the least significant bits. Therefore it may not be enough to exhaust the cnt_E counter in order to stuck ED. However, we do think that, due to lack of clarity of the specification regarding the rationale in storing more than one id_J value into ED, and the fact that LoRaWAN 1.1 inherits from the previous version of the protocol, it is likely that only one id_J value will be stored into ED (as in the previous version, where cnt_E is a pseudo-random value). Moreover it has been shown [4] that it is possible to compel ED to repeatedly send Join Request messages, hence to likely use all the cnt_E values. Therefore exhausting ED's counter appears feasible.

Assuming that ED sends one Join Request message every 5 seconds [36], the duration of this attack is $2^{16} \times 5$ seconds $\simeq 91$ hours. Note that, if ED stores k samples of id_J values, the duration of the attack is $k \times 91$ hours.

Now, let us assume that ED stores several id_J values. Then the attacker can target the cnt_J counter. As said above, the attacker can compel ED in repeatedly sending Join Request messages. Each Join Request message triggers a new Join Accept response, hence consumes a new cnt_J value. Yet, cnt_J is 3-byte

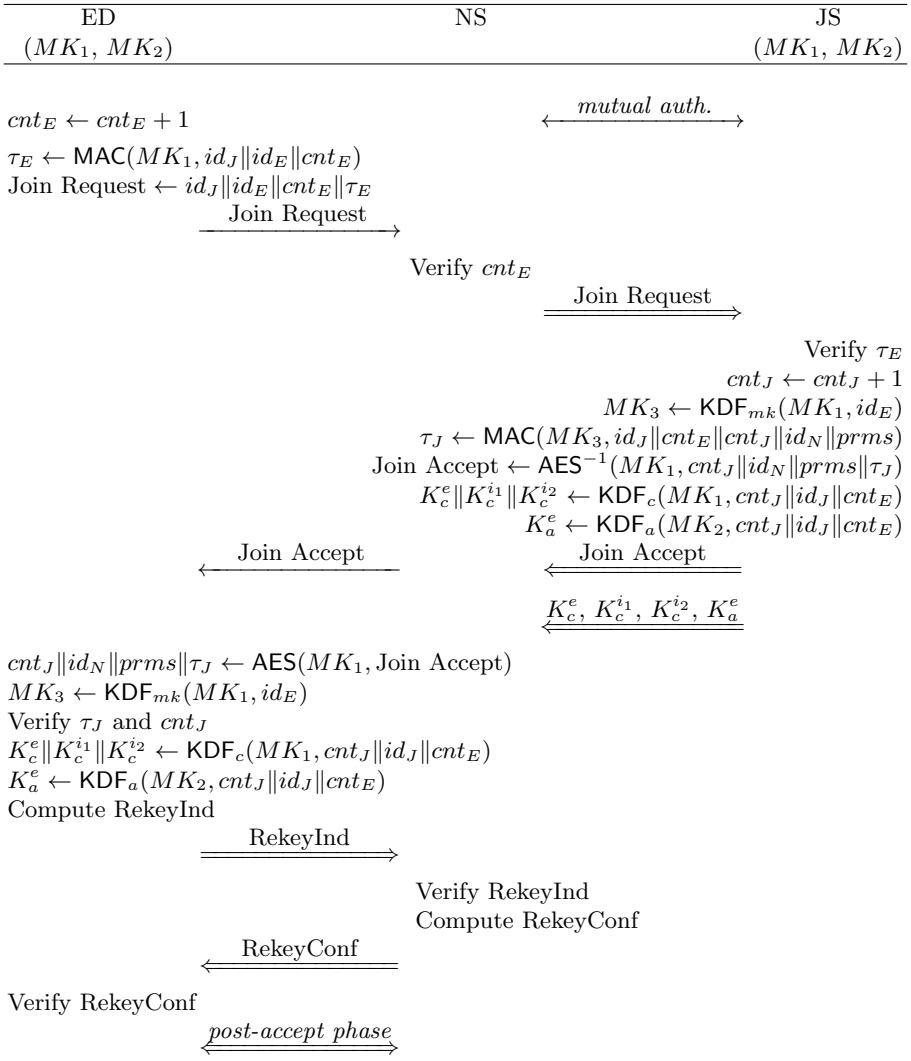

Fig. 2. Correct execution of LoRaWAN 1.1. Double line arrows indicate the use of secure channel keys. There are two secure channels: ED-NS (LoRaWAN), and NS-JS (undefined).

long, whereas cnt_E is 2-byte long. Therefore, in order for ED to send as many Join Request messages as possible cnt_J values, ED must store a number of id_J values equal to $|cnt_J|/|cnt_E| = 2^{24}/2^{16} = 256$. The duration of this attack is $2^{24} \times 5$ seconds = 2.66 years. This is a very long attack, yet less than the expected lifespan of ED (up to 10 years), and it ends up with ED being possibly

unable to connect the back-end network ever again.

The only remaining possibility in order for ED to connect the back-end network is the Rejoin type 1 procedure. This is an "emergency" procedure aiming at reconnecting ED in case of total loss of the cryptographic context by NS, and the latter is *not* compelled to respond to a Rejoin type 1 Request.

Size of MAC Tags. The MAC's output is 4-byte long. Hence, MAC forgeries are made easier, and, in combination with the fact that data encryption is done in CTR mode, so are attacks against data integrity. Note that the duration of this attack is higher if the attacker acts on the air interface than if she is able to act in the back-end network.

Known Encryption Keystream. Per specification, ED must send a (encrypted) RekeyInd message *as long as* it does not receive a RekeyConf response (up to a fixed number of RekeyInd messages, afterwards ED must start a new session). Conversely, NS must respond to *each* RekeyInd message with a (encrypted) RekeyConf response. The (plaintext) content of both kind of messages is *known*. Hence, an attacker can get multiple *valid encryption keystreams* for free. If she succeeds in forging a valid MAC tag, then she can get messages carrying the plaintext of her choice. Of course, simple encryption does not provide data integrity, and the attacker needs to forge a valid MAC tag. However this provides a way to compute encrypted messages which underlying plaintext is semantically correct.

In order to collect the keystreams, the attacker can forbid NS from receiving the RekeyInd messages, or discard the RekeyConf messages sent by NS. This compels ED to send multiple messages. The adversary collects all these messages (and lets the first RekeyInd reach NS so that NS uses the new session keys). The adversary can then use the other $n-1$ messages to try to forge a valid message (i.e., compute a valid MAC tag).

Similarly, the adversary can forbid ED from receiving the RekeyConf message sent by NS (which receives the RekeyInd messages from ED). This compels NS to respond with multiple messages. The adversary collects all these messages. It sends the first RekeyConf message to ED (in order for ED to continue the session) and can use the remaining $n-1$ RekeyConf messages in order to mount the attack.

The parameter $n =$ `ADR_ACK_LIMIT` is at most 2^{15}. Nonetheless, the default settings [36] in several geographical areas (e.g., USA, Europe, China) demand that $n = 64$. Therefore, the attacker has at his disposal between 64 and 2^{15} frames (in both directions).

Downgrade Attack. According to the specification, an ED implementing version 1.1 must fall back to version 1.0 when it faces an NS implementing version 1.0. Hence, even an ED in version 1.1 may succumb to the attacks that have been shown possible against LoRaWAN 1.0 [4]. Therefore, a current deployment

of LoRaWAN 1.1 may inherit the flaws of the previous version. This scenario implies that the (upgraded) ED in version 1.1 either uses as its MK_1 master key the same key used in version 1.0, or has already faced an NS in version 1.0.

It may be possible that, when executing 1.0, ED implements also the recommendations published by LoRa Alliance [37] which aim at strengthening version 1.0. In such a case, the attacks against that version are (partially or completely) mitigated.

Lack of Data Integrity. There is no end-to-end data integrity provided by LoRaWAN between ED and AS. The specification demands data integrity be guaranteed by an additional (and undefined) protocol between NS and AS. At the same time, a companion document [53] demands that data integrity (and other security properties such as data confidentiality and mutual authentication) be ensured hop by hop between the components of a LoRaWAN network. Managing data security in such an hop-by-hop fashion is hazardous because it does not take into account the intermediate servers between NS and AS. Handing down security properties that is, according to us, incumbent upon the LoRaWAN protocol may lead to security breaches, as some of these servers (such as a MQTT server) have been shown to be insecurely managed [38].

An attacker that succeeds in accessing a weak point between NS and AS can exploit the lack of data integrity in a LoRaWAN application frame, and alter or truncate the frame. It may also be possible to break data confidentiality by applying a kind of "message oracle attack". The attacker first makes a guess regarding the plaintext data encrypted in some message. She replaces the alleged message with a (per specification defined) LoRaWAN command. Based on the behaviour of the AS (it may apply the commands chosen by the attacker, or reject the message), the attacker learns if her guess is correct (hence deduce the plaintext data).[8]

Reuse of an Application Session Key. The session key K_a^e used by AS to encrypt application messages is either sent by JS to AS (through a protocol undefined by the specification), or sent by JS to NS (which relays it to AS with the corresponding encrypted application frame). In the latter case, K_a^e is encrypted with a key known only to JS and AS. The LoRaWAN specification does not make clear the properties of the security scheme used to wrap K_a^e. We observe that if this scheme does not provide non-replayability of the messages, then it may be possible to compel AS to reuse a previous application session key K_a^e. The attacker acts on a weakly protected intermediary point between NS and AS where it can replace the current application key (and the corresponding application frame, due to the lack of end-to-end data integrity between ED and AS) with a previous one.

[8] Rupprecht, Kohls, Holz, and Pöpper [48] have shown (though in another context) the consequences of the combination of encryption in CTR mode and lack of data integrity.

If AS reuses a past application session key, all the previous messages encrypted by ED with this key become cryptographically valid anew. Hence the attacker can replay these messages to AS. Furthermore, AS uses that same key to send back new application frames to ED. Since encryption is done in CTR mode, the attacker can decrypt application messages. Indeed, two different messages (each from two distinct sessions) protected with the same key, and corresponding to the same counter, are encrypted with the same keystream. Therefore the bitwise combination of the two encrypted messages is equal to the combination of the two corresponding plaintexts. Hence the two plaintexts can be partially or completely retrieved, in an obvious manner if either message is known, or through statistical analysis [39].

Another issue arises also. The application frame encrypted by AS with a *previous* session key K_a^e, and intended to ED, is MAC-ed by NS (with the current MAC session key). Upon reception of that frame, ED deems it is correct since data integrity is valid. However decryption with the *current* session key yields roughly garbage. It is unclear how ED behaves with the output of the decryption process.

Malicious or Corrupted NS. A malicious or corrupted NS which receives the encrypted application keys K_e^a from JS can apply the scenario described above, and compel AS to reuse a previous application key. Note that this can be done even if the malicious or corrupted NS does not own any master key. More generally, a third party reduces the security of a client-server type connection (as in LoRaWAN 1.0) by increasing the attack surface. Whereas a given ED is bound to a given JS, many NS servers may relay the data between an ED and its JS and AS. Thus the security of a whole network can be shattered by a malicious NS or the weakest NS which relays data back and forth between many ED and AS.

2.3 The Need for a Suitable Security Model

During the key exchange phase, the only cryptographic operation that NS does, in order to accept ED as partner, is verifying the RekeyInd message with keys received from JS. This allows the following theoretical attack (see Figure 3). If the attacker, on the one hand, succeeds in sending keys of her choice to NS on behalf of JS, she can, on the other hand, provide a consistent RekeyInd message (computed under these keys), bringing NS to accept although no ED (and possibly no JS) is actually involved in the session. The attacker is then able to send valid messages to NS on behalf of ED (the same session keys are used to compute the RekeyInd message and the subsequent messages of the post-accept phase).

This scenario implies being able either to impersonate JS to NS, or to break the channel security established (with a protocol undefined by the LoRaWAN specification) between NS and JS. This attack does *not* even need to target the core LoRaWAN protocol. It is conceivable because of the way the cryptographic

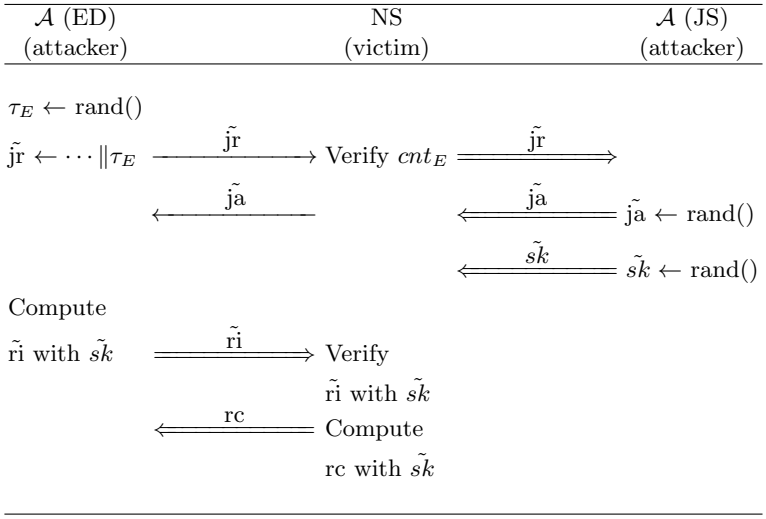

Fig. 3. Impersonation of ED based on a weak protocol between NS and JS

operations in LoRaWAN are shared between ED, NS and JS, and interleaved with the undefined protocol used between NS and JS. This highlights how the security of LoRaWAN crucially depends on this additional protocol. Analysing LoRaWAN implies to take the latter into account.

LoRaWAN 1.1 is a 3-party protocol, not a 2-party protocol between a client (ED) and a backend network (NS-JS). Assessing its security (as a 3-party protocol) needs care. Therefore, it requires a suitable security model that incorporates all its subtleties, and makes explicit the security requirements which, for some of them (such as the protocol between NS and JS), are barely mentioned in the specification despite their crucial role in the overall security of a LoRaWAN network. In Section 3, we describe a 3-ACCE security model that aims at capturing the security goals of such 3-party protocols.

3 Extended 3-ACCE Model

3.1 Preliminaries

In this section, we recall the definitions of the main security notions we use in our results.

Pseudo-Random Function. A *pseudo-random function* (PRF) F is a deterministic algorithm which given a key $K \in \{0,1\}^\lambda$ and a bit string $x \in \{0,1\}^*$ outputs a string $y = F(K,x) \in \{0,1\}^\gamma$ (with γ being polynomial in λ). Let $Func$ be the set of all functions of domain $\{0,1\}^*$ and range $\{0,1\}^\gamma$. The security of

a PRF is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \xleftarrow{\$} \{0,1\}^\lambda$, $G \xleftarrow{\$} Func$, and $b \xleftarrow{\$} \{0,1\}$ uniformly at random.
2. The adversary may adaptively query values x to the challenger. The challenger replies to each query with either $y = F(K,x)$ if $b = 1$, or $y = G(x)$ if $b = 0$.
3. Finally, the adversary outputs her guess $b' \in \{0,1\}$ of b.

The adversary's advantage is defined as

$$\mathsf{adv}_F^{\mathsf{PRF}}(\mathcal{A}) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

Definition 1 (Secure PRF). *A function F:$\{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\gamma$ is said to be a* secure PRF *if, for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_F^{\mathsf{PRF}}(\mathcal{A})$ is a negligible function in λ.*

Pseudo-Random Permutation. A *pseudo-random permutation* (PRP) F is a deterministic algorithm which given a key $K \in \{0,1\}^\lambda$ and a bit string $x \in \{0,1\}^\gamma$ outputs a string $y = F(K,x) \in \{0,1\}^\gamma$ (with γ being polynomial in λ). Let $Perm$ be the set of all permutations on $\{0,1\}^\gamma$. The security of a PRP is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples $K \xleftarrow{\$} \{0,1\}^\lambda$, $G \xleftarrow{\$} Perm$, $b \xleftarrow{\$} \{0,1\}$ uniformly at random.
2. The adversary may adaptively query values x to the challenger. The challenger replies to each query with either $y = F(K,x)$ if $b = 1$, or $y = G(x)$ if $b = 0$.
3. Finally, the adversary outputs her guess $b' \in \{0,1\}$ of b.

The adversary's advantage is defined as

$$\mathsf{adv}_F^{\mathsf{PRP}}(\mathcal{A}) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

Definition 2 (Secure PRP). *A function F:$\{0,1\}^\lambda \times \{0,1\}^\gamma \to \{0,1\}^\gamma$ is said to be a* secure PRP *if, for all probabilistic polynomial time adversary $\mathcal{A}$, $\mathsf{adv}_F^{\mathsf{PRP}}(\mathcal{A})$ is a negligible function in λ.*

Stateful Authenticated Encryption. A *stateful authenticated encryption scheme* (sAE) consists of two algorithms $\mathsf{StAE} = (\mathsf{StAE.Enc}, \mathsf{StAE.Dec})$. The encryption algorithm, given as $(C, st'_e) \leftarrow \mathsf{StAE.Enc}(K, H, M, st_e)$, takes as input a secret key $K \in \{0,1\}^\lambda$, a header data $H \in \{0,1\}^*$, a plaintext M, and the current encryption state $st_e \in \{0,1\}^*$. It outputs an updated state st'_e, and either a ciphertext $C \in \{0,1\}^*$ or an error symbol $\perp$. The decryption algorithm,

given as $(M, st'_d) \leftarrow \mathsf{StAE.Dec}(K, H, C, st_d)$, takes as input a key K, a header data H, a ciphertext C, and the current decryption state st_d. It outputs an updated state st'_d, and either a value M, which is the message encrypted in C, or an error symbol $\perp$. The states st_e and st_d are initialised to the empty string $\emptyset$. The security of a sAE scheme is defined with the following experiment between a challenger and an adversary $\mathcal{A}$:

1. The challenger samples uniformly at random $K \xleftarrow{\$} \{0,1\}^\lambda$, and $b \xleftarrow{\$} \{0,1\}$.
2. The adversary may adaptively query the encryption oracle Encrypt and the decryption oracle Decrypt, as described by Figure 4.
3. Finally, the adversary outputs her guess $b' \in \{0,1\}$ of b.

This game captures both the confidentiality and integrity properties of a stateful AEAD scheme. The adversary's advantage is defined as

$$\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}}(\mathcal{A}) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

Definition 3 (Secure sAE). *The encryption scheme* StAE *is said to be a* secure sAE encryption scheme *if, for all probabilistic polynomial time adversary* $\mathcal{A}$, $\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}}(\mathcal{A})$ *is a negligible function in* λ.

$\underline{\mathsf{Encrypt}(M_0, M_1, H)}$	$\underline{\mathsf{Decrypt}(C, H)}$
$u \leftarrow u + 1$	if $b = 0$ then return $\perp$
$(C^0, st^0_e) \xleftarrow{\$} \mathsf{StAE.Enc}(K, H, M_0, st_e)$	$v \leftarrow v + 1$
$(C^1, st^1_e) \xleftarrow{\$} \mathsf{StAE.Enc}(K, H, M_1, st_e)$	$(M, st_d) \leftarrow \mathsf{StAE.Dec}(K, H, C, st_d)$
if $C^0 = \perp$ or $C^1 = \perp$ then return $\perp$	if $v > u$ or $C \neq C_v$ or $H \neq H_v$
$(C_u, H_u, st_e) \leftarrow (C^b, H, st^b_e)$	then $sync \leftarrow \texttt{false}$
return C_u	if $sync = \texttt{false}$ then return M
	return $\perp$

Fig. 4. The Encrypt and Decrypt oracles in the sAE security experiment. The counters u and v are initialised to 0, and *sync* to `true` at the beginning of the experiment.

3.2 Execution Environment

We describe the execution environment related to our model, using the notations of the ACCE model of Jager et al. [30], and Bhargavan et al. [12]. We use this execution environment to analyse our generic 3-party protocol Π.

Protocol Entities. Our model considers three sets of parties: a set $\mathcal{E}$ of end-devices, a set $\mathcal{N}$ of Network Servers, and a set $\mathcal{J}$ of Join Servers. Each party is given a long term key ltk.

Session Instances. Each party P_i maintains a set of instances $\mathsf{Instances} = \{\pi_i^0, \pi_i^1, \ldots\}$ modeling several (sequential or parallel) executions of the 3-party protocol Π. Each instance π_i^n has access to the long term key ltk of its party parent P_i. Moreover, each instance π_i^n maintains the following internal state:

- The **instance parent** $\pi_i^n.\mathsf{parent} \in \mathcal{E} \cup \mathcal{N} \cup \mathcal{J}$ indicating the party P_i that owns that instance: $\pi_i^n.\mathsf{parent} = P_i$.
- The **partner-party** $\pi_i^n.\mathsf{pid} \in \mathcal{E} \cup \mathcal{N} \cup \mathcal{J}$ indicating the party $\pi_i^n.\mathsf{parent}$ is presumably running the protocol with. $P_i \in \mathcal{E}$ can only be partnered with a party $P_k \in \mathcal{J}$. $P_k \in \mathcal{J}$ can only be partnered with a party $P_j \in \mathcal{N}$. $P_j \in \mathcal{N}$ can be partnered with either $P_i \in \mathcal{E}$ or $P_k \in \mathcal{J}$.
- The **role** $\pi_i^n.\rho \in \{\mathsf{ed}, \mathsf{ns\text{-}client}, \mathsf{ns\text{-}server}, \mathsf{js}\}$ of $P_i = \pi_i^n.\mathsf{parent}$. If $P_i \in \mathcal{E}$, then $\pi_i^n.\rho = \mathsf{ed}$. If $P_i \in \mathcal{J}$, then $\pi_i^n.\rho = \mathsf{js}$. If $P_i \in \mathcal{N}$, then $\pi_i^n.\rho \in \{\mathsf{ns\text{-}client}, \mathsf{ns\text{-}server}\}$. In such a case, $\pi_i^n.\rho = \mathsf{ns\text{-}client}$ if $\pi_i^n.\mathsf{pid} \in \mathcal{J}$, and $\pi_i^n.\rho = \mathsf{ns\text{-}server}$ if $\pi_i^n.\mathsf{pid} \in \mathcal{E}$.
- The **session identifier** $\pi_i^n.\mathsf{sid}$ of an instance.
- The **acceptance flag** $\pi_i^n.\alpha$ originally set to $\perp$ when the session is ongoing, and set to 1/0 when the party accepts/rejects the partner's authentication.
- The **session keys** $\pi_i^n.\mathsf{ck}$ set to $\perp$ at the beginning of the session, and set to a non-null bitstring corresponding to the encryption and decryption session keys once π_i^n computes the session keys.
- The **key material** $\pi_i^n.\mathsf{km}$ set to $\perp$ if $\pi_i^n.\rho \in \{\mathsf{ed}, \mathsf{ns\text{-}server}\}$. Otherwise km is set to $\perp$ at the beginning of the session, and set to a non-null bitstring once π_i^n ends in accepting state.
- The **security bit** $\pi_i^n.\mathsf{b}$ sampled at random at the beginning of the security experiments.
- The **partner-instances set** $\pi_i^n.\mathsf{ISet}$ stores the *instances* that are involved in the same protocol run as π_i^n (including π_i^n itself).
- The **partner-parties set** $\pi_i^n.\mathsf{PSet}$ stores the *parties* parent of the instances in $\pi_i^n.\mathsf{ISet}$ (including $P_i = \pi_i^n.\mathsf{parent}$ itself).

A correct execution of the protocol Π involves four instances π_i^n, π_j^u, π_j^v, π_k^ℓ such that

- $\pi_i^n.\mathsf{parent} = P_i \in \mathcal{E}$, $\pi_j^u.\mathsf{parent} = \pi_j^v.\mathsf{parent} = P_j \in \mathcal{N}$, $\pi_k^\ell.\mathsf{parent} = P_k \in \mathcal{J}$
- $\pi_j^u.\rho = \mathsf{ns\text{-}server}$ and $\pi_j^v.\rho = \mathsf{ns\text{-}client}$
- $\pi_i^n.\mathsf{sid} = \pi_j^u.\mathsf{sid} \neq \perp$ and $\pi_j^v.\mathsf{sid} = \pi_k^\ell.\mathsf{sid} \neq \perp$
- $\pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = \pi_j^v.\mathsf{km} = \pi_k^\ell.\mathsf{km} \neq \perp$

Then, the partner-instances set and the partner-parties set are defined as $\pi.\mathsf{ISet} = \{\pi_i^n, \pi_j^u, \pi_j^v, \pi_k^\ell\}$ and $\pi.\mathsf{PSet} = \{P_i, P_j, P_k\}$, $\forall \pi \in \{\pi_i^n, \pi_j^u, \pi_j^v, \pi_k^\ell\}$.

Adversarial Queries. An adversary may interact with the instances by issuing the following queries.

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$: this query creates a new session π_i^n with role ρ, executed by party P_i, and intended partner-party pid.

- $\mathsf{Send}(\pi_i^n, M)$: the adversary can send a message M to π_i^n, receiving a response M', or an error message $\perp$ if the instance does not exist or if $\pi_i^n.\alpha = 1$. (Send queries in an accepting state are handled by the $\mathsf{Decrypt}$ query.)
- $\mathsf{Reveal}(\pi_i^n)$: this query returns the session keys $\pi_i^n.\mathsf{ck}$ and the key material $\pi_i^n.\mathsf{km}$ of an instance π_i^n ending in accepting state.
- $\mathsf{Corrupt}(P_i)$: this query returns the long term key $P_i.\mathsf{ltk}$ of P_i.
- $\mathsf{Encrypt}(\pi_i^n, M_0, M_1, H)$: it encrypts the message M_b, $b = \pi_i^n.\mathsf{b}$, with header H, with the encryption session keys (stored within $\pi_i^n.\mathsf{ck}$) of an *accepting* instance π_i^n (if $\pi_i^n.\alpha \neq 1$, then π_i^n returns $\perp$).
- $\mathsf{Decrypt}(\pi_i^n, C, H)$: this query decrypts the ciphertext C with header H, with the decryption session keys (stored within $\pi_i^n.\mathsf{ck}$) of an *accepting* instance π_i^n (if $\pi_i^n.\alpha \neq 1$, then π_i^n returns $\perp$). Figure 5 depicts this oracle.

$\underline{\mathsf{Encrypt}(\pi_i^n, M_0, M_1, H)}$
if $\pi_i^n.\alpha \neq 1$ then return $\perp$
$u \leftarrow u + 1$
$(C^0, st_e^0) \xleftarrow{\$} \mathsf{StAE.Enc}(kenc, H, M_0, st_e)$
$(C^1, st_e^1) \xleftarrow{\$} \mathsf{StAE.Enc}(kenc, H, M_1, st_e)$
if $C^0 = \perp$ or $C^1 = \perp$ then return $\perp$
$b \leftarrow \pi_i^n.\mathsf{b}$
$(C_u, H_u, st_e) \leftarrow (C^b, H, st_e^b)$
return C_u

$\underline{\mathsf{Decrypt}(\pi_i^n, C, H)}$
if $\pi_i^n.\alpha \neq 1$ then return $\perp$
if $\pi_i^n.\mathsf{b} = 0$ then return $\perp$
$v \leftarrow v + 1$
$(M, st_d) \leftarrow \mathsf{StAE.Dec}(kdec, H, C, st_d)$
if $v > u$ or $C \neq C_v$ or $H \neq H_v$
  then $sync \leftarrow \mathtt{false}$
if $sync = \mathtt{false}$ then return M

Fig. 5. The $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ oracles in the 3-ACCE security experiment. The counters u and v are initialised to 0, and *sync* to `true` at the beginning of every session.

3.3 Security Definitions

Partnership. In order to define the partnership between two instances, we use the definition of matching conversations initially proposed by Bellare and Rogaway [9], and modified by Jager et al. [30].

Let $T_{i,n}$ be the sequence of all (valid) messages sent and received by an instance π_i^n in chronological order. For two transcripts $T_{i,n}$ and $T_{j,u}$, we say that $T_{i,n}$ is a prefix of $T_{j,u}$ if $T_{i,n}$ contains at least one message, and the messages in $T_{i,n}$ are identical to the first $|T_{i,n}|$ messages of $T_{j,u}$.

Definition 4 (Matching Conversations). *We say that π_i^n has a* matching conversation *to π_j^u, if*

- *π_i^n has sent all protocol messages and $T_{j,u}$ is a prefix of $T_{i,n}$, or*
- *π_j^u has sent all protocol messages and $T_{i,n} = T_{j,u}$.*

Consequently, we define sid to be the transcript, in chronological order, of all the (valid) messages sent and received by an instance during the key exchange, but, possibly, the last message. We say that two instances π_i^n and π_j^u are pairwise partnered if $\pi_i^n.\mathsf{sid} = \pi_j^u.\mathsf{sid}$. Then, we define the 3-ACCE partnering with the sets ISet and PSet. $\pi_i^n.\mathsf{ISet}$ stores instances partnered with π_i^n, and $\pi_i^n.\mathsf{PSet}$ stores parties partnered with π_i^n.

Correctness. The correctness in 3-ACCE is defined as follows. We demand that, for any instance π ending in an accepting state, the following conditions hold:

- $\forall \pi \in \{\pi_i^n, \pi_j^u, \pi_j^v, \pi_k^\ell\}$, $\pi.\mathsf{ISet} = \{\pi_i^n, \pi_j^u, \pi_j^v, \pi_k^\ell\}$ and $|\pi.\mathsf{ISet}| = 4$
- $\pi_i^n.\mathsf{parent} = P_i \in \mathcal{E}$, $\pi_j^u.\mathsf{parent} = \pi_j^v.\mathsf{parent} = P_j \in \mathcal{N}$, $\pi_k^\ell.\mathsf{parent} = P_k \in \mathcal{J}$
- $\pi.\mathsf{PSet} = \{P_i, P_j, P_k\}$
- $\pi_i^n.\mathsf{ck} = \pi_j^u.\mathsf{ck} = \pi_j^v.\mathsf{km} = \pi_k^\ell.\mathsf{km} \neq \perp$
- $\pi_i^n.\mathsf{sid} = \pi_j^u.\mathsf{sid} \neq \perp$
- $\pi_j^v.\mathsf{sid} = \pi_k^\ell.\mathsf{sid} \neq \perp$

Security of ACCE protocols is defined by requiring that (i) the protocol is a secure authentication protocol, and (ii) in the post-accept phase all data is transmitted over an authenticated and confidential channel in the sense of length-hiding sAE. Security of 3-ACCE protocols is defined in a similar way (but the length-hiding property), but we include an additional requirement in the entity authentication property in order to "bind" all the parties involved in a session. The adversary's advantage to win is defined with two games: the *entity authentication* game, and the *channel security* game. In both, the adversary can query all oracles NewSession, Send, Reveal, Corrupt, Encrypt, and Decrypt.

Entity Authentication (EA). This security property must guarantee that any instance π_i^n ending in accepting state is partnered with a unique instance. In addition to the two parties explicitly involved in the communication, we guarantee that a third party participate in the session (each one belonging to a different set $\mathcal{E}$, $\mathcal{N}$, $\mathcal{J}$). The purpose of this property, that we borrow from Bhargavan et al. [12], is to make sure that if some ED establishes a communication with some NS, there is a JS that is also involved. Conversely if a secure channel is established between an NS and a JS, we want to make sure that it is with the aim of establishing a communication between that NS and some ED. In this EA security experiment, the adversary is successful if, when it terminates, there exists an instance that *maliciously accepts* according to the following definition.

Definition 5 (Entity Authentication). *An instance is said to* maliciously accept *if the adversary succeeds in fulfilling one of the following winning conditions.*

ED adversary – *An instance π_i^n of parent $P_i \in \mathcal{E}$ is said to* maliciously accept *if*

- $\pi_i^n.\alpha = 1$ *and* $\pi_i^n.\mathsf{pid} = P_k \in \mathcal{J}$.

- *No instance in* $\pi_i^n.\mathsf{ISet}$ *was queried in* Reveal *queries.*
- *No party in* $\pi_i^n.\mathsf{PSet}$ *is corrupted.*
- *There is no unique* $\pi_j^u \mid (\pi_j^u.\mathsf{parent} \in \mathcal{N} \wedge \pi_j^u.\mathsf{sid} = \pi_i^n.\mathsf{sid})$, *or there is no* $\pi_k^\ell \in P_k.\mathsf{Instances} \mid \pi_k^\ell.\mathsf{km} = \pi_i^n.\mathsf{ck}$.

<u>*NS adversary*</u> – *An instance* π_j^u *of parent* $P_j \in \mathcal{N}$ *is said to* maliciously accept *if at least one of the following two conditions holds*

(a)
- $\pi_j^u.\alpha = 1$ *and* $\pi_j^u.\mathsf{pid} = P_i \in \mathcal{E}$.
- *No instance in* $\pi_j^u.\mathsf{ISet}$ *was queried in* Reveal *queries.*
- *No party in* $\pi_j^u.\mathsf{PSet}$ *is corrupted.*
- *There is no unique* $\pi_i^n \mid (\pi_i^n \in P_i.\mathsf{Instances} \wedge \pi_j^u.\mathsf{sid} = \pi_i^n.\mathsf{sid})$, *or there is no* $\pi_k^\ell \mid (\pi_k^\ell.\mathsf{parent} = P_k \in \mathcal{J} \wedge \pi_i^n.\mathsf{pid} = P_k \wedge \pi_k^\ell.\mathsf{km} = \pi_j^u.\mathsf{ck})$.

(b)
- $\pi_j^v.\alpha = 1$ *and* $\pi_j^v.\mathsf{pid} = P_k \in \mathcal{J}$.
- *No instance in* $\pi_j^v.\mathsf{ISet}$ *was queried in* Reveal *queries.*
- *No party in* $\pi_j^v.\mathsf{PSet}$ *is corrupted.*
- *There is no unique* $\pi_k^\ell \in P_k.\mathsf{Instances} \mid (\pi_j^v.\mathsf{sid} = \pi_k^\ell.\mathsf{sid})$, *or there is no* $\pi_i^n \mid (\pi_i^n.\mathsf{parent} \in \mathcal{E} \wedge \pi_i^n.\mathsf{pid} = P_k \wedge \pi_i^n.\mathsf{ck} = \pi_j^v.\mathsf{km})$.

<u>*JS adversary*</u> – *An instance* π_k^ℓ *of parent* $P_k \in \mathcal{J}$ *is said to* maliciously accept *if*

- $\pi_k^\ell.\alpha = 1$ *and* $\pi_k^\ell.\mathsf{pid} = P_j \in \mathcal{N}$.
- *No instance in* $\pi_k^\ell.\mathsf{ISet}$ *was queried in* Reveal *queries.*
- *No party in* $\pi_k^\ell.\mathsf{PSet}$ *is corrupted.*
- *There is no unique* $\pi_j^v \in P_j.\mathsf{Instances} \mid (\pi_j^v.\mathsf{sid} = \pi_k^\ell.\mathsf{sid})$, *or there is no* $\pi_i^n \mid (\pi_i^n.\mathsf{parent} \in \mathcal{E} \wedge \pi_i^n.\mathsf{pid} = P_k \wedge \pi_k^\ell.\mathsf{km} = \pi_i^n.\mathsf{ck})$.

The adversary's advantage is defined as its winning probability:

$$\mathsf{adv}_{\Pi}^{\mathsf{EA}}(\mathcal{A}) = \Pr[\mathcal{A} \textit{ wins the } \mathsf{EA} \textit{ game}].$$

Channel Security (CS). In the channel security game, the adversary can use all oracles. At some point, the adversary sends a challenge M_0, M_1 (issuing a query Encrypt) to some instance π_i^n, and gets C_b the encryption of M_b, $b = \pi_i^n.\mathsf{b}$. The adversary is successful if she guesses b. That is, she must output an instance π_i^n and its security bit. The security bit $\pi_i^n.\mathsf{b}$ is chosen at random at the beginning of the game.

Definition 6 (Channel Security). *An adversary* $\mathcal{A}$ *breaks the channel security if she terminates the channel security game with a tuple* (π_i^n, b) *such that*

- $\pi_i^n.\alpha = 1$
- *No instance in* $\pi_i^n.\mathsf{ISet}$ *was queried in* Reveal *queries.*
- *No party in* $\pi_i^n.\mathsf{PSet}$ *is corrupted.*
- $\pi_i^n.\mathsf{b} = b$

The adversary's advantage is defined as

$$\mathsf{adv}^{\mathsf{CS}}_{\Pi}(\mathcal{A}) = \left|\Pr[\mathcal{A} \textit{ wins the } \mathsf{CS} \textit{ game}] - \frac{1}{2}\right|.$$

Definition 7 (3-ACCE-security). *A 3-party protocol Π is 3-*ACCE*-secure if Π satisfies correctness, and for all probabilistic polynomial time adversaries $\mathcal{A}$, $\mathsf{adv}^{\mathsf{EA}}_{\Pi}(\mathcal{A})$ and $\mathsf{adv}^{\mathsf{CS}}_{\Pi}(\mathcal{A})$ are a negligible function of the security parameter.*

3.4 Comparison with Existing Models

The 3-ACCE security notion we propose takes inspiration from that of Bhargavan et al. [12], which is used to analyse the security of TLS when an intermediary server (the middleware) is involved between the client and the server. In turn the latter is built on the Authenticated and Confidential Channel Establishment (ACCE) model introduced by Jager et al. [30] to prove the security of TLS 1.2 [20] in DHE mode, and used by Kohlar et al. to prove the security of TLS 1.2 in RSA and DH modes [32]. Our model follows the same execution environment and adversarial model, and reuse the corresponding notations to deal with the entity authentication and the channel security properties. Yet we relax the stateful length-hiding AE (sLHAE) security used by Jager et al. and use the sAE-security. That is, we do not demand the "length-hiding" property (i.e., the ciphertext hides the length of the corresponding plaintext) for the encryption schemes. Obviously TLS 1.2 remains secure with respect to sAE-security.

The model of Bhargavan et al. includes a property requiring that whenever a client identifies a server as its partner, that server should be able to decrypt channels established between the client and the middleware, hence audit the behaviour of the middleware.[9] In our model, we extend this property in two ways. Firstly, we demand that it be ensured by all parties involved in a correct execution of the protocol, in order to "bind" these parties. Secondly, this property guarantees to JS that an ED is actually involved in the key exchange, prior to establishing the secure channel between NS and the purported ED (hence JS is not merely used as a session keys derivation oracle). This means that when an ED establishes a session with an NS, a JS has been part of the key derivation. When a JS is requested by an NS, there is an ED expecting to connect the network. When NS relays data, it is to enlist an ED with the help of a JS. We demand this additional guarantee because the purpose of such a channel established by JS is *only* to compensate for the cryptographic operations that NS is unable to perform. Another option would have been to separate into two properties: the entity authentication and the "entity binding". This entity binding property, that the three parties involved in the session take on, is a way to extend to three "dimensions" what tie the parties in a classical 2-party protocol. We do mean by "entity authentication" in a 3-party setting a property that guarantees not only a unique partner to a given party, but also the unavoidable involvement of

[9] This property is called *accountability* in [12].

a third party. This is the reason of our choice, despite a possible lack of modularity.

Moreover, as pointed out by Bhargavan et al., their model has two main limitations: it does not handle client authentication, and does not consider forward secrecy.[10] In addition, in the 3-party protocol they propose, when instantiated with TLS 1.2, the middlebox merely forwards messages but does not have any added value. On the contrary, in the model we propose, we do consider client (ED) authentication, and retain the genuine operations done by NS.

The Authenticated and Confidential Channel Establishment with Accountable Proxies (ACCE-AP) model of Bhargavan et al. [11] allows capturing a context where several middleboxes are interspersed between a (TLS) client and a server. It aims at providing fine-grained rights (defined through contexts) to the middleboxes. We choose instead to use the intuitive and elegant 3-ACCE model since, in our setting, one intermediate server only (NS) is involved, which predefined rights are attributed to. Moreover, the authors of [11] observe that their model is complex and achieves limited record-layer guarantees in multi-middlebox setting.

3.5 Building 3-ACCE from 2-ACCE

In this section we describe a generic 3-party protocol Π. Next, we show that Π is generically secure in the 3-ACCE model described in Sections 3.2 and 3.3.

Our Generic 3-party Protocol. The Figure 6 depicts our view of the 3-ACCE protocol Π between ED, NS and JS. It is composed of two distinct protocols denoted P and P' respectively. P is a 2-ACCE protocol between ED and NS, and P' is a 2-ACCE protocol between NS and JS. The details of the protocol Π are given in Figure 7.

Π is generic in the sense that it depicts a whole class of protocols. Informally, this class corresponds to 3-party protocols where one entity behaves mostly as a key server (JS), whereas the post-accept phase is managed by the other two entities (ED, NS). Moreover, the P component has the following features. Its key exchange is made of four main messages: the first two with the major purpose of exchanging the material intended for the key derivation, and the last two in order to confirm the session keys or to authenticate the parties. For example, TLS-PSK [25], SRP [52], and SIGMA-R [33] can be instances of P. As we will see in Section 4, LoRaWAN is such another instance.

Main Theorem and Sketch Proof. Based on the security of P and P', we show that Π is 3-ACCE-secure according to Definition 7.

Theorem 1. *The protocol Π is a secure* 3-ACCE *protocol under the assumption that P is a secure* 2-ACCE *protocol, and P' is a secure* 2-ACCE *protocol, with the*

[10] Our model does not require forward secrecy either because of the use of static symmetric keys in LoRaWAN.

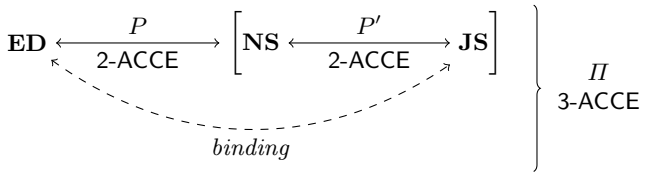

Fig. 6. 3-ACCE protocol Π

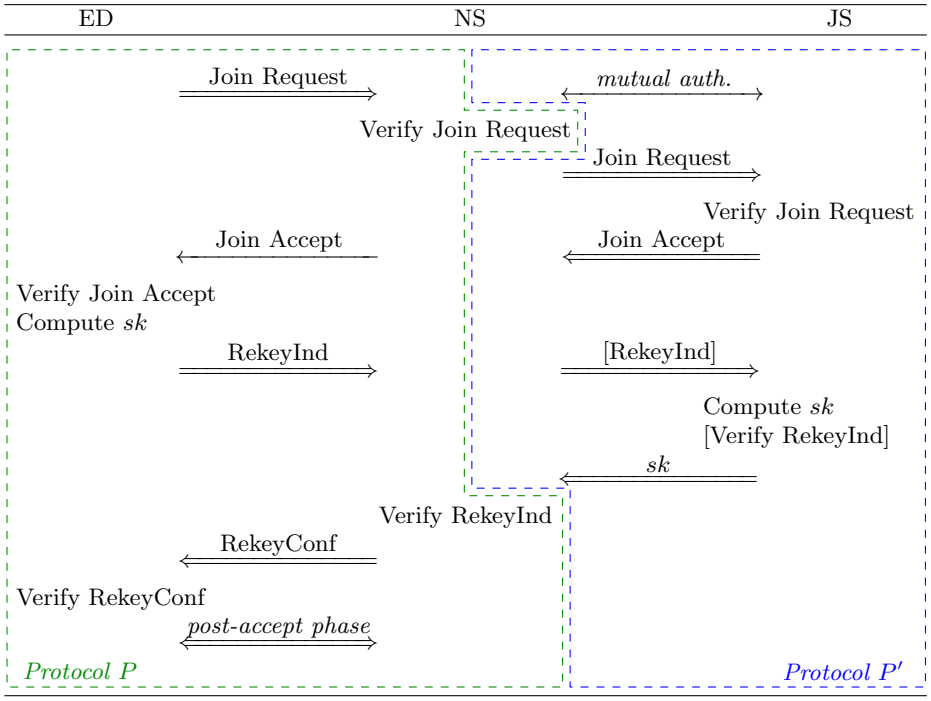

Fig. 7. Correct execution of protocol Π, made of P (left) and P' (right) components. Double line arrows indicate the use of the secure channel keys.

following reductions

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{\Pi} &\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(2\mathsf{adv}^{\mathsf{CS}}_{P} + 3\mathsf{adv}^{\mathsf{CS}}_{P'} + 2p_{jr} + 2p_{ja} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}\right) \\
&\quad + n_{\mathsf{E}} \left(n_{\mathsf{J}} \cdot \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} + n_{\mathsf{N}} \cdot \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}\right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(3\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}\right) \\
\mathsf{adv}^{\mathsf{CS}}_{\Pi} &\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\mathsf{CS}}_{P} + 3\mathsf{adv}^{\mathsf{CS}}_{P'}\right) + \mathsf{adv}^{\mathsf{EA}}_{\Pi}
\end{aligned}$$

where n_{E}, n_{N}, and n_{J} are respectively the number of ED, NS, and JS parties.

We give here only a sketch of proof of Theorem 1. The extended proof is provided in Appendix A. Let us first consider the EA security property. We split the proof into three parts depending which party (ED, NS, JS) the adversary targets.

ED adversary. Roughly speaking, in order to be successful, the adversary must make the ED accept without an NS or a JS being involved. Hence the adversary can first try to impersonate the NS to the ED as in a 2-party execution of protocol P (in such a case no NS and no JS are involved in the session). This corresponds to an advantage $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}}$. The adversary can also try to bypass the intermediate NS in order to get from the JS all the necessary material (Join Accept message, session keys sk) in order for ED to accept. This implies necessarily that a server adversary be able to impersonate a legitimate NS to the JS, that is to break the EA-security of P' (corresponding to an advantage $\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}$). Finally, the adversary can try to make ED and NS have different sid. In order to be successful, the adversary has to provide a valid RekeyInd message to the NS different than the one computed by the ED. This implies either forging such a message, or getting the keys used to compute it and transmitted by the JS to the NS. We reduce both possibilities to the channel security with respect to P on the one hand ($\mathsf{adv}^{\mathsf{CS}}_{P}$), and to the channel security with respect to P' on the other hand ($\mathsf{adv}^{\mathsf{CS}}_{P'}$).

Since we have ruled out the impersonation of NS to ED, and the impersonation of NS to JS, ED uses the Join Accept message sent by the JS upon reception of the Join Request message computed by the ED. Therefore, ED and JS compute the P-session keys with the same inputs (and the same function). Hence they output the same keys (that is $\pi^n_i.\mathsf{ck} = \pi^\ell_k.\mathsf{km}$). In addition, ED and NS have matching conversations (that is, they share the same sid).

Accounting for the fact that the reduction must guess the identity of the three parties involved, the advantage of an ED adversary is bounded by

$$\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{E}} \leq n_{\mathsf{E}} \cdot n_{\mathsf{J}_\mathsf{E}} \left(\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} + n_{\mathsf{N}} \cdot \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} + \mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) \right)$$

where $n_{\mathsf{J}_\mathsf{E}} \leq n_{\mathsf{J}}$ is the number of JSs that can be partnered with a given ED.

NS adversary. First we deal with the winning condition (a). The adversary can try to impersonate the ED to the NS in order to preclude the existence of a partner to NS. This implies breaking the EA-security of P when the server side is targeted. The corresponding advantage is $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$. Then the adversary can try to impersonate a legitimate JS to NS, in order to preclude the involvement of the JS in the protocol run. This corresponds to an advantage $\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}}$.

The only cryptographic operation that the NS does in order to accept is verifying the RekeyInd message it gets from the ED with the keys provided by the JS. Therefore the adversary is successful if, on the one hand, she provides some keys sk to NS (through the P' secure channel), and, on the other hand, she sends to NS a RekeyInd message computed under these keys sk. Note that the

adversary can be successful even if a legitimate JS is involved in the protocol.[11] This is possible if the adversary forges a valid P' application message carrying the keys sk she has chosen. This can be reduced to the channel security with respect to P', which corresponds to the advantage $\mathsf{adv}^{\mathsf{CS}}_{P'}$.

The remaining possibility in order for the adversary to win is to provide a RekeyConf message so that NS and ED do not share the same sid (i.e., they do not have matching conversations).[12] This is possible either if the adversary forges such a message, or if the adversary is able to get the keys used to compute the message, and transmitted by the JS to the NS through a secure channel with respect to P'. We reduce either possibility respectively to the channel security with respect to P ($\mathsf{adv}^{\mathsf{CS}}_{P}$), and to the channel security with respect to P' ($\mathsf{adv}^{\mathsf{CS}}_{P'}$). Furthermore, since we have ruled out the impersonation of JS to NS, and also the possibility to forge P' application messages, NS and JS share the same P session keys. That is $\pi^u_j.\mathsf{ck} = \pi^\ell_k.\mathsf{km}$.

Accounting for the fact that the reduction must guess the identity of the three parties involved, the advantage of an NS adversary in winning through condition (a) is bounded by

$$p_a \leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} + n_{\mathsf{J_E}} \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{CS}}_{P} \right) \right)$$

Regarding condition (b), the adversary can first try to impersonate a legitimate JS to NS in order to preclude the involvement of such a JS. This implies an adversary able to break the EA-security of P' when the client side is targeted, which corresponds to an advantage $\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}}$. Then the adversary can proceed as under condition (a). That is providing to the NS some keys sk of her choice, and a RekeyInd message computed under sk. This implies forging a valid P' application message carrying the keys sk. We reduce such a possibility to the channel security with respect to P', which corresponds to an advantage $\mathsf{adv}^{\mathsf{CS}}_{P'}$. Then, in order to have that NS and JS do not share the same sid (i.e., they do not have a matching conversation), the adversary can try to forge a P' application message (carrying a Join Request or a RekeyInd message) intended to JS.[13] We can reduce the latter to the channel security of P' ($\mathsf{adv}^{\mathsf{CS}}_{P'}$).

So far, this guarantees that NS and JS have a matching conversation, that is they share the same sid ($\pi^v_j.\mathsf{sid} = \pi^\ell_k.\mathsf{sid}$). Finally the adversary wins if the NS and the ED do not share the same P session keys. This is possible if the adversary forges either a Join Request message or a Join Accept message. These two possibilities are respectively bounded by the probabilities p_{jr} and p_{ja} (see Section 4.1).

[11] The adversary sends first a fake Join Request message to the NS (random MAC tag, correct counter cnt_E). Then the adversary sends a random Join Accept to the NS followed by session keys sk of her choice, and a RekeyInd message computed under sk (as depicted in Section 2.3 by Figure 3).

[12] Forging a RekeyInd message is already ruled out because we have precluded the impersonation of ED to NS.

[13] Different session keys are (likely) used in either direction in order to protect P' application messages.

Therefore, accounting that the reduction must guess the identity of the parties involved, the advantage of an NS adversary in winning through condition (b) is bounded by

$$p_b \leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + n_{\mathsf{E}_\mathsf{J}} \left(p_{jr} + p_{ja} \right) \right)$$

where $n_{\mathsf{E}_\mathsf{J}} \leq n_{\mathsf{E}}$ is the number of EDs that can be partnered with a given JS. Therefore

$$\begin{aligned} \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{N}} &\leq p_a + p_b \\ &\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J}_\mathsf{E}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\ &\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + n_{\mathsf{E}_\mathsf{J}} \left(p_{jr} + p_{ja} \right) \right) \end{aligned}$$

with $n_{\mathsf{E}_\mathsf{J}} \leq n_{\mathsf{E}}$, and $n_{\mathsf{J}_\mathsf{E}} \leq n_{\mathsf{J}}$.

JS adversary. In this setting, the adversary can first try to impersonate the NS to the JS, which corresponds to an advantage $\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}$. Then, in order to have that the NS and the JS do not share the same sid (i.e., do not have a matching conversation), the adversary can try to forge one of the messages exchanged through the secure channel (in either direction), which can be reduced to the channel security with respect to P' ($\mathsf{adv}^{\mathsf{CS}}_{P'}$). Ruling out all these possibilities guarantees that JS and NS share the same sid ($\pi^{\ell}_{k}.\mathsf{sid} = \pi^{v}_{j}.\mathsf{sid}$).

Finally, the adversary can try to make the ED and the JS compute different P-session keys. Since these keys depend on the data carried in the Join Request and Join Accept messages, this implies forging either message, corresponding to a probability $p_{jr} + p_{ja}$. Hence, ruling out both possibilities guarantees that $\pi^{n}_{i}.\mathsf{ck} = \pi^{\ell}_{k}.\mathsf{km}$.

Taking account of all the parties involved, the advantage of a JS adversary is bounded by

$$\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{J}} \leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} + \mathsf{adv}^{\mathsf{CS}}_{P'} + n_{\mathsf{E}_\mathsf{J}} \left(p_{jr} + p_{ja} \right) \right)$$

Regarding the CS property of Π we apply the following hops. First we rule out the possibility that an instance maliciously accepts. That is we follow the same steps as in the EA proof. This leads to an advantage equal to $\mathsf{adv}^{\mathsf{EA}}_{\Pi}$. This leaves two possibilities in order for the adversary to be successful: either she targets directly the secure channel between the ED and the NS, or she targets the secure channel between the NS and the JS. We can reduce the latter possibility to the CS-security of P' corresponding to an advantage $\mathsf{adv}^{\mathsf{CS}}_{P'}$. Regarding the former possibility, the adversary can first try to get the P session keys (sk) sent by the JS to the NS (which we reduce to the channel security with respect to P' leading to an advantage $\mathsf{adv}^{\mathsf{CS}}_{P'}$). Then the adversary can try to break the channel security with respect to P ($\mathsf{adv}^{\mathsf{CS}}_{P}$). We have also to take into account that the session keys sk are sent by the JS to the NS through the secure channel provided by P'. Since the CS-security of P relies implicitly on the indistinguishability of

sk from random, we have to rely on the real-from-random indistinguishability for the plaintexts guaranteed by the channel provided by P' (which we reduce to $\mathsf{adv}^{\mathsf{CS}}_{P'}$).
Accounting that the reduction must guess the identity of the parties involved, the advantage of the adversary is bounded by

$$\mathsf{adv}^{\mathsf{CS}}_{\Pi} \leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}^{\mathsf{CS}}_{P} + 3\mathsf{adv}^{\mathsf{CS}}_{P'} \right) + \mathsf{adv}^{\mathsf{EA}}_{\Pi}$$

4 3-ACCE Security with LoRaWAN 1.1

In this section, we use the generic result of Section 3.5, and apply it to LoRaWAN. For this purpose, we have to (i) show that LoRaWAN 1.1 fulfills the structure of the protocol Π proved to be secure by Theorem 1, (ii) prove that the underlying protocol $P = P_{LoRaWAN}$ is 2-ACCE-secure, and (iii) choose a 2-ACCE-secure instantiation for the protocol $P' = P'_{LoRaWAN}$.

As described in Section 2.1, a typical LoRaWAN network involves four entities: ED, NS, JS, and AS. But only the first three are actually involved in the key exchange, and the channel establishment. Moreover, in actual deployments, AS is often co-localised with NS. That is, AS is in fact merely a functionality handled by NS, and the latter is given the four session keys K^e_a, K^e_c, $K^{i_1}_c$, $K^{i_2}_c$. Hence, we instantiate LoRaWAN accordingly: our protocol is made of three active entities (ED, NS, JS) which the different cryptographic operations are attributed to.

Since LoRaWAN is based on static symmetric keys, we define the long term key of each party to be $\mathsf{ltk} = (\mathsf{pk}, \mathsf{sk}, \mathsf{mk})$, made of (i) a private key sk, (ii) the corresponding certified public key pk, and (iii) a master symmetric key mk. If $P_k \in \mathcal{J}$, the three components of ltk are defined. Otherwise, $P_j.\mathsf{ltk} = (\mathsf{pk}, \mathsf{sk}, \perp)$ if $P_j \in \mathcal{N}$, and $P_i.\mathsf{ltk} = (\perp, \perp, \mathsf{mk})$ if $P_i \in \mathcal{E}$. Each party $P_i \in \mathcal{E}$ has a unique master key mk, shared with a party $P_k \in \mathcal{J}$.

4.1 2-party Protocol P in LoRaWAN 1.1 is 2-ACCE Secure

Theorem for $P_{LoRaWAN}$. Let $P_{LoRaWAN}$ correspond to the messages exchanged, and the operations done between a client (ED) and a server (NS-JS). Let $\mathsf{StAE}_{\mathsf{client}}$ (resp. $\mathsf{StAE}_{\mathsf{server}}$) be the AEAD function used by the client (resp. server) to encrypt and MAC the messages.

Theorem 2. *Under the assumption that* $\mathsf{StAE}_{\mathsf{client}}$ *and* $\mathsf{StAE}_{\mathsf{server}}$ *are* sAE*-secure,* $P_{LoRaWAN}$ *is a secure* 2-ACCE *protocol with the following reductions:*

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{P} \leq q \Big[& (n_{\mathsf{C}} + n_{\mathsf{S}}) \left(\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) + n_{\mathsf{C}} \left(\mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} \right) \\
& + n_{\mathsf{S}} \cdot \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + 2^{-\mu} \left(n_{\mathsf{C}} \cdot (1 - 2^{-\beta}) + n_{\mathsf{S}} \right) \Big] \\
\mathsf{adv}^{\mathsf{CS}}_{P} \leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} & \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P}
\end{aligned}$$

where q is the number of instances per party, n_{C} (resp. n_{S}) is the number of client (resp. server) parties, μ is the bit length of the MAC tag, and β is the bit length of the counter cnt_J.

Sketch Proof of Theorem 2. We consider the ACCE security model of Jager et al. [30], and define the entity authentication and the channel security experiments accordingly, but we forbid any corruption of the party (and its presumed partner) involved in the security experiments (the entity authentication game and the channel security game). That is LoRaWAN does not provide forward secrecy, nor protects against key-compromise impersonation attacks [13]. We give here only a sketch of proof of Theorem 2. The full proof is given in Appendix C.

As for the EA-security of $P_{LoRaWAN}$, we consider first a client (ED) adversary, and then a server (NS-JS) adversary.

Regarding a client adversary, we idealise each cryptographic function used to compute a Join Accept message: the KDF_{mk} function used to compute the MAC key (MK_3), the MAC function, and the encryption function AES. Being able to distinguish such changes corresponds respectively to the advantages $\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$, $\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}}$, and $\mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}}$. To that point, the ability of an adversary to forge a valid Join Accept message lies on the ability to provide a valid counter (probability at most $\frac{2^\beta-1}{2^\beta}$), and a valid MAC tag (probability $2^{-\mu}$) carried in the Join Accept message. Hence Pr[forgery Join Accept] $\leq p_{ja} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + 2^{-\mu}(1-2^{-\beta})$. Then the adversary is successful if the client and the server do not share the same sid (i.e., if they do not have a matching conversation). This is possible if the adversary succeeds in forging a valid RekeyConf message. We reduce this event to the security of the underlying AEAD function $\mathsf{StAE}_{\mathsf{server}}$ used to compute that message (corresponding to an advantage $\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}}$). Taking account of all possible client instances adds a factor $q \cdot n_{\mathsf{C}}$, where n_{C} is the number of client parties, and q the number of instances per party.

Therefore, the advantage of a client adversary in winning the EA experiment is bounded by

$$\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \leq q \cdot n_{\mathsf{C}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2^{-(\mu+\beta)}(2^\beta - 1) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right)$$

Regarding the server adversary, the reasoning is quite similar. First we idealise each cryptographic function used to compute a Join Request and a RekeyInd message: the MAC function used to compute the Join Request's MAC tag, and the KDF_c and KDF_a functions used to compute the session keys involved in the calculation of the RekeyInd message. Being able to distinguish these changes corresponds respectively to the advantages $\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}}$, and $2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$. To this point, the probability to forge a valid Join Request message corresponds to the probability to forge a valid MAC tag (that is $2^{-\mu}$). Hence Pr[forgery Join Request] $\leq p_{jr} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2^{-\mu}$. Finally, the only remaining possibility for the adversary is that client and server do not share the same sid (i.e., they do not have a matching conversation). This implies forging a valid RekeyInd message. We reduce this event to the security of the underlying AEAD function $\mathsf{StAE}_{\mathsf{client}}$ used to compute that message (corresponding to an advantage $\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}}$). Taking account of all possible server instances adds a factor $q \cdot n_{\mathsf{S}}$, where n_{S} is the number of server parties, and q the number of instances per party.

Therefore, the advantage of a server adversary in winning the EA experiment is bounded by

$$\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \leq q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right)$$

In addition, we have also

$$\Pr[\text{forgery Join Request}] \leq p_{jr} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2^{-\mu}$$
$$\Pr[\text{forgery Join Accept}] \leq p_{ja} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + 2^{-(\mu+\beta)} \cdot (2^{\beta} - 1)$$

Regarding the CS experiment, we first abort if there exists an instance of some client or server party that accepts maliciously, which adds an advantage $\mathsf{adv}^{\mathsf{EA}}_{P}$. Then we idealise the cryptographic functions used to compute the session keys K^e_a, K^e_c, $K^{i_1}_c$, and $K^{i_2}_c$. Being able to distinguish the change leads to an advantage $2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$. Finally we reduce the ability to win the CS experiment to the security of the underlying AEAD functions that are used to encrypt messages in either direction: $\mathsf{StAE}_{\mathsf{client}}$ and $\mathsf{StAE}_{\mathsf{server}}$. This corresponds to an advantage $\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}}$. Taking account of all possible instances adds a factor $q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}}$.

Therefore, the advantage of an adversary in winning the CS experiment is bounded by

$$\mathsf{adv}^{\mathsf{CS}}_{P} \leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P}$$

4.2 Meeting 3-ACCE Security

As seen in Section 2.2, and also exhibited by Theorem 2, the genuine LoRaWAN 1.1 protocol suffers from several flaws that forbid from concluding regarding its security. In particular, the (too) short size of several parameters (notably the size μ of the MAC output) provides useless security bounds in Theorem 2. Therefore, we modify LoRaWAN 1.1 the following way.

- We demand that the size μ of the MAC output be high enough so that the security bounds $\mathsf{adv}^{\mathsf{EA}}_{P}$ and $\mathsf{adv}^{\mathsf{CS}}_{P}$ be tight.
- We slightly change the behaviour of the JS as follows[14]: the JS verifies entirely the Join Request message (including the ED counter cnt_E), and the RekeyInd message. It sends the session keys sk to the NS only if the RekeyInd message is valid. This change aims at precluding an attack that allows the adversary to trivially win the EA experiment. Indeed, if JS does not verify the RekeyInd message, this means that it accepts as soon as it sends the Join Accept message. Yet, the JS has no guarantee that the ED successfully completes the protocol (it is enough for the adversary to drop or alter the Join Accept message in order for the ED to not accept).

[14] The components surrounded with brackets in Figure 7 depict these additional operations.

– The genuine LoRaWAN specification states that ED must send a RekeyInd message to NS as long as it does not receive a RekeyConf response. We demand that ED send only one message. Firstly in order to clearly separate the pre-accept and post-accept phases. Secondly, because sending multiple RekeyInd messages allows the adversary to trivially win the EA experiment. Indeed, the adversary has to merely forbid NS from receiving the first RekeyInd message, and this breaks the transcript equality. Finally, this change also reduces the surface of the "Encrypted message forgery" attack described in Section 2.2.
– We require that *all* entities implement version 1.1 (including NS) so that no fallback[15] to LoRaWAN 1.0 be possible (and the vulnerabilities of that version [4] be avoided).

Hence our adapted version of LoRaWAN 1.1 fulfills the structure of protocol Π, and the protocol $P_{LoRaWAN}$ is 2-ACCE-secure.

Now we define the companion security protocol $P'_{LoRaWAN}$ that is used between NS and JS. As explained in Section 2.3, the careful choice of this protocol is crucial to the overall security of a LoRaWAN network. Indeed, the theoretical attack described in Section 2.3 illustrates that choosing an unreliable protocol as $P'_{LoRaWAN}$ drastically weakens the security of LoRaWAN, independently of the security of the LoRaWAN cryptographic functions, and how well protected the master keys are. Therefore, we define the protocol $P'_{LoRaWAN}$ to be TLS 1.2 [20] in DHE, or RSA mode, with mutual authentication, and instantiated with AEAD encryption schemes such as AES-GCM, AES-CCM [40], or ChaCha20-Poly1305 [44]. TLS 1.2 is known to be 2-ACCE-secure [30, 32]. Alternatively, $P'_{LoRaWAN}$ can be defined as TLS 1.3 [45] in (EC)DHE mode, with mutual authentication. We recall that TLS 1.3 uses only AEAD encryption schemes. TLS 1.3 is proved to be 2-AKE-secure [23]. Although this result applies to an earlier draft of the protocol, we may reasonably assume that the final version also guarantees 2-AKE-security. Since AEAD encryption schemes are used, this implies 2-ACCE-security for TLS 1.3.

Combining all the above with Theorem 1, we obtain the 3-ACCE-security of our adapted version of LoRaWAN 1.1.

5 Conclusion

Using a provable security approach, we have provided the first analysis of LoRaWAN 1.1, a dedicated IoT protocol that aims at replacing the previous 1.0 version currently deployed worldwide. We have described several theoretical attacks against LoRaWAN 1.1, and enlightened, in particular, that the security of LoRaWAN 1.1 crucially depends on the companion security protocol (undefined by the specification) that is used between two of the parties. A theoretical attack illustrates that exploiting weaknesses of this additional protocol allows

[15] Falling back to version 1.0 is what an ED in version 1.1 *must* do, per specification, when it faces an NS in version 1.0 (see Section 2.2).

breaking LoRaWAN 1.1, independently of the security of the LoRaWAN core cryptographic functions. This also shows that analysing such a 3-party protocol requires a suitable security model that incorporates all its subtleties, and makes explicit the security requirements.

Consequently, we have extended the notion of 3-ACCE-security to provide a general framework that captures the security properties a 3-party protocol should guarantee, and allows assessing its security. We have described such a generic protocol provably secure in our model. Applying these results, we have proposed a slightly modified version of LoRaWAN 1.1 with stronger security properties, formally proved it to be secure in our security model, and described how to concretely instantiate it. This version implies to increase the size of several parameters, and to slightly change the behaviour of some entities involved in the protocol.

This work contributes to the field of the 3-party protocols and their security models. As the theoretical attacks we have exhibited against LoRaWAN 1.1 illustrate, these protocols require a careful cryptographic analysis against strong threat models. It will hopefully help analyse and better understand the security of multi-party protocols which reflect the growing complexity of the communications and interactions as the IoT arises.

References

1. Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., Watteyne, T.: Understanding the Limits of LoRaWAN. IEEE Communications Magazine **55**(9), 34–40 (September 2017)
2. Alt, S., Fouque, P.A., Macario-Rat, G., Onete, C., Richard, B.: A cryptographic analysis of UMTS/LTE AKA. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16: 14th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 9696, pp. 18–35. Springer, Heidelberg, Germany, Guildford, UK (Jun 19–22, 2016). https://doi.org/10.1007/978-3-319-39555-5_2
3. Avoine, G., Canard, S., Ferreira, L.: Symmetric-key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy. Cryptology ePrint Archive, Report 2019/444 (2019)
4. Avoine, G., Ferreira, L.: Rescuing LoRaWAN 1.0. In: Financial Cryptography and Data Security (FC 2018) (2018), https://fc18.ifca.ai/preproceedings/13.pdf
5. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: 38th Annual Symposium on Foundations of Computer Science. pp. 394–403. IEEE Computer Society Press, Miami Beach, Florida (Oct 19–22, 1997). https://doi.org/10.1109/SFCS.1997.646128
6. Bellare, M., Kohno, T., Namprempre, C.: Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In: Atluri, V. (ed.) ACM CCS 02: 9th Conference on Computer and Communications Security. pp. 1–11. ACM Press, Washington D.C., USA (Nov 18–22, 2002). https://doi.org/10.1145/586110.586112
7. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) Advances in Cryptology – ASIACRYPT 2000. Lecture Notes in Computer Science,

vol. 1976, pp. 531–545. Springer, Heidelberg, Germany, Kyoto, Japan (Dec 3–7, 2000). https://doi.org/10.1007/3-540-44448-3_41
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 139–155. Springer, Heidelberg, Germany, Bruges, Belgium (May 14–18, 2000). https://doi.org/10.1007/3-540-45539-6_11
9. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994). https://doi.org/10.1007/3-540-48329-2_21
10. Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331 (2004), http://eprint.iacr.org/2004/331
11. Bhargavan, K., Boureanu, I., Delignat-Lavaud, A., Fouque, P., Onete, C.: A Formal Treatment of Accountable Proxying over TLS. In: 2018 IEEE Symposium on Security and Privacy (SP). vol. 00, pp. 339–356. https://doi.org/10.1109/SP.2018.00021, doi.ieeecomputersociety.org/10.1109/SP.2018.00021
12. Bhargavan, K., Boureanu, I., Fouque, P.A., Onete, C., Richard, B.: Content delivery over TLS: a cryptographic analysis of keyless SSL. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 1–16. IEEE (April 2017). https://doi.org/10.1109/EuroSP.2017.52
13. Blake-Wilson, S., Johnson, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M. (ed.) 6th IMA International Conference on Cryptography and Coding. Lecture Notes in Computer Science, vol. 1355, pp. 30–45. Springer, Heidelberg, Germany, Cirencester, UK (Dec 17–19, 1997)
14. Butun, I., Pereira, N., Gidlund, M.: Security Risk Analysis of LoRaWAN and Future Directions. Future Internet **11**(1) (2018), http://www.mdpi.com/1999-5903/11/1/3
15. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 453–474. Springer, Heidelberg, Germany, Innsbruck, Austria (May 6–10, 2001). https://doi.org/10.1007/3-540-44987-6_28
16. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 143–161. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002). https://doi.org/10.1007/3-540-45708-9_10, http://eprint.iacr.org/2002/120/
17. Choo, K.K.R., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment protocols. In: Roy, B.K. (ed.) Advances in Cryptology – ASIACRYPT 2005. Lecture Notes in Computer Science, vol. 3788, pp. 585–604. Springer, Heidelberg, Germany, Chennai, India (Dec 4–8, 2005). https://doi.org/10.1007/11593447_32
18. Cremers, C.J.F.: Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) ACNS 09: 7th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 5536, pp. 20–33. Springer, Heidelberg, Germany, Paris-Rocquencourt, France (Jun 2–5, 2009). https://doi.org/10.1007/978-3-642-01957-9_2

19. Cremers, C.J.F.: Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In: Atluri, V., Díaz, C. (eds.) ESORICS 2011: 16th European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 6879, pp. 315–334. Springer, Heidelberg, Germany, Leuven, Belgium (Sep 12–14, 2011). https://doi.org/10.1007/978-3-642-23822-2_18
20. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol – Version 1.2. https://tools.ietf.org/html/rfc5246 (August 2008)
21. Dönmez, T.C.M., Nigussie, E.: Security of Join Procedure and its Delegation in LoRaWAN v1.1. In: FNC/MobiSPC. vol. 134, pp. 204–211 (2018), https://doi.org/10.1016/j.procs.2018.07.202
22. Dönmez, T.C.M., Nigussie, E.: Security of LoRaWAN v1.1 in Backward Compatibility Scenarios. In: Shakshuki, E., Yasar, A. (eds.) FNC/MobiSPC. vol. 134, pp. 51–58 (2018), https://doi.org/10.1016/j.procs.2018.07.143
23. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel:, C. (eds.) ACM CCS 15: 22nd Conference on Computer and Communications Security. pp. 1197–1210. ACM Press, Denver, CO, USA (Oct 12–16, 2015). https://doi.org/10.1145/2810103.2813653
24. Eldefrawy, M., Butun, I., Pereira, N., Gidlund, M.: Formal Security Analysis of LoRaWAN. Computer Networks **148**, 328–339 (November 2018), https://doi.org/10.1016/j.comnet.2018.11.017
25. Eronen, P., Tschofenig, H.: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS) (December 2005), RFC 4279
26. Ferguson, N., Schneier, B.: A Cryptographic Evaluation of IPsec (1999)
27. Fouque, P.A., Onete, C., Richard, B.: Achieving better privacy for the 3GPP AKA protocol. Cryptology ePrint Archive, Report 2016/480 (2016), http://eprint.iacr.org/2016/480
28. Han, J., Wang, J.: An Enhanced Key Management Scheme for LoRaWAN. In: Wang, G., Chen, J., Yang, L.T. (eds.) Security, Privacy, and Anonymity in Computation, Communication, and Storage. pp. 407–416. Springer International Publishing (2018)
29. Iwata, T., Kurosawa, K.: Stronger security bounds for OMAC, TMAC, and XCBC. In: Johansson, T., Maitra, S. (eds.) Progress in Cryptology - INDOCRYPT 2003: 4th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 2904, pp. 402–415. Springer, Heidelberg, Germany, New Delhi, India (Dec 8–10, 2003)
30. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. Cryptology ePrint Archive, Report 2011/219 (2011), http://eprint.iacr.org/2011/219
31. Kim, J., Song, J.: A Dual Key-Based Activation Scheme for Secure LoRaWAN. Wireless Communications and Mobile Computing **2017** (2017), https://doi.org/10.1155/2017/6590713
32. Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367 (2013), http://eprint.iacr.org/2013/367
33. Krawczyk, H.: SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 400–425. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003). https://doi.org/10.1007/978-3-540-45146-4_24

34. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 429–448. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40041-4_24
35. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007: 1st International Conference on Provable Security. Lecture Notes in Computer Science, vol. 4784, pp. 1–16. Springer, Heidelberg, Germany, Wollongong, Australia (Nov 1–2, 2007)
36. LoRa Alliance Technical Committe Regional Parameters Workgroup: LoRaWAN 1.1 Regional Parameters (January 2018), LoRa Alliance, rev. B
37. LoRa Alliance Technical committee: Technical Recommendations for Preventing State Synchronization Issues around LoRaWAN 1.0.x Join Procedure (August 2018), LoRa Alliance, version 1.0.0
38. Lundgren, L.: Taking over the world through MQTT – Aftermath. Black Hat USA (2017)
39. Mason, J., Watkins, K., Eisner, J., Stubblefield, A.: A natural language approach to automated cryptanalysis of two-time pads. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 06: 13th Conference on Computer and Communications Security. pp. 235–244. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006). https://doi.org/10.1145/1180405.1180435
40. McGrew, D.: An Interface and Algorithms for Authenticated Encryption. https://tools.ietf.org/html/rfc5116 (January 2008), RFC 5116
41. Morrissey, P., Smart, N.P., Warinschi, B.: A modular security analysis of the TLS handshake protocol. In: Pieprzyk, J. (ed.) Advances in Cryptology – ASIACRYPT 2008. Lecture Notes in Computer Science, vol. 5350, pp. 55–73. Springer, Heidelberg, Germany, Melbourne, Australia (Dec 7–11, 2008). https://doi.org/10.1007/978-3-540-89255-7_5
42. Naylor, D., Schomp, K., Varvello, M., Leontiadis, I., Blackburn, J., López, D.R., Papagiannaki, K., Rodriguez Rodriguez, P., Steenkiste, P.: Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. pp. 199–212. SIGCOMM '15, ACM (2015), http://doi.acm.org/10.1145/2785956.2787482
43. Naylor, D., Li, R., Gkantsidis, C., Karagiannis, T., Steenkiste, P.: And Then There Were More: Secure Communication for More Than Two Parties. In: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies. pp. 88–100. CoNEXT '17, ACM (2017), http://doi.acm.org/10.1145/3143361.3143383
44. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. https://tools.ietf.org/html/rfc7539 (May 2015), RFC 7539
45. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/rfc8446 (August 2018)
46. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-23. https://tools.ietf.org/html/draft-ietf-tls-tls13-23 (January 2018)
47. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 373–390. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_23
48. Rupprecht, D., Kohls, K., Holz, T., Pöpper, C.: Breaking LTE on layer two. In: 2019 IEEE Symposium on Security and Privacy (SP). vol. 00, pp. 91–106 (2019)

49. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), http://eprint.iacr.org/2004/332
50. Sornin, N.: LoRaWAN 1.1 Specification (October 2017), LoRa Alliance, version 1.1, 11/10/2017
51. Sornin, N., Luis, M., Eirich, T., Kramp, T.: LoRaWAN Specification (July 2016), LoRa Alliance, version 1.0
52. Wu, T.: The SRP Authentication and Key Exchange System (September 2000), RFC 2945
53. Yegin, A.: LoRaWAN Backend Interfaces 1.0 Specification (October 2017), LoRa Alliance, version 1.0, 11/10/2017

A Extended Security Proof for the 3-party Protocol Π

In this section, we give a proof of Theorem 1. We proceed through a sequence of games [10, 49] between a challenger and an adversary $\mathcal{A}$.

A.1 Entity Authentication

Let $\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{X}}$ be the probability that an X adversary succeeds, with $\mathsf{X} \in \{\mathsf{E}, \mathsf{N}, \mathsf{J}\}$, where E, N, J indicate respectively a party from $\mathcal{E}$, $\mathcal{N}$, $\mathcal{J}$. We have that $\mathsf{adv}^{\mathsf{EA}}_{\Pi} \leq \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{E}} + \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{N}} + \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{J}}$.

ED adversary. Let E_i be the event that the adversary succeeds in making an instance maliciously accept during Game i, where the instance is of a party $P_i \in \mathcal{E}$.

Game 0. This game corresponds to the EA-security game of the 3-party protocol Π described in Section 3.3 when the adversary targets an ED. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{E}}$$

Game 1. In this game, the challenger aborts the experiment if he does not guess which party $P_i \in \mathcal{E}$ the instance that will maliciously accept belongs to, and the corresponding partner-party $P_k \in \mathcal{J}$. Therefore

$$\Pr[E_1] = \Pr[E_0] \times \frac{1}{n_{\mathsf{E}} \cdot n_{\mathsf{J}_{\mathsf{E}}}}$$

where n_{E} is the number of parties in $\mathcal{E}$ and $n_{\mathsf{J}_{\mathsf{E}}} \leq n_{\mathsf{J}}$ is the number of parties in $\mathcal{J}$ that can be partnered with a party in $\mathcal{E}$.

Game 2. Now the party $P_i \in \mathcal{E}$ and its party partner $P_k \in \mathcal{J}$ are fixed. We want to rule out the event that there is no unique instance of some party in $\mathcal{N}$ that is partnered (i.e., shares the same session identifier sid) with some instance π_i^n of the party $P_i \in \mathcal{E}$.

If the adversary succeeds in forging valid Join Accept and RekeyConf messages, this implies (in particular) that there is no such instance of some party of $\mathcal{N}$ that is partnered with π_i^n (in addition, this implies that there is no instance of P_k that has computed the Join Accept message). Therefore we want to preclude such an event. Forging these messages corresponds to the advantage $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}}$ for a client adversary of breaking the EA-security of the protocol P. Note however that precluding such a forgery *does not* imply the existence of a unique instance $\pi_j^u \in P_j.\mathsf{Instances}$ for some party $P_j \in \mathcal{N}$ that is partnered with π_i^n. Indeed there is, in this execution of the 3-party protocol Π, other means to rule out the existence of such an instance. Nonetheless, ruling out the forgery of a Join Accept message implies necessarily the existence of an instance $\pi_k^\ell \in P_k.\mathsf{Instances}$ that computes that message. Therefore we have

$$\Pr[E_1] \leq \Pr[E_2] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}}$$

Game 3. Now we want to rule out the event that the adversary gets from P_k valid parameters with respect to P (Join Accept message, session keys sk) so that the adversary be able to reply to P_i without any party of $\mathcal{N}$ being involved.

But first we add an abort rule. In this game, the challenger aborts the experiment if she does not guess which party $P_j \in \mathcal{N}$ is partnered with P_k with respect to protocol P'. Therefore

$$\Pr[E_3] = \Pr[E_2] \times \frac{1}{n_{\mathsf{N}}}$$

Game 4. In this game, the challenger aborts the experiment if an adversary succeeds in impersonating P_j to P_k. This implies an adversary able to break the EA-security of P' when targeting the server side. Therefore

$$\Pr[E_3] \leq \Pr[E_4] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}$$

Game 5. So far, we have ruled out the non-existence of an instance $\pi_j^u \in P_j.\mathsf{Instances}$ that is involved in the execution of P with π_i^n. Therefore, π_i^n receives the Join Accept message sent by π_j^u upon reception of the Join Request message sent by π_i^n (recall that we have ruled out the forgery of such a message in Game 2). Therefore the only way to have $\pi_i^n.\mathsf{sid} \neq \pi_j^u.\mathsf{sid}$ is if π_i^n and π_j^u do not share the same RekeyInd or RekeyConf message. This implies either forging a RekeyInd message, or getting the suitable keys, transmitted by P_k to P_j, in order to compute a RekeyInd or a RekeyConf message. We can reduce these events to the channel security with respect to P on the one hand, and to the channel security with respect to P' on the other hand.

Therefore, in this game, the challenger aborts the experiment if π_j^u ever receives a valid RekeyInd message but π_i^n has not output that message, or if π_i^n receives a valid RekeyConf message but π_j^u has not computed it. We have

$$\Pr[E_4] \leq \Pr[E_5] + \mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'}$$

To this point, the execution of P between π_i^n and π_j^u is correct. Hence π_j^u is the unique instance such that $\pi_i^n.\mathsf{sid} = \pi_j^u.\mathsf{sid}$.

We also have that π_k^ℓ computes the Join Accept message, hence the existence of that instance. In addition, π_i^n uses the Join Accept message computed by π_k^ℓ, due to Game 2. Reciprocally, π_k^ℓ uses the Join Request message computed by π_i^n (because π_j^u receives correctly that message from π_i^n, and no impersonation of P_j to P_k takes place). Therefore both instances π_i^n and π_k^ℓ use the same inputs and the same permutation to compute the P session keys. Hence the output is equal. That is $\pi_k^\ell.\mathsf{km} = \pi_i^n.\mathsf{ck}$. Therefore, to that point, the adversary has no chance of winning the experiment. That is

$$\Pr[E_5] = 0$$

Collecting all probabilities from Game 0 to Game 5, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{E}} &= \Pr[E_0] \\
&= n_{\mathsf{E}} \times n_{\mathsf{J_E}} \times \Pr[E_1] \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right) \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \Pr[E_3] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(Pr[E_4] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(Pr[E_5] + \mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right) \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{J_E}} \left(n_{\mathsf{N}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right)
\end{aligned}$$

NS adversary. Let us consider the winning conditions (a) and (b) of an NS adversary. Let p_a (resp. p_b) the probability that the adversary wins through condition (a) (resp. condition (b)). We have that $\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{N}} \leq p_a + p_b$. We first consider a sequence of changes related to condition (a).

Let E_i^a be the event that the adversary succeeds in making an instance maliciously accept during Game^a i through condition (a), where the instance parent is in $\mathcal{N}$.

Game^a 0. This game corresponds to the EA-security game of the 3-party protocol Π described in Section 3.3 when the adversary targets NS, and tries to win through condition (a). Therefore we have that

$$\Pr[E_0^a] = p_a$$

Game^a 1. In this game, the challenger stops the experiment if she does not guess which party $P_j \in \mathcal{N}$ and its partner-party $P_i \in \mathcal{E}$ the adversary targets. Therefore

$$\Pr[E_1^a] = \Pr[E_0^a] \times \frac{1}{n_{\mathsf{E}} \cdot n_{\mathsf{N}}}$$

Game^a *2.* Now the parties $P_j \in \mathcal{N}$ and its partner-party $P_i \in \mathcal{E}$ are fixed.

In this game, the challenger aborts the experiment if the adversary succeeds in forging valid Join Request and RekeyInd messages, hence impersonates P_i to P_j. This event corresponds exactly to the event that an adversary against the EA-security of the protocol P wins when the server side is targeted. The advantage of such an event is $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$. Therefore

$$\Pr[E_1^a] \leq \Pr[E_2^a] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$$

Game^a *3.* So far the parties P_i and P_j are fixed. Moreover, due to Game^a 2, for any instance $\pi_j^u \in P_j.\mathsf{Instances}$ such that $\pi_j^u.\alpha = 1$ and $\pi_j^u.\mathsf{pid} = P_i$, there is an instance $\pi_i^n \in P_i.\mathsf{Instances}$ that is involved in the protocol Π (i.e., at least this instance computes a Join Request message) *under the assumption* that the run of protocol P' between P_j and some party in $\mathcal{J}$ that is the intended partner of P_i is executed honestly. However, precluding the adversary to break the EA-security of P when the server side is targeted *does not* imply in general the existence of such an instance $\pi_i^n \in P_i.\mathsf{Instances}$. Indeed, the only cryptographic operation that π_j^u does in order to accept is verifying the RekeyInd message with keys that it does not even compute but receives from some party in $\mathcal{J}$. Therefore, if the adversary, on the one hand, succeeds in sending keys sk of his choice to P_j, he can, on the other hand, provide a consistent RekeyInd message (computed under sk), bringing P_j to accept although no party in $\mathcal{E}$ is actually involved. This is possible either if the adversary impersonates to P_j some party in $\mathcal{J}$, or if the adversary forges a valid P' application message (carrying the keys sk chosen by her). We preclude such events in the subsequent sequence of games.

But, before considering this case, the challenger aborts the experiment if he does not guess which party $P_k \in \mathcal{J}$ is the intended partner of P_i (during the execution of protocol P). There is n_{J} parties in $\mathcal{J}$. However each party in $\mathcal{E}$ may communicate with a limited number of parties in $\mathcal{J}$. That number is $n_{\mathsf{J_E}} \leq n_{\mathsf{J}}$. Therefore we have that

$$\Pr[E_3^a] = \Pr[E_2^a] \times \frac{1}{n_{\mathsf{J_E}}}$$

Game^a *4.* In this game we want to preclude the impersonation of P_k to P_j. Therefore, the challenger aborts if an adversary succeeds in making P_j accept with P_k as its purported partner. Therefore

$$\Pr[E_3^a] \leq \Pr[E_4^a] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}}$$

Game^a *5.* Now we want to preclude the possibility for the adversary to forge a valid P' application message. This event can be reduced to the channel security with respect to P'. Therefore we have

$$\Pr[E_4^a] \leq \Pr[E_5^a] + \mathsf{adv}^{\mathsf{CS}}_{P'}$$

Game^a *6.* So far we have ruled out the possibility for the adversary to forge valid Join Request and RekeyInd messages. Therefore, if π_j^u does not receive these genuine messages from an instance π_i^n, this means that π_j^u accepts because it has been provided with the necessary material (i.e., some keys and a RekeyInd message consistent with these keys). However we have also ruled out the possibility for the adversary to make π_j^u accept such a material (in Game^a 5). Therefore π_i^n and π_j^u share these two messages.

Since no impersonation of P_k takes place (Game^a 4), there exists an instance $\pi_k^\ell \in P_k.\mathsf{Instances}$ that receives the Join Request message sent by π_i^n and forwarded by some instance $\pi_j^v \in P_j.\mathsf{Instances}$, and computes the Join Accept message. π_k^ℓ sends to π_j^v (hence to π_j^u) the Join Accept message and the P session keys. π_j^u correctly receives these data because we have ruled out an impersonation of P_k to P_j, and a forgery of a P' application message intended to P_j. Therefore $\pi_j^u.\mathsf{ck} = \pi_k^\ell.\mathsf{km}$.

If π_i^n does not use the same Join Accept message, it computes different P session keys (because the key derivation function is a permutation). These keys are used by π_i^n to compute the RekeyInd message it sends. Since π_j^u accepts by assumption, this implies that either the adversary provides to π_j^u some alternative valid RekeyInd message, or the RekeyInd message computed by π_i^n with different keys is correctly verified by π_j^u using other keys. But this is ruled out in Game^a 2. Therefore, π_i^n receives necessarily the Join Accept message sent by π_k^ℓ. Since the Join Accept message is shared (in addition to the Join Request and RekeyInd messages) by π_i^n and π_j^u, the only way to have that $\pi_i^n.\mathsf{sid} \neq \pi_j^u.\mathsf{sid}$ is if the RekeyConf message sent by π_j^u is not the one received by π_i^n. This implies either a forgery of such a message, or the ability of an adversary to get the keys used to compute a RekeyConf message (that is the session keys sk sent by π_k^ℓ to π_j^v (hence to π_j^u) through the secure channel provided by P').

Therefore, in this game, the challenger aborts the experiment if either event happens. We can reduce the first event to the channel security with respect to P, and the second event to the channel security with respect to P'. Hence

$$\Pr[E_5^a] \leq \Pr[E_6^a] + \mathsf{adv}_P^{\mathsf{CS}} + \mathsf{adv}_{P'}^{\mathsf{CS}}$$

To this point, we have that $\pi_i^n.\mathsf{sid} = \pi_j^u\mathsf{sid}$. Moreover, due to the EA-security of P, π_i^n is the unique instance that shares the same sid with π_j^u. Therefore, the adversary has no chance of winning the experiment through condition (a). That is

$$\Pr[E_6^a] = 0$$

Collecting all the probabilities from Game^a 0 to Game^a 6, we have that

$$\begin{aligned}
p_a &= \Pr[E_0^a] \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot \Pr[E_1^a] \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(\Pr[E_2^a] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \Pr[E_3^a] + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\Pr[E_4^a] + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\Pr[E_5^a] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\Pr[E_6^a] + \mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right)
\end{aligned}$$

Now let E_i^b be the event that the adversary succeeds in making an instance accept maliciously during Game^b i through condition (b), where the parent instance is in $\mathcal{N}$.

Game^b 0. This game corresponds to the EA-security game of the 3-party protocol Π described in Section 3.3 when the adversary targets NS, and tries to win through condition (b). Therefore we have that

$$\Pr[E_0^b] = p_b$$

Game^b 1. In this game, the challenger aborts the experiment if he does not guess which party $P_j \in \mathcal{N}$ and partner-party $P_k \in \mathcal{J}$ the adversary targets. Therefore

$$\Pr[E_1^b] = \Pr[E_0^b] \times \frac{1}{n_{\mathsf{N}} \cdot n_{\mathsf{J}}}$$

Game^b 2. Now the parties $P_j \in \mathcal{N}$ and $P_k \in \mathcal{J}$ are fixed.

The only cryptographic operation that π_j^v does in order to accept is verifying the RekeyInd message with keys sk that it does not compute but receives allegedly from P_k. Therefore, if the adversary, on the one hand, succeeds in sending to π_j^v keys sk of her choice, she can, on the other hand, provide a consistent RekeyInd message (computed under sk), bringing π_j^v to accept although no party in $\mathcal{J}$ (neither in $\mathcal{E}$) is actually involved. This is possible either if the adversary impersonates P_k to P_j, or if the adversary forges a valid P' application message (intended to P_j). We can reduce both events to the channel security with respect to P'. Therefore we have

$$\Pr[E_1^b] \leq \Pr[E_2^b] + \mathsf{adv}^{\mathsf{CS}}_{P'}$$

Game^b 3. In this game we want to preclude the impersonation of P_k to P_j. Therefore, the challenger aborts if an adversary succeeds in making P_j accept with P_k as its purported partner. Therefore

$$\Pr[E_2^b] \leq \Pr[E_3^b] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}}$$

Game^b *4.* Since no impersonation of P_k to P_j takes place, there is an instance $\pi_k^\ell \in P_k.\mathsf{Instances}$ that computes the Join Accept message and the P session keys sk upon reception of the Join Request message forwarded by π_j^v. Moreover $\pi_j^v.\alpha = 1$ and we have ruled out the forgery of a P' application message (intended to P_j). Therefore, necessarily π_j^v receives a Join Accept message and session keys sk, and these messages are computed by π_k^ℓ.

Now, the only reason why the transcript of the messages exchanged between π_j^v and π_k^ℓ may differ (i.e., $\pi_j^v.\mathsf{sid} \neq \pi_k^\ell.\mathsf{sid}$) is that π_k^ℓ receives a P' application message (carrying a Join Request or a RekeyInd message) different than the one sent by π_j^v. This implies the ability to forge a valid P' application message intended to π_k^ℓ (i.e., with *different* P' session keys than the ones used in the opposite direction). Hence, in this game, the challenger aborts the experiment if π_k^ℓ ever receives a valid P' application message but that message is not computed by π_j^v. We can reduce this event to the channel security with respect to P'. We have

$$\Pr[E_3^b] \leq \Pr[E_4^b] + \mathsf{adv}_{P'}^{\mathsf{CS}}$$

So far, we have shown that for any instance $\pi_j^v \in P_j.\mathsf{Instances}$ such that $\pi_j^v.\alpha = 1$ and $\pi_j^v.\mathsf{pid} = P_k$, there exists an instance $\pi_k^\ell \in P_k.\mathsf{Instances}$ such that $\pi_k^\ell.\mathsf{sid} = \pi_j^v.\mathsf{sid}$. Moreover, the EA-security of P' implies that π_k^ℓ is the unique instance of P_k that shares with π_j^v the P' handshake messages. This guarantees that the whole transcript of the P' messages is shared only by π_j^v and π_k^ℓ. That is, π_k^ℓ is the unique instance of P_k such that $\pi_j^v.\mathsf{sid} = \pi_k^\ell.\mathsf{sid}$.

Game^b *5.* In this game, the challenger aborts the experiment if she does not guess which party $P_i \in \mathcal{E}$ presumably computes the Join Request received by P_j. Therefore

$$\Pr[E_5^b] = \Pr[E_4^b] \times \frac{1}{n_{\mathsf{E_J}}}$$

where $n_{\mathsf{E_J}} \leq n_{\mathsf{E}}$ is the number of EDs that can be partnered with a given JS.

Game^b *6.* In this game, the challenger aborts the experiment if P_j ever receives a valid Join Request message but no instance of P_i has computed that message. Therefore

$$\Pr[E_5^b] \leq \Pr[E_6^b] + \Pr[\text{forgery of Join Request}] \leq \Pr[E_6^b] + p_{jr}$$

Game^b *7.* So far there is an instance $\pi_i^n \in P_i.\mathsf{Instances}$ that computes the Join Request message received by P_j. Due to Game^b 2, this message is received by π_k^ℓ.[16] Therefore both instances π_i^n and π_k^ℓ use the same Join Request message (and the same derivation function) in order to compute the P session keys. Furthermore, the P session keys computed by π_k^ℓ are received by π_j^v (because a forgery of a P' application message intended to P_j has been ruled out in Game^b 2). That is $\pi_j^v.\mathsf{km} = \pi_k^\ell.\mathsf{km}$. The only reason why π_i^n and π_k^ℓ would not

[16] P_k is identified in this Join Request message. That is $\pi_i^n.\mathsf{pid} = P_k$.

compute the same P session keys is if they do not use the same Join Accept message.

Therefore, in this game, the challenger aborts the experiment if π_i^n ever receives a valid Join Accept message but P_j has not sent it. Therefore we have

$$\Pr[E_6^b] \leq \Pr[E_7^b] + \Pr[\text{forgery of Join Accept}] \leq \Pr[E_7^b] + p_{ja}$$

To this point, π_i^n and π_k^ℓ use the same Join Request and Join Accept messages, and the same permutation to compute the P session keys. Hence $\pi_i^n.\mathsf{ck} = \pi_k^\ell.\mathsf{km} = \pi_j^v.\mathsf{km}$. Therefore, the adversary has no chance of winning the experiment through condition (b). That is

$$\Pr[E_7^b] = 0$$

Collecting all the probabilities from Game^b 0 to Game^b 7, we have that

$$\begin{aligned}
p_b &= \Pr[E_0^b] \\
&= n_{\mathsf{N}} \cdot n_{\mathsf{J}} \cdot \Pr[E_1^b] \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_2^b] + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_3^b] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\Pr[E_4^b] + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&= n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \cdot \Pr[E_5^b] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(\Pr[E_6^b] + p_{jr} \right) + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\leq n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(\Pr[E_7^b] + p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&= n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right)
\end{aligned}$$

Therefore we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{N}} &\leq p_a + p_b \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J_E}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E_J}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} + (n_{\mathsf{E}} + 1) \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \cdot \left(\mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\quad + n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(p_{jr} + p_{ja} + \mathsf{adv}^{\mathsf{CS}}_{P} \right)
\end{aligned}$$

because $n_{\mathsf{E_J}} \leq n_{\mathsf{E}}$, and $n_{\mathsf{J_E}} \leq n_{\mathsf{J}}$.

JS adversary. Let E_i be the event that the adversary succeeds in making an instance accept maliciously during Game i, where the instance parent is in $\mathcal{J}$.

Game 0. This game corresponds to the EA-security game of the 3-party protocol Π described in Section 3.3 when the adversary targets JS. Therefore we have that

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{J}}$$

Game 1. In this game, the challenger aborts the experiment if he does not guess which party $P_k \in \mathcal{J}$ the instance that will maliciously accept belongs to, and the corresponding partner-party $P_j \in \mathcal{N}$. Therefore

$$\Pr[E_1] = \Pr[E_0] \times \frac{1}{n_{\mathsf{J}} \cdot n_{\mathsf{N}}}$$

Game 2. Now the party $P_k \in \mathcal{J}$ and its partner-party $P_j \in \mathcal{N}$ are fixed. We want to rule out the event that there is no unique instance of P_j that is partnered (i.e., shares the same session identifier sid) with any instance π^{ℓ}_k of P_k that ends in accepting state.

The non-existence of an instance $\pi^{v}_j \in P_j.\mathsf{Instances}$ that is presumably partnered with π^{ℓ}_k implies that a server adversary successfully breaks the EA-security of P'. Therefore, in this game, the challenger aborts the experiment if the adversary succeeds in breaking the EA-security of P' when the server side is targeted. Hence we have that

$$\Pr[E_1] \leq \Pr[E_2] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}}$$

Game 3. So far, the EA-security of P' ensures that π^{v}_j is the unique instance to share the handshake messages related to P' exchanged with π^{ℓ}_k. However, in order for π^{ℓ}_k to accept π^{ℓ}_k has to receive in addition valid Join Request and RekeyInd messages (carried in P' application messages). In turn, π^{ℓ}_k sends P' application messages carrying a Join Accept message and session keys sk. This implies necessarily that π^{v}_j is the unique instance such that $\pi^{\ell}_k.\mathsf{sid} = \pi^{v}_j.\mathsf{sid}$ unless one of these P' application messages is forged by an adversary.

Therefore, in this game, the challenger aborts the experiment if the adversary succeeds in forging P' application messages. We reduce this (in)ability to the channel security with respect to P'. Therefore

$$\Pr[E_2] \leq \Pr[E_3] + \mathsf{adv}^{\mathsf{CS}}_{P'}$$

Game 4. Now, for each instance $\pi^{\ell}_k \in P_k.\mathsf{Instances}$ such that $\pi^{\ell}_k.\alpha = 1$, there is a unique instance $\pi^{v}_j \in P_j.\mathsf{Instances}$ such that $\pi^{\ell}_k.\mathsf{sid} = \pi^{v}_j.\mathsf{sid}$.

In this game, the challenger aborts the experiment if she does not guess which party $P_i \in \mathcal{E}$ has presumably triggered the execution of protocol P (i.e., has computed the Join Request and RekeyInd messages intended to P_k). Therefore

$$\Pr[E_4] = \Pr[E_3] \times \frac{1}{n_{\mathsf{E_J}}}$$

Game 5. Now the party $P_i \in \mathcal{E}$ that presumably computes the messages with respect to protocol P is fixed.

The non-existence of an instance $\pi_i^n \in P_i.\mathsf{Instances}$ that computes the Join Request message forwarded by π_j^v to π_k^ℓ implies a forgery. Therefore, in this game, the challenger aborts the experiment if π_k^ℓ receives a valid Join Request message but no instance of P_i has output that message. Therefore

$$\Pr[E_4] \leq \Pr[E_5] + \Pr[\text{forgery of Join Request}] \leq \Pr[E_5] + p_{jr}$$

Game 6. To this point the adversary is unable to forge a Join Request message (Game 5) and cannot impersonate P_j to P_k (Game 2). This implies that π_k^ℓ uses necessarily the Join Request message computed by π_i^n. Hence $\pi_i^n.\mathsf{ck} \neq \pi_k^\ell.\mathsf{km}$ necessarily implies that π_i^n and π_k^ℓ do not use the same Join Accept message.

Hence, in this game, the challenger aborts the experiment if π_i^n ever receives a Join Accept message but π_k^ℓ has not computed such a message. Therefore we have

$$\Pr[E_5] \leq \Pr[E_6] + \Pr[\text{forgery of Join Accept}] \leq \Pr[E_6] + p_{ja}$$

To this point, if π_i^n verifies correctly the Join Accept message it receives, it holds necessarily that this message is computed by π_k^ℓ upon reception of a Join Request presumably sent by π_i^n. Moreover we have also ruled out the event of a Join Request forgery. Therefore the only way π_i^n verifies correctly the Join Accept it receives is if that message is computed by π_k^ℓ upon reception of the Join Request message sent by π_i^n. Therefore, we have necessarily that $\pi_k^\ell.\mathsf{km} = \pi_i^n.\mathsf{ck}$ (i.e., both instances compute the same P session keys because they use the same inputs, and the same derivation function).

Hence, up to this point, the adversary has no chance of winning the experiment. That is

$$\Pr[E_6] = 0$$

Collecting all the probabilities from Game 0 to Game 6, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{J}} &= \Pr[E_0] \\
&= n_{\mathsf{J}} \cdot n_{\mathsf{N}} \cdot \Pr[E_1] \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(\Pr[E_3] + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&= n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \cdot \Pr[E_4] + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(\Pr[E_5] + p_{jr} \right) + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&\leq n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(\Pr[E_6] + p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&= n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right)
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{\Pi} &\leq \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{E}} + \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{N}} + \mathsf{adv}^{\mathsf{EA}}_{\Pi,\mathsf{J}} \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{J}_{\mathsf{E}}} \left(n_{\mathsf{N}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} \right) \\
&\quad + n_{\mathsf{E}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{J}_{\mathsf{E}}} \cdot \left(\mathsf{adv}^{\mathsf{CS}}_{P} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} \right) + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + 2\mathsf{adv}^{\mathsf{CS}}_{P'} \right) \\
&\quad + n_{\mathsf{J}} \cdot n_{\mathsf{N}} \left(n_{\mathsf{E}_{\mathsf{J}}} \left(p_{jr} + p_{ja} \right) + \mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(2\mathsf{adv}^{\mathsf{CS}}_{P} + 3\mathsf{adv}^{\mathsf{CS}}_{P'} + 2p_{jr} + 2p_{ja} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right) \\
&\quad + n_{\mathsf{E}} \left(n_{\mathsf{J}} \cdot \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} + n_{\mathsf{N}} \cdot \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \right) \\
&\quad + n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(3\mathsf{adv}^{\mathsf{CS}}_{P'} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P',\mathsf{server}} \right)
\end{aligned}$$

because $n_{\mathsf{E}_{\mathsf{J}}} \leq n_{\mathsf{E}}$, and $n_{\mathsf{J}_{\mathsf{E}}} \leq n_{\mathsf{J}}$.

A.2 Channel Security

Let $\mathsf{adv}^{\mathsf{CS}}_{\Pi}$ be the advantage of the adversary in winning the channel security experiment. Let E_i be the event that the adversary wins in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$.

Game 0. This game corresponds to the channel security game described in Section 3.3. Therefore

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}_0 = \frac{1}{2} + \mathsf{adv}^{\mathsf{CS}}_{\Pi}$$

Game 1. In this game, the challenger proceeds as in the previous game but aborts and chooses a bit b uniformly at random if there exists an oracle of some party in $\mathcal{E} \cup \mathcal{N} \cup \mathcal{J}$ that accepts maliciously. In other words, in this game we make the same modifications as in the games performed during the entity authentication proof. Hence we have

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{EA}}_{\Pi}$$

Game 2. In this game, the challenger aborts the experiment if she does not guess the three parties involved in the session. Therefore

$$\mathsf{adv}_2 \geq \mathsf{adv}_1 \times \frac{1}{n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}}}$$

because $n_{\mathsf{J}_{\mathsf{E}}} \leq n_{\mathsf{J}}$, and $n_{\mathsf{E}_{\mathsf{J}}} \leq n_{\mathsf{E}}$.

Let π^n_i, π^u_j, π^v_j, and π^ℓ_k be the four instances sharing the same bid, with $\pi^n_i.\mathsf{parent} \in \mathcal{E}$, $\pi^u_j.\mathsf{parent} = \pi^v_j.\mathsf{parent} = P_j \in \mathcal{N}$, and $\pi^\ell_k.\mathsf{parent} = P_k \in \mathcal{J}$, such that, in the one hand, π^n_i and π^u_j are partnered ($\pi^n_i.\mathsf{sid} = \pi^u_j.\mathsf{sid}$), and, in the other hand, π^v_j and π^ℓ_k are partnered ($\pi^v_j.\mathsf{sid} = \pi^\ell_k.\mathsf{sid}$).

Game 3. In this game, the challenger aborts the experiment if the adversary is able to find $\pi_j^v.\mathsf{b}$ or $\pi_k^\ell.\mathsf{b}$ (that is if the adversary wins the experiment when targeting the NS-JS link). We can reduce such an event to the channel security with respect to P'. Therefore

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \mathsf{adv}_{P'}^{\mathsf{CS}}$$

Game 4. Now, the adversary can try to target π_i^n or π_j^u (i.e., the ED-NS link). In order to be successful, the adversary can first try to get the P session keys (sk) sent by P_k to P_j (the parent of π_j^v) through the secure channel provided by the protocol P'. We can reduce this possibility to the channel security with respect to P'. Therefore

$$\mathsf{adv}_3 \leq \mathsf{adv}_4 + \mathsf{adv}_{P'}^{\mathsf{CS}}$$

Game 5. Now the only remaining possibility in order for the adversary to be successful is breaking the channel security with respect to P. The inability of an adversary in breaking the channel security with respect to P relies implicitly on the inability of such an adversary in distinguishing the corresponding session keys sk from random. These session keys are also sent by P_k to P_j through the secure channel provided by P'. That channel guarantees real-from-random indistinguishability for the plaintexts. Indeed the security of this channel relies upon the underlying encryption function. The latter guarantees left-or-right security when keyed with the session keys, and the left-or-right security notion is equivalent to the real-from-random notion [5]. Hence the real-from-random indistinguishability for the plaintexts with respect to P'.

Therefore, in this game we add an abort rule. The challenger aborts the experiment if an adversary succeeds in distinguishing plaintexts (sent through the channel provided by P') from random. Therefore

$$\mathsf{adv}_4 \leq \mathsf{adv}_5 + \mathsf{adv}_{P'}^{\mathsf{CS}}$$

Now the only possibility for the adversary to be successful is to break the CS-security with respect to P. That is

$$\mathsf{adv}_5 \leq \mathsf{adv}_P^{\mathsf{CS}}$$

Collecting all the probabilities from Game 0 to Game 5, we have that

$$\begin{aligned}
\mathsf{adv}_{\Pi}^{\mathsf{CS}} &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}_{\Pi}^{\mathsf{EA}} \\
&= n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \cdot \mathsf{adv}_2 + \mathsf{adv}_{\Pi}^{\mathsf{EA}} \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}_3 + \mathsf{adv}_{P'}^{\mathsf{CS}} \right) + \mathsf{adv}_{\Pi}^{\mathsf{EA}} \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}_4 + 2\mathsf{adv}_{P'}^{\mathsf{CS}} \right) + \mathsf{adv}_{\Pi}^{\mathsf{EA}} \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}_5 + 3\mathsf{adv}_{P'}^{\mathsf{CS}} \right) + \mathsf{adv}_{\Pi}^{\mathsf{EA}} \\
&\leq n_{\mathsf{E}} \cdot n_{\mathsf{N}} \cdot n_{\mathsf{J}} \left(\mathsf{adv}_P^{\mathsf{CS}} + 3\mathsf{adv}_{P'}^{\mathsf{CS}} \right) + \mathsf{adv}_{\Pi}^{\mathsf{EA}}
\end{aligned}$$

B Extended Security Proof for $P_{LoRaWAN}$ in LoRaWAN 1.1

In this section, we give a proof of Theorem 2. We consider the ACCE security model [30, 32], and define the entity authentication and the channel security experiments accordingly.

Entity Authentication. Let $\mathsf{adv}^{\mathsf{EA}}_{P}$ be the probability that the adversary wins the entity authentication game. Let $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}}$ bounds the probability that a client (ED) adversary succeeds, and $\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$ bounds the probability that a server (NS-JS) adversary succeeds. We have that $\mathsf{adv}^{\mathsf{EA}}_{P} \leq \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$.

Client adversary. Let E_i be the event that the adversary succeeds in making a client instance accept maliciously during Game i.

Game 0. This game corresponds to the EA game of the 2-party protocol P when the client side is targeted. Thus we have

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}}$$

Two different clients (EDs) cannot share the same transcript because they will at least differ with the client identifiers. On the client side, each session is individualised with the parameters id_E, id_J, cnt_E (they appear in clear in the first message sent by the client). On the server side, each session is individualised with the parameters id_J, id_E, cnt_J. The Join Accept message sent by a server instance cannot repeat unless a collision appears in the function $\mathsf{AES}^{-1}(MK_1, \cdot)$. An adversary can then try to forge a valid Join Accept message. This will be handled in the successive experiments described below.

Game 1. In this game, the challenger tries to guess which client instance π_i^s will be the first instance to accept maliciously. If the guess is wrong, then the game is aborted. Hence

$$\Pr[E_1] = \Pr[E_0] \times \frac{1}{q \cdot n_{\mathsf{C}}}$$

where n_{C} is the number of client parties, and q the number of instances per party.

Game 2. In this game we replace the KDF_{mk} function used by π_i^s to compute the key MK_3 with a random function $\mathsf{F}^{\mathsf{KDF}_{mk}}_{MK_1}$. We do the same for any server instance that uses the KDF_{mk} function with the same master key MK_1 as π_i^s in order to compute MK_3. We use the fact that the master key MK_1 is uniformly drawn at random. Therefore distinguishing Game 1 and Game 2 implies an algorithm able to distinguish the KDF_{mk} function from a random function. Hence

$$\Pr[E_1] - \Pr[E_2] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{KDF}_{mk}} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$$

Game 3. In this game we replace the MAC function used by π_i^s to verify the Join Accept message (verification of τ_J) with a random function $\mathsf{F}^{\mathsf{MAC}}_{MK_3}$. We do the same for any server instance that uses the MAC function with the same master key MK_3 as π_i^s in order to compute τ_J. Since $MK_3 \leftarrow \mathsf{F}^{\mathsf{KDF}_{mk}}_{MK_1}(id_E)$, we use the fact that the key MK_3 is uniformly drawn at random. Therefore distinguishing Game 2 and Game 3 implies an algorithm able to distinguish the MAC function from a random function. Hence

$$\Pr[E_2] - \Pr[E_3] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}}$$

Game 4. The server instance computes the Join Accept message as Join Accept $= \mathsf{AES}^{-1}(MK_1, cnt_J \| id_N \| prms \| \tau_J)$. In this game we replace the AES decryption function used by the server instance with a random permutation Perm_{MK_1}. Moreover if a client instance uses the same master key MK_1 to "decrypt" the Join Accept message, then we replace the AES encryption function with the inverse permutation $\mathsf{Perm}^{-1}_{MK_1}$. We use the fact that MK_1 is uniformly drawn at random. Therefore distinguishing Game 3 and Game 4 implies an algorithm able to distinguish $\mathsf{AES}^{-1}(MK_1, \cdot)$ from a random permutation. Hence

$$\Pr[E_3] - \Pr[E_4] \leq \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}}$$

Game 5. In this game, we want to ensure that the client instance π_i^s receives exactly the Join Accept message computed by some other uncorrupted instance that has received the first message sent by π_i^s. Therefore, we add an abort rule. The challenger aborts the experiment if π_i^s ever receives a Join Accept message but no server instance having a matching conversation to π_i^s has output that message. After receiving the Join Accept message the client makes two verifications: τ_J and cnt_J. In order to accept the message as valid, both verifications must be correct. Let us assume that the adversary be able to compute Join Accept $\leftarrow \mathsf{Perm}_{MK_1}(cnt_J \| id_N \| prms \| \tilde{\tau})$ for some value $\tilde{\tau}$ and a correct value cnt_J. Due to Game 3, $\tau_J \leftarrow \mathsf{F}^{\mathsf{MAC}}_{MK_3}(id_J \| cnt_E \| cnt_J \| id_N \| prms)$ is computed by the client by evaluating a truly random function that is only accessible to the client instance and to the server instance knowing which master key MK_3 to use. Therefore the probability of the adversary to provide a correct value $\tilde{\tau}$ (i.e., the probability that $\tilde{\tau} = \tau_J$) is at most $2^{-\mu}$. Moreover the adversary provides some value as the Join Accept message (and not $\mathsf{Perm}_{MK_1}(cnt_J \| id_N \| prms \| \tilde{\tau})$). Hence the probability that $\tilde{\tau}$ is valid when the adversary picks a Join Accept message at random is not greater than $2^{-\mu}$.

Due to Game 4, the client instance computes from $\mathsf{Perm}^{-1}_{MK_1}(\text{Join Accept})$ some value cnt_J, that is by evaluating a truly random permutation that is only accessible to the client instance and to the server instance knowing which master key MK_1 to use. Let β be the bit length of the cnt_J parameter: there are 2^β possible values for cnt_J. Each new session triggers a new value cnt_J. Therefore the number of remaining correct values for cnt_J is $2^\beta - u \leq 2^\beta - 1$ at the u-th session. Hence the probability that cnt_J is correct is at most $(2^\beta - 1)/2^\beta$ at any session. Since both conditions (correctness of $\tilde{\tau}$ and cnt_J) have to be verified, we

have that

$$\Pr[E4] - \Pr[E_5] \leq 2^{-\mu} \times \frac{2^{\beta} - 1}{2^{\beta}}$$

Game 6. In this game we replace the KDF_a function used by π_i^s to compute K_a^e with a random function $\mathsf{F}^{\mathsf{KDF}_a}_{MK_2}$. We do the same for any server instance that uses the KDF_a function with the same master key MK_2 as π_i^s in order to compute K_a^e. We use the fact that the master key MK_2 is uniformly drawn at random. Therefore distinguishing Game 5 and Game 6 implies an algorithm able to distinguish the KDF_a function from a random function. Hence

$$\Pr[E_5] - \Pr[E_6] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{KDF}_a} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$$

Game 7. So far, the only possibility for the adversary to win is forging a RekeyConf message so that the two instances do not share the same sid. Therefore we add an abort rule. In this game, the challenger aborts if the client instance π_i^s ever receives a valid message RekeyConf but there exists no server instance having a matching conversation to π_i^s (i.e., sharing the same transcript of exchanged messages so far) that has output that message.

The keys K_c^e, $K_c^{i_2}$, and (optionally) K_a^e are used to compute the RekeyConf message. K_c^e, $K_c^{i_2}$ are output by the KDF_c function, and K_a^e is output by KDF_a. In Game 2, the $\mathsf{KDF}_c = \mathsf{KDF}_{mk} = \mathsf{AES}(MK_1, \cdot)$ function has been replaced with a truly random function $\mathsf{F}^{\mathsf{KDF}_{mk}}_{MK_1}$ that is only accessible to the client instance and to the server instance knowing which master key MK_1 to use. In Game 6, the $\mathsf{KDF}_a = \mathsf{AES}(MK_2, \cdot)$ function has been replaced with a truly random function $\mathsf{F}^{\mathsf{KDF}_a}_{MK_2}$ that is only accessible to the client instance and to the server instance knowing which master key MK_2 to use. Therefore, the keys K_c^e, $K_c^{i_2}$, and K_a^e are uniformly drawn at random. Hence, we can reduce the forgery of a RekeyConf message to the sAE-security of the $\mathsf{StAE}_{\mathsf{server}}$ function used to compute that message. Therefore

$$\Pr[E_6] - \Pr[E_7] \leq \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}}$$

To that point the only way for an adversary to make π_i^s accept maliciously is to send a RekeyConf message different from all the messages sent by all the server instances, such that RekeyConf is valid. However, in such a case the challenger aborts. Therefore

$$\Pr[E_7] = 0$$

Collecting all probabilities from Game 0 to Game 7, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} &= \Pr[E_0] \\
&= q \cdot n_{\mathsf{C}} \cdot \Pr[E_1] \\
&\leq q \cdot n_{\mathsf{C}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\leq q \cdot n_{\mathsf{C}} \left(\Pr[E_3] + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\leq q \cdot n_{\mathsf{C}} \left(\Pr[E_4] + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\leq q \cdot n_{\mathsf{C}} \left(\Pr[E_5] + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_6] + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \\
&\qquad + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \Big) \\
&\leq q \cdot n_{\mathsf{C}} \Big(\Pr[E_7] + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \\
&\qquad + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \Big) \\
&= q \cdot n_{\mathsf{C}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right)
\end{aligned}$$

Server adversary. Let E_i be the event that the adversary succeeds in making an server instance accept maliciously during Game i.

Game 0. This game corresponds to the EA game of the 2-party protocol P when the server side is targeted. Thus we have

$$\Pr[E_0] = \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}}$$

Game 1. In this game, the challenger tries to guess which server instance π^t_j will be the first instance to accept maliciously. If the guess is wrong, then the game is aborted. Hence

$$\Pr[E_1] = \Pr[E_0] \times \frac{1}{q \cdot n_{\mathsf{S}}}$$

where n_{S} is the number of server parties, and q the number of instances per party.

Game 2. In this game we replace the MAC function used by the server instance π^t_j to verify the first message from the client instance (verification of τ_E) with a random function $\mathsf{F}^{\mathsf{MAC}}_{MK_1}$. We do the same for any client instance that uses the MAC function with the same master key MK_1 as π^t_j in order to compute τ_E. We use the fact that the key MK_1 is uniformly drawn at random. Therefore distinguishing Game 1 and Game 2 implies an algorithm able to distinguish the MAC function from a random function. Hence

$$\Pr[E_1] - \Pr[E_2] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}}$$

Game 3. In this game, the challenger aborts if the server instance π_j^t ever receives a valid message Join Request $= id_J \| id_E \| cnt_E \| \tau_E$ but there is no client instance that has output the message. Due to Game 2, $\tau_E \leftarrow \mathsf{F}^{\mathsf{MAC}}_{MK_1}(id_J \| id_E \| cnt_E)$ is computed by evaluating a truly random function that is only accessible to the server instance and the client instance knowing the key MK_1 to use. Therefore the probability of the adversary to provide a correct value τ_E is at most $2^{-\mu}$. Therefore

$$\Pr[E_2] - \Pr[E_3] \leq 2^{-\mu}$$

Game 4. In this game we replace the KDF_n function used by the server instance π_j^t to compute K_c^e, $K_c^{i_1}$ and $K_c^{i_2}$ with a random function $\mathsf{F}^{\mathsf{KDF}_c}_{MK_1}$. We do the same for any client instance that uses the KDF_n function with the same master key MK_1 as π_j^t in order to compute $K_c^{i_1}$ and $K_c^{i_2}$. We use the fact that the master key MK_1 is uniformly drawn at random. Therefore distinguishing Game 3 and Game 4 implies an algorithm able to distinguish the KDF_n function from a random function. Therefore

$$\Pr[E_3] - \Pr[E_4] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{KDF}_c} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$$

Game 5. In this game we replace the KDF_a function used by π_j^t to compute K_a^e with a random function $\mathsf{F}^{\mathsf{KDF}_a}_{MK_2}$. We do the same for any client instance that uses the KDF_a function with the same master key MK_2 as π_j^t in order to compute K_a^e. We use the fact that the master key MK_2 is uniformly drawn at random. Therefore distinguishing Game 4 and Game 5 implies an algorithm able to distinguish the KDF_a function from a random function. Hence

$$\Pr[E_4] - \Pr[E_5] \leq \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{KDF}_a} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}}$$

Game 6. In this game, the challenger aborts if the server instance π_j^t ever receives a valid message RekeyInd but there exists no client instance having a matching conversation to π_j^t (i.e., sharing the same transcript of exchanged messages so far) that has output that message. The keys K_c^e, $K_c^{i_1}$, $K_c^{i_2}$, and (optionally) K_a^e are used to compute the RekeyInd message. Since the keys K_c^e, $K_c^{i_1}$, $K_c^{i_2}$, and K_a^e are uniformly drawn at random, due to Game 4 and Game 5, we can reduce the possibility of forging a RekeyInd message to the sAE-security of the $\mathsf{StAE}_{\mathsf{client}}$ function used to compute the message. Hence

$$\Pr[E_5] - \Pr[E_6] \leq \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}}$$

To that point, the only way for an adversary to make the server instance π_j^t accept maliciously is to send a RekeyInd message different from all the messages sent by all the client instances, such that RekeyInd is valid. However, in such a case, the challenge aborts. Therefore $\Pr[E_6] = 0$.

Collecting all the probabilities from Game 0 to Game 6, we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} &= \Pr[E_0] \\
&= q \cdot n_{\mathsf{S}} \cdot \Pr[E_1] \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_2] + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_3] + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_4] + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_5] + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&\leq q \cdot n_{\mathsf{S}} \left(\Pr[E_6] + \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&= q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right)
\end{aligned}$$

Therefore we have that

$$\begin{aligned}
\mathsf{adv}^{\mathsf{EA}}_{P} &\leq \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{client}} + \mathsf{adv}^{\mathsf{EA}}_{P,\mathsf{server}} \\
&\leq q \cdot n_{\mathsf{C}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + 2^{-\mu}(1 - 2^{-\beta}) + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\quad + q \cdot n_{\mathsf{S}} \left(\mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + 2^{-\mu} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} \right) \\
&\leq q \Big[n_{\mathsf{S}} \cdot \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{client}}} + n_{\mathsf{C}} \cdot \mathsf{adv}^{\mathsf{sAE}}_{\mathsf{StAE}_{\mathsf{server}}} + (n_{\mathsf{C}} + n_{\mathsf{S}}) \left(\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2\mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} \right) \\
&\qquad + n_{\mathsf{C}} \cdot \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + 2^{-\mu} \left(n_{\mathsf{C}}(1 - 2^{-\beta}) + n_{\mathsf{S}} \right) \Big]
\end{aligned}$$

In addition, we have also

$$\begin{aligned}
\Pr[\text{forgery Join Request}] &\leq p_{jr} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + 2^{-\mu} \\
\Pr[\text{forgery Join Accept}] &\leq p_{ja} = \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{AES}} + \mathsf{adv}^{\mathsf{PRF}}_{\mathsf{MAC}} + \mathsf{adv}^{\mathsf{PRP}}_{\mathsf{AES}} + 2^{-\mu}(1 - 2^{-\beta})
\end{aligned}$$

Channel Security. Let $\mathsf{adv}^{\mathsf{CS}}_{P}$ be the advantage of the adversary in winning the channel security experiment against an instance of some client or server party. That is $\mathsf{adv}^{\mathsf{CS}}_{P} = \left| \Pr[\pi^s_i.\mathsf{b} = b] - \frac{1}{2} \right|$, where (π^s_i, b) is the tuple output by the adversary when it terminates the CS game. Let E_i be the event that the adversary wins in Game i, and $\mathsf{adv}_i = \Pr[E_i] - \frac{1}{2}$.

Game 0. This game corresponds to the CS game of the 2-party protocol P. Therefore

$$\Pr[E_0] = \frac{1}{2} + \mathsf{adv}^{\mathsf{CS}}_{P} = \frac{1}{2} + \mathsf{adv}_0$$

Game 1. In this game, the challenger aborts and chooses a bit b uniformly at random if there exists an instance of some client or server party that accepts maliciously. Hence we have

$$\mathsf{adv}_0 \leq \mathsf{adv}_1 + \mathsf{adv}^{\mathsf{EA}}_{P}$$

Game 2. So far, for any client instance π_i^s (resp. server instance π_i^s), ending in accepting state, there is a unique server instance π_j^t (resp. client instance π_j^t) that is partnered with π_i^s.

In this game, the challenger aborts the experiment if she does not guess which instance is targeted by the adversary. Therefore

$$\mathsf{adv}_2 = \mathsf{adv}_1 \times \frac{1}{q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}}}$$

So far, the adversary knows the instance he targets. That is he knows the indices i, s corresponding to some client or server party, and its instance π_i^s.

Game 3. In this game we replace the KDF_c function used to compute the session keys K_c^e, $K_c^{i_1}$, $K_c^{i_2}$ with a random function $\mathsf{F}_{MK_1}^{\mathsf{KDF}_c}$. We replace also the KDF_a function used to compute the session key K_a^e with a random function $\mathsf{F}_{MK_2}^{\mathsf{KDF}_a}$. We do the same for any instance that uses the functions KDF_c with the same master key MK_1, and KDF_a with the same master key MK_2 as π_i^s in order to compute these session keys. We use the fact that the master keys MK_1 and MK_2 are uniformly drawn at random. Therefore distinguishing Game 2 and Game 3 implies an algorithm able to distinguish the functions KDF_c and KDF_a from random functions. Therefore

$$\mathsf{adv}_2 \leq \mathsf{adv}_3 + \mathsf{adv}_{\mathsf{KDF}_c}^{\mathsf{PRF}} + \mathsf{adv}_{\mathsf{KDF}_a}^{\mathsf{PRF}} = \mathsf{adv}_3 + 2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{PRF}}$$

Game 4. In this game we construct an adversary $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) against the sAE-security of the underlying authenticated encryption scheme used by the client (resp. server) to encrypt and MAC the messages. We use the fact that a random function $\mathsf{F}_{MK_1}^{\mathsf{KDF}_c}$ is used to compute the session keys K_c^e, $K_c^{i_1}$, $K_c^{i_2}$, and a random function $\mathsf{F}_{MK_2}^{\mathsf{KDF}_a}$ is used to compute the session key K_a^e. Therefore these keys are random. The adversary $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) is built on an adversary $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$) able to win the CS experiment against a client (resp. server) instance. $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) forwards any $\mathsf{Encrypt}(\pi_i^s, *)$ query to $\mathsf{Encrypt}(*)$, and sends the response to $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$). It forwards any $\mathsf{Decrypt}(\pi_j^t, *)$ query to $\mathsf{Decrypt}(*)$, and sends the response to $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$). Otherwise $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) behaves as the challenger in Game 3. Therefore we have

$$\mathsf{adv}_4 = \mathsf{adv}_3$$

If $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$) outputs a tuple (π_i^s, b), then $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) forwards b to its sAE challenger. Otherwise $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) flips a bit at random and sends it to its challenger. The probability for $\mathcal{B}_{\mathsf{client}}$ (resp. $\mathcal{B}_{\mathsf{server}}$) to find the correct value b is at least the probability for $\mathcal{A}_{\mathsf{client}}$ (resp. $\mathcal{A}_{\mathsf{server}}$) to win the CS experiment. Moreover, by assumption, the advantage of an attacker in breaking the sAE-security of the authenticated encryption scheme is at most $\mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\mathsf{sAE}}$ (resp. $\mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sAE}}$). Hence

$$\mathsf{adv}_4 \leq \mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\mathsf{sAE}} + \mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sAE}}$$

Collecting all the probabilities from Game 0 to Game 4, we have that

$$\begin{aligned}
\mathsf{adv}_P^{\mathsf{CS}} &= \mathsf{adv}_0 \\
&\leq \mathsf{adv}_1 + \mathsf{adv}_P^{\mathsf{EA}} \\
&= q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \cdot \mathsf{adv}_2 + \mathsf{adv}_P^{\mathsf{EA}} \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_3 + 2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{PRF}}\right) + \mathsf{adv}_P^{\mathsf{EA}} \\
&= q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_4 + 2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{PRF}}\right) + \mathsf{adv}_P^{\mathsf{EA}} \\
&\leq q^2 \cdot n_{\mathsf{C}} \cdot n_{\mathsf{S}} \left(\mathsf{adv}_{\mathsf{StAE}_{\mathsf{client}}}^{\mathsf{sAE}} + \mathsf{adv}_{\mathsf{StAE}_{\mathsf{server}}}^{\mathsf{sAE}} + 2\mathsf{adv}_{\mathsf{AES}}^{\mathsf{PRF}}\right) + \mathsf{adv}_P^{\mathsf{EA}}
\end{aligned}$$

C sAE Security in LoRaWAN 1.1

Here we give a sketch proof of the sAE-security of the AEAD functions $\mathsf{StAE}_{\mathsf{client}}$ and $\mathsf{StAE}_{\mathsf{server}}$ used in LoRaWAN 1.1.

Bellare et al. [7] show that the Encrypt-then-MAC (EtM) construction is IND-CCA and INT-CTXT if the underlying symmetric encryption function is IND-CPA and the underlying MAC function is SUF-CMA. Moreover, Rogaway et al. [47] show that an AEAD encryption scheme that is IND-CCA and INT-CTXT provides AE-security. In addition, the CTR mode is proved IND-CPA by Bellare et al. [5] under the assumption that the block cipher is a good PRF. Iwata et al. [29] show that CMAC is SUF-CMA if the underlying block cipher is a good PRP. Finally we recall that the AEAD encryption schemes used in LoRaWAN 1.1 follow the EtM paradigm. One, $\mathsf{StAE}_{\mathsf{server}}$ used to encrypt downlink messages, is composed of AES-CTR and a tweaked version of AES-CMAC. The second AEAD scheme, $\mathsf{StAE}_{\mathsf{client}}$ used to encrypt uplink messages, is composed of AES-CTR, and a concatenated hash combiner made with a tweaked version of AES-CMAC. Moreover a monotonically increasing counter (embedded in each frame's header) is used to compute the encryption keystream, and involved in the MAC computation. Hence the sAE-security of the AEAD functions $\mathsf{StAE}_{\mathsf{client}}$ and $\mathsf{StAE}_{\mathsf{server}}$.